PCI-X Protocol Addendum to the PCI Local Bus Specification Revision 2.0

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
2.0	Added Mode 2 with 1.5V signaling, source-synchronous clocking of two or four subphases per clock. Added ECC, device ID messaging, and 16-bit interface. Incorporated the PCI-X slot and card identification ECN	7/29/02

The 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 the 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 the PCI-SIG may be forwarded to:

Membership Services
http://www.pcisig.com
E-mail: administration@pcisig.com
Phone: 503-291-2569
800-433-5177 (USA Only)
Fax: 503-297-1090

Technical Support
techsupp@pcisig.com

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. The 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.

All product names are trademarks, registered trademarks, or servicemarks of their respective owners.

Contents

Figures

Tables

Preface

Since the introduction of revision 1.0 of the *PCI-X Addendum to the PCI Local Bus Specification* in 1999, peripheral data rates have continued to grow. Recognizing the need to keep the PCI specifications (the most popular industry standard interface ever developed) ahead of the market need for these peripherals, the PCI-SIG formed a working group in 2001 to double and quadruple the data rates supported by the PCI-X definition. This document, the *PCI-X Protocol Addendum to the PCI Local Bus Specification, Revision 2.0* (PCI-X PT 2.0), and the *PCI-X Electrical and Mechanical Addendum to the PCI Local Bus Specification, Revision 2.0* (PCI-X EM 2.0) are the result of that effort. PCI-X PT 2.0 and PCI-X EM 2.0 are collectively referred to as PCI-X 2.0 and together replace the *PCI-X Addendum to the PCI Local Bus Specification, Revision 1.0b* (PCI-X 1.0b).

To achieve these higher data rates, PCI-X 2.0 adds a source-synchronous clocking mode for the highest-performance burst transactions that enables transfer rates in excess of 4 GB/s. It also defines:

- 1.5V signaling levels for faster signaling and improved signal integrity characteristics
- Error correcting codes (ECC) for improved error immunity
- Expanded Configuration Space address range of 4096 bytes
- A low-pin-count 16-bit interface for embedded applications
- A new device-ID messaging transaction for peer-to-peer communication

Following the well-established tradition of all PCI specifications, PCI-X 2.0 stresses backward compatibility. Devices and cards compliant with PCI-X 1.0b are fully supported in PCI-X 2.0, in what is defined as Mode 1. The higher transfer rates in PCI-X 2.0 are defined as Mode 2. PCI-X devices that support both Mode 1 and Mode 2 are completely interoperable with devices designed to support the previous revision of the PCI-X definition and 33 MHz, 3.3V conventional PCI devices. They optionally support the 66 MHz conventional PCI mode. ECC support, the expanded Configuration Space, and the 16-bit interface are required for Mode 2 devices. Forwarding device ID messages is required for PCI-X Mode 2 bridges. ECC support and device ID messaging are optional for Mode 1 devices.

The new features in PCI-X 2.0 are defined to minimize the changes required when migrating from a PCI-X 1.0b design. Most requirements of PCI-X 1.0b remain unchanged in PCI-X 2.0. Therefore, the new requirements for the new modes defined in PCI-X 2.0 are distributed throughout PCI-X PT 2.0 and PCI-X EM 2.0 in the locations where they make the most sense. The result is a single document that defines the protocol requirements for PCI-X devices ranging from 266 MB/s (32-bit, 66 megatransfers/s) to more than 4 GB/s (64-bit, 533 megatransfers/s). A separate document defines the electrical and mechanical requirements for the same range of devices.

Future Changes

Following publication of PCI-X PT 2.0, 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

1. Introduction

The *PCI-X Protocol Addendum to the PCI Local Bus Specification, Revision 2.0* (PCI-X PT 2.0), together with the *PCI-X Electrical and Mechanical Addendum to the PCI Local Bus Specification, Revision 2.0* (PCI-X EM 2.0), define enhancements to the *PCI Local Bus Specification, Revision 2.3* (PCI 2.3), the *PCI to PCI Bridge Architecture Specification, Revision 1.1* (PCI Bridge 1.1), the *PCI Power Management Interface Specification, Revision 1.1* (PCI PM 1.1), and the *PCI Hot-Plug Specification, Revision 1.1* (PCI HP 1.1), which are the latest versions of these specifications at the time of release of this document. Contact the PCI-SIG for any later revisions.

PCI-X PT 2.0 and PCI-X EM 2.0 are collectively referred to as PCI-X 2.0 and together replace the *PCI-X Addendum to the PCI Local Bus Specification, Revision 1.0b* (PCI-X 1.0b).

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 the 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.

---

[1] This specification follows the precedent of PCI 2.3 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 2.1.2.4.1, "Clock Specification," in PCI-X EM 2.0.

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

PCI-X 2.0 defines two modes of operation. Mode 1 is the mode of operation defined in previous versions of the PCI-X definition.

The following features defined in PCI-X 2.0 are required for Mode 2 devices and optional for Mode 1 devices:

1. An error correcting code (ECC) replaces parity.
2. The fastest target device decode speed in ECC mode is one clock slower than parity mode. This allows time for the target to check and correct errors in the transaction address before responding.
3. A previously reserved transaction command is defined to be the Device ID Message command.
4. A new clock jitter class is added that reduces the absolute minimum clock period to allow for jitter.
5. Interrupt mask and status bits defined in PCI 2.3.

Mode 2 also includes the following features:

1. Almost all of the Mode 1 control signal and transaction phase rules have been kept the same in Mode 2 to minimize design changes.
2. The initiator of transactions that use the highest performance commands drives data at two or four times the common clock frequency. These cases use source-synchronous clocking techniques for transferring the data, rather than capturing data with the common clock.
3. The Configuration Space addresses are extended from eight to 12 bits, enabling the addressing of 4096 bytes for each function of each device.
4. A low-pin-count 16-bit interface for embedded applications.
5. Electrical and logical changes have been made to improve the signal integrity and reduce electrical noise. Signaling voltages are lowered to 1.5V for the source-synchronous

signals. (The common-clock control signals remain at 3.3V.) The system switches the $V_{I/O}$ supply voltage between 3.3V or 1.5V depending upon whether the bus is operating in Mode 1 or Mode 2 respectively. Input buffers for the source-synchronous signals include terminating resistors.

6. The signal terminating voltage for the source-synchronous signals is centered around the receiver threshold, and thus requires that receivers never be enabled when no driver is driving the signal. The lower buses are generally driven all the time, except during one-clock bus turn-around.

Table 1-1 compares maximum transfer rates and signaling levels among all the conventional and PCI-X modes.

Table 1-1: Conventional PCI and PCI-X Comparison Summary

		Maximum Transfer Rate (MB/s)		
Mode	**Signaling Level**	**64-Bit Bus**	**32-Bit Bus**	**16-Bit Bus**
Conventional PCI 33 (ref)	5V or 3.3V	266	133	na
Conventional PCI 66 (ref)	3.3V	533	266	na
PCI-X 66	3.3V	533	266	na
PCI-X 133	3.3V	1066	533	na
PCI-X 266	1.5V and 3.3V	2133	1066	533
PCI-X 533	1.5V and 3.3V	4266	2133	1066

Table 1-2 shows the supported combinations of modes and features for PCI-X.

Table 1-2: Supported PCI-X Modes and Features

Feature	**PCI-X Mode 1**	**PCI-X Mode 2**
Device types	PCI-X 66 PCI-X 133	PCI-X 266 PCI-X 533
Bus width	64 bit optional 32-bit required	64 bit optional 32-bit: Required for add-in card applications. Optional for embedded applications 16-bit required
Interface Signaling	3.3V	1.5V and 3.3V
Error protection	Parity required ECC optional	ECC required
Data capture	Common-clock	Common-clock and source-synchronous
12-bit Configuration Space address	Not supported	Required
Device ID messages	Optional	Required

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 2.3, PCI Bridge 1.1, PCI PM 1.1, and PCI HP 1.1. 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 are required to support a 32-bit data bus. They optionally support a 64-bit data bus.
2. Address and data are multiplexed on the same bus.
3. 64- and 32-bit 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, except a new identification requirement (see Section 3.2, "Identification Requirements," in PCI-X EM 2.0). PCI-X Mode 1 add-in cards are keyed for 3.3V or Universal signaling. PCI-X Mode 2 add-in cards are keyed exclusively for 3.3V signaling.
7. Maximum power consumption for add-in cards.
8. 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. In PCI-X Mode 1, electrical signaling voltage levels are the same as conventional 3.3V signaling, except V_{il}(max) is slightly higher in PCI-X mode to provide additional noise margin.
5. Device and add-in card electrical specification ranges are generally narrower for PCI-X devices.
6. In PCI-X Mode 1 and PCI-X Mode 2 common-clock transfers, there is a single clock signal for all devices and transactions. The bus changes state and data transfers on the rising edge of the clock. In PCI-X Mode 2 source-synchronous transfers, the device driving data also drives strobes used for latching the data.
7. In parity mode, signal names and add-in card connector pin-out remains unchanged (except two new pins, PCIXCAP and MODE2). In ECC mode, new pins are added to support ECC on each phase of the transaction.

8. Power supply voltages remain unchanged, except $V_{I/O}$ switches to 1.5V in PCI-X Mode 2.

1.1. Documentation Conventions

In addition to the documentation conventions established in PCI 2.3, 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 2.3, 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 2.3, 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 data transfer rate.

Conventional PCI 33

Conventional PCI 66

PCI-X 66

PCI-X 133

PCI-X 266

PCI-X 533

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

Clock numbering

As in PCI 2.3, this document designates the clock that a particular event occurs based on its appearance on the bus. For example, if as a result of a particular clock edge, N, a device changes its output state to begin asserting a signal, that device is said to "assert the signal on clock N+1" or the signal is said to be "asserted on clock N+1."

In PCI-X Mode 2, each source-synchronous data phase contains multiple subphases. For these data phases, a device is said to "drive data between clocks N and N+1" even if the actual output delay is such that one or more subphases are delayed past clock N+1.

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 driven on the bus during the attribute phase(s) of a PCI-X transaction. Used for further definition of the transaction.

attribute phase — The clock after the address phase(s).

burst push transaction — A burst transaction for which the initiator is the source of the data. Burst push transactions use one of the following commands:

Memory Write
Memory Write Block
Alias to Memory Write Block
Split Completion
Device ID Message

In PCI-X Mode 2, all burst push transactions except Memory Write transactions use source-synchronous clocking. No other transactions use source-synchronous clocking.

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
Device ID Message

Burst transactions can generally 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.) On 64-bit buses, they are permitted to be initiated both as 64-bit and 32-bit transactions.

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.)

common clock

A data transfer method in which the devices that are the source and destination of the data use a single centrally generated clock. See also "source synchronous."

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.

The Completer ID appears in the address phase of a Device ID Message transaction and is used to route the transaction to the completer.

The Completer ID also appears in the attribute phase of Split Completion transactions. 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.

See also "Requester ID."

correctable ECC error

An error interpreted by the ECC mechanism as the result of the inversion of a single bit. See Section 5.1.2.1.

data phase Each clock in which the target signals some kind of data transfer or terminates the transaction. Clocks in which the target signals Wait State one or more times and then signals something else are part of the same data phase. See also "data subphase."

data phase boundary The address of the first byte within each data phase of a burst transaction.

Mode	64-Bit Transfer	32-Bit Transfer	16-Bit Transfer
PCI-X 66	AD[2::0]=000b	AD[1::0]=00b	AD[0]=0b
PCI-X 133	AD[2::0]=000b	AD[1::0]=00b	AD[0]=0b
PCI-X 266	AD[3::0]=0000b	AD[2::0]=000b	AD[1::0]=00b
PCI-X 533	AD[4::0]=0 0000b	AD[3::0]=0000b	AD[2::0]=000b

The starting address and ending address of the transaction are independent of data phase boundaries. See Section 2.3.

data subphase One of multiple data transfer times within a single data phase. Data subphases use source-synchronous clocking and are used only in PCI-X Mode 2 for burst push transactions other than Memory Write.

Mode	Number of Data Subphases per Data Phase
PCI-X 66	none
PCI-X 133	none
PCI-X 266	2
PCI-X 533	4

device A component of a PCI system that connects to a PCI bus. As defined by PCI 2.3, 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 2.3, 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 2.3. 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 PCI 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.

device ID message A Sequence using the Device ID Message command. Device ID message transactions are burst transactions that use the Completer ID (the completer bus number, completer device number, and completer function number) to address the completer. They are intended for direct peer-to-peer communication between devices.

DIM Address Information driven by the requester or a bridge on the AD bus during the address phase of a transaction using the Device ID Message command. The DIM Address includes the Route Type and Completer ID and is used to route the device ID message to the completer.

DIM Attributes Format of the attributes of transactions using the Device ID Message command. Includes information about the message and the requester of the message.

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).

drive When a device has acquired exclusive ownership of a bus through a handshake with the bus arbiter or the initiator of a transaction, and when the target of a read transaction has reached the appropriate point in the read 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 Four bytes 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 on 64- and 32-bit buses 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.

ECC Error correcting code. In ECC mode, each address, attribute, common-clock data phase, and source-synchronous data subphase include additional bits that are used to correct (if enabled) any single bit errors and detect any double bit errors. See Section 5.1.2.

ECC mode The mode of operation of the bus that uses ECC error protection. A bus operating in PCI-X Mode 1 operates either in parity mode or ECC mode, as determined by a bit in the ECC Control and Status register. A bus operating in PCI-X Mode 2 always operates in ECC mode.

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.

parity mode The mode of operation of the bus that uses parity error protection. A bus operating in PCI-X Mode 1 operates either in parity mode or ECC mode, as determined by a bit in the ECC Control and Status register. A bus operating in PCI-X Mode 2 never operates in parity mode.

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 1 (with parity or ECC protection) or Mode 2 at the rising edge of RST#. Also indicates the range of frequency of the clock. See Section 6.2.

PCI-X Mode 1 A mode of operation compliant with revisions of this document prior to PCI-X 2.0. It includes:

- PCI-X 66 and PCI-X 133 modes
- Data transfers always use common clock
- 3.3V signaling levels
- Optional ECC (added in PCI-X 2.0)

Devices compliant with revision 1.0 of this specification are automatically compliant with this revision in PCI-X Mode 1.

Sometimes abbreviated as "Mode 1" when the meaning is clear from the context.

When used as an adjective, for example "Mode 1 device," "Mode 1 add-in card," "Mode 1 system," or "Mode 1 slot," it defines an object whose highest operating mode is either PCI-X 66 or PCI-X 133, whether or not that object is actually operating in PCI-X Mode 1.

PCI-X Mode 2 A mode of operation compliant with PCI-X 2.0 that includes:

- PCI-X 266 and PCI-X 533 modes
- Burst push transactions other than Memory Write use source-synchronous clocking
- 1.5V signaling levels for source-synchronous signals
- ECC

All features not defined in revisions of this document prior to PCI-X 2.0 apply in PCI-X Mode 2. (Some are optionally available in Mode 1, also.)

Sometimes abbreviated as "Mode 2" when the meaning is clear from the context.

When used as an adjective, for example "Mode 2 device," "Mode 2 add-in card," "Mode 2 system," or "Mode 2 slot," it defines an object whose highest operating mode is either PCI-X 266 or PCI-X 533, whether or not that object is actually operating in PCI-X Mode 2.

QWORD Eight bytes 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 DWORD and all read transactions that cross it 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, Type 0 configuration, and device ID message transactions. See also "Completer Attributes" and "DIM 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 Sequence. 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.

source synchronous A data transfer method in which the device that is the source of the data also drives strobes to be used for latching the data at the destination. Source-synchronous transactions use source-synchronous sampling for the AD and ECC buses in PCI-X Mode 2 for data phases of burst push transactions other than Memory Write. They use common-clock sampling for the AD and ECC buses during all other data phases and all address and attribute phases. They use common-clock sampling for the control signal during all phases. See also "common clock."

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 DWORD transaction (except Special Cycle) and any read 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, Special Cycles, and Device ID Messages. This is the first byte affected by 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.

system — Any functional combination of a system board, add-in cards, and/or software.

system board — A collection of hardware that includes a CPU, host bridge, and a PCI bus. In some system boards, the bus includes one or more devices in addition to the source bridge. In some system boards the bus connects to one or more add-in card slots.

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.

uncorrectable data error An uncorrectable error that occurred in a data phase or subphase.

uncorrectable error A parity error in PCI-X parity mode (and conventional PCI) or a detected multiple-bit ECC error (as described in Section 5.1.2.1) in ECC mode. (Correctable ECC errors are also treated as uncorrectable if error correction is disabled. See Sections 7.2.3, 8.6.2.3, and 8.6.2.4.)

1.3. Figure Legend

The following conventions for the state of a signal at clock N are used in PCI-X Mode 1 timing diagrams throughout this document.

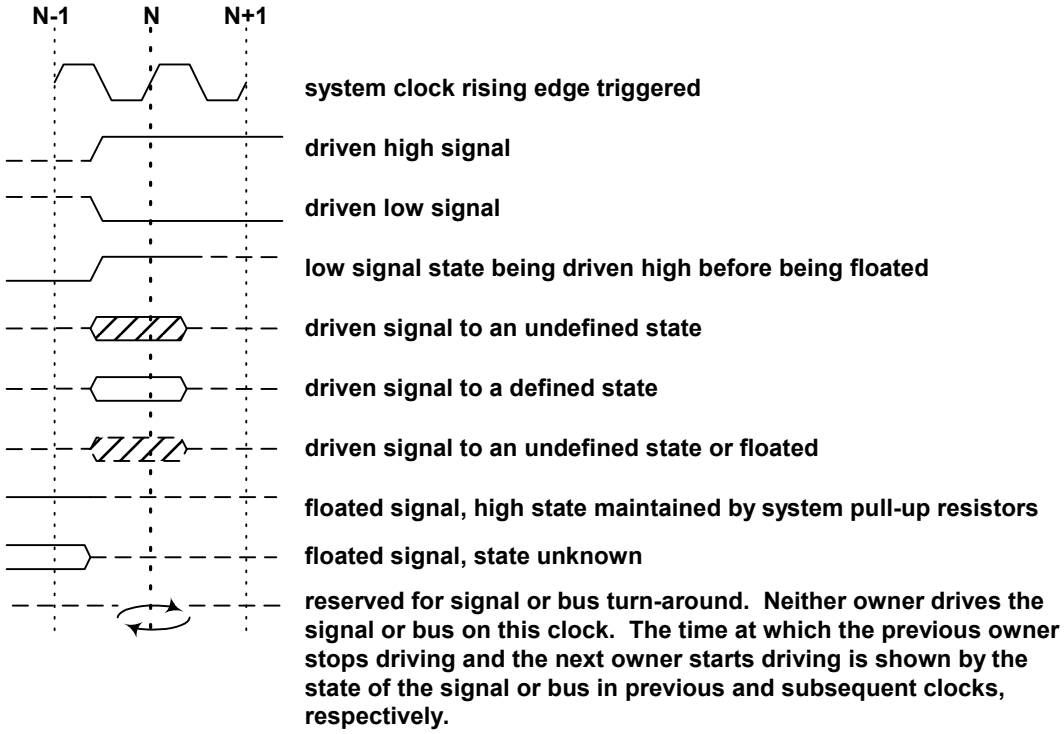

Figure 1-1: Figure Legend, Mode 1

The following conventions for the state of a signal at clock N are used in PCI-X Mode 2 timing diagrams throughout this document.

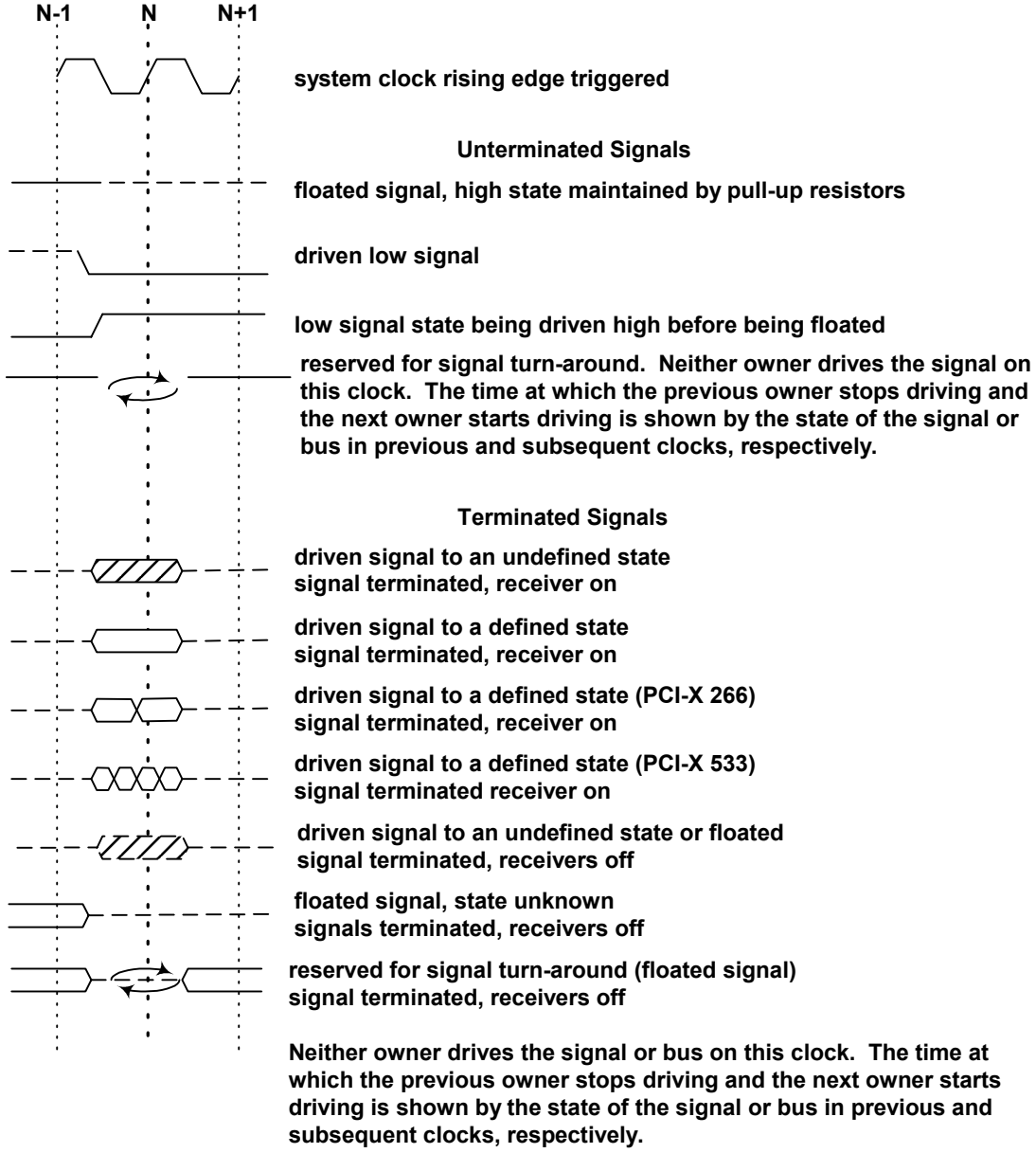

Figure 1-2: Figure Legend, Mode 2

In most cases, Mode 2 transaction diagrams show the bus idle on the clock before the transaction starts. If the bus were not idle, the clock immediately before the start of the transaction would be a bus turn-around cycle. Most Mode 2 transaction figures also show a bus turn-around cycle at the end of the transaction and a subsequent transaction from another bus owner starting as soon as possible. If no other transactions are available to execute, the bus would be idle after the bus turn-around cycle and driven by the parked owner, as specified in Section 4.1.2.

1.4. Comparison of Transactions in Different Bus Modes

The following illustrations compare examples of typical write transactions in the following modes:

- Conventional PCI
- PCI-X PCI-X 66 or PCI-X 133 (Mode 1), 64- or 32-bit bus
- PCI-X 266 (Mode 2), 64- or 32-bit bus
- PCI-X 533 (Mode 2), 64- or 32-bit bus
- PCI-X 266 (Mode 2), 16-bit bus
- PCI-X 533 (Mode 2), 16-bit bus

All the illustrations depict write transactions with initiator termination, identical wait states, and six data phases. The conventional PCI and PCI-X Mode 1 illustrations show identical device select timing. The PCI-X Mode 2 illustrations show one additional clock for device select timing, which is the fastest timing allowed for transactions that support ECC (Mode 2 requires ECC support).

Figure 1-3 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.

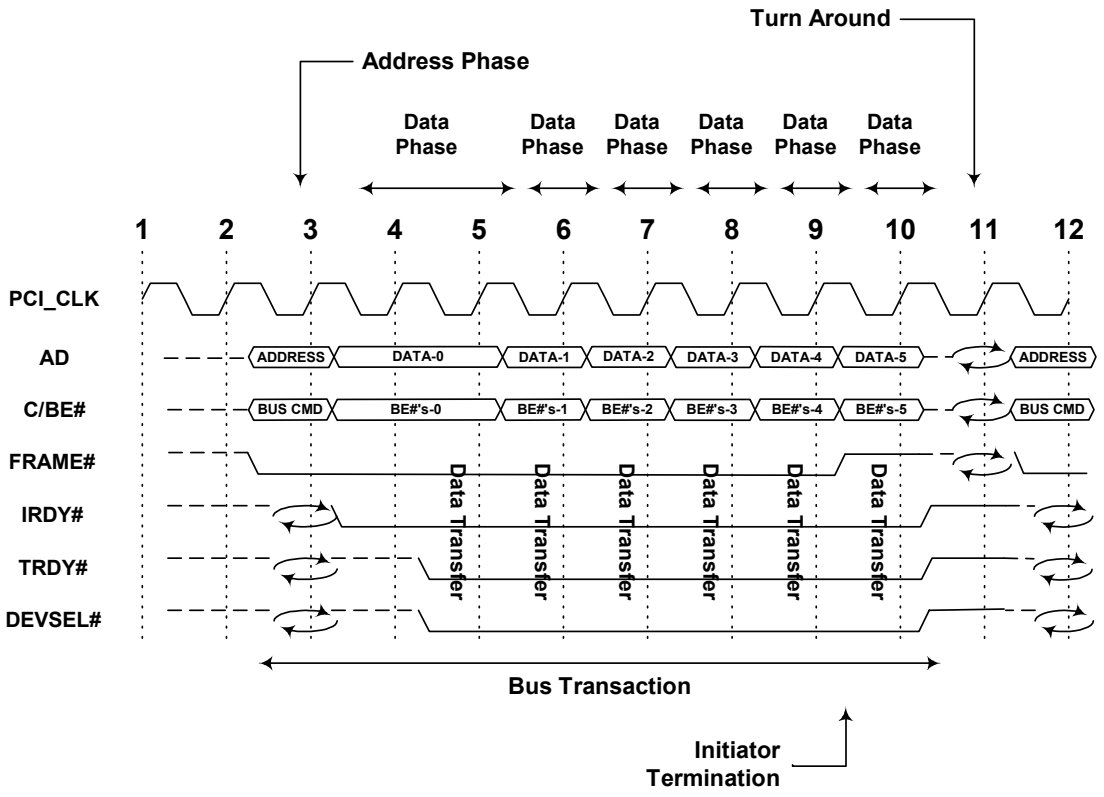

Figure 1-3: Typical Conventional PCI Write Transaction

Table 1-3 lists the phase definitions for conventional PCI bus transactions as shown in Figure 1-3.

Table 1-3: 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 initial 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 1-4 shows a typical PCI-X Mode 1 write transaction using parity protection, with six data phases. The initiator and target insert minimum wait states for DEVSEL# timing A (not allowed in ECC mode), and the transaction completes in ten clocks including the bus turn-around.

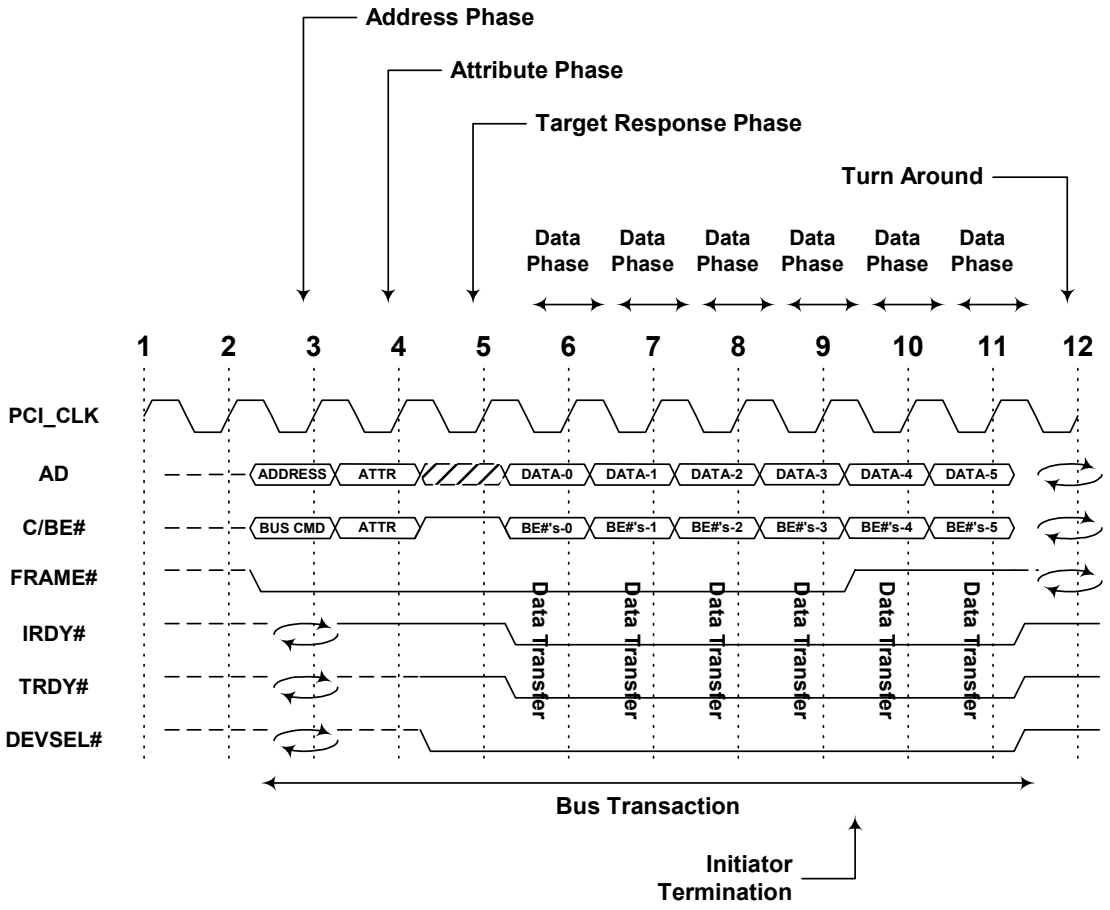

Figure 1-4: Typical PCI-X Mode 1 Write Transaction

Figure 1-5 shows a typical 64- or 32-bit PCI-X 266 source-synchronous transaction and Figure 1-6 shows a typical 64- or 32-bit PCI-X 533 source-synchronous transaction, which are the two maximum transfer rates for PCI-X Mode 2. Both figures show write transactions with six data phases. Each data phase of the PCI-X 266 transaction contains two data subphases and each data phase of the PCI-X 533 transaction contains four data subphases, so each of these data phases transfers two and four times (respectively) as much data as in the previous two examples. As in the PCI-X Mode 1 example, the initiator and target insert minimum wait states. In PCI-X Mode 2, the fastest allowable DEVSEL# decode speed is B, to allow the target to correct ECC errors in the address. Note that the C/BE# signals are used as data transfer strobes during source-synchronous data phases. See Section 2.1.3.5.2, "1.5V Environment Source-Synchronous Timing Parameters," in PCI-X EM 2.0 for details.

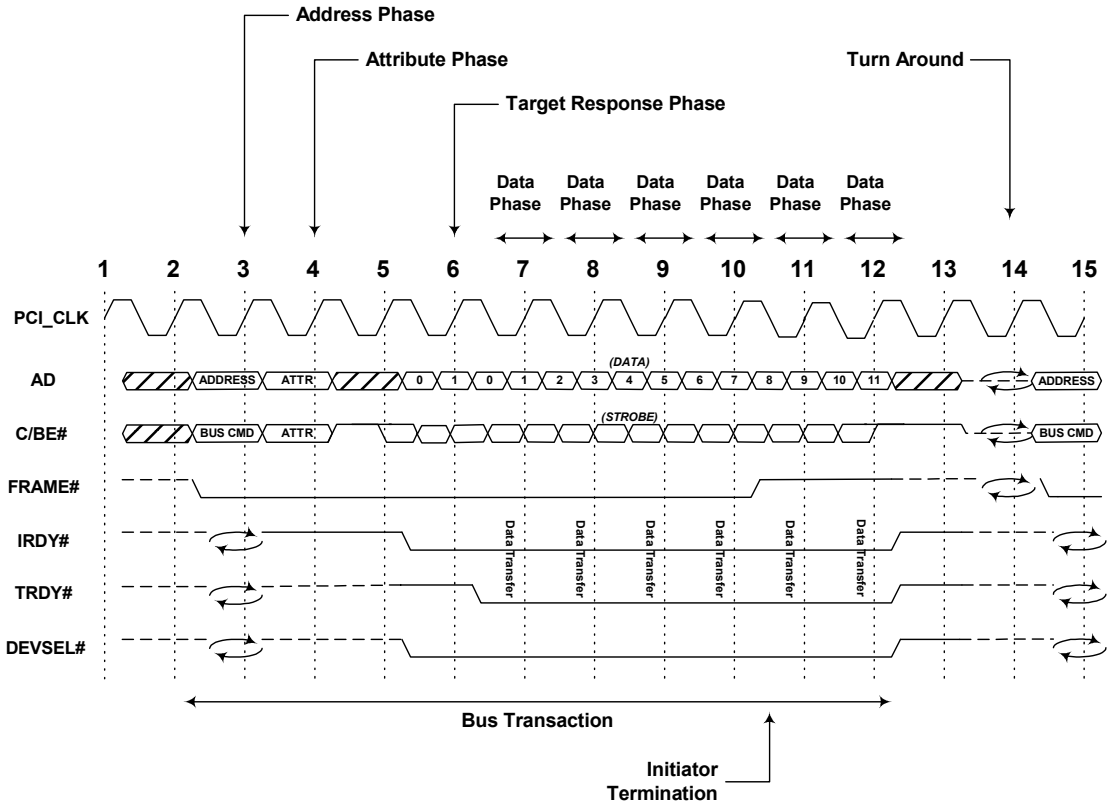

Figure 1-5: Typical PCI-X 266 (Mode 2) Burst Write Transaction

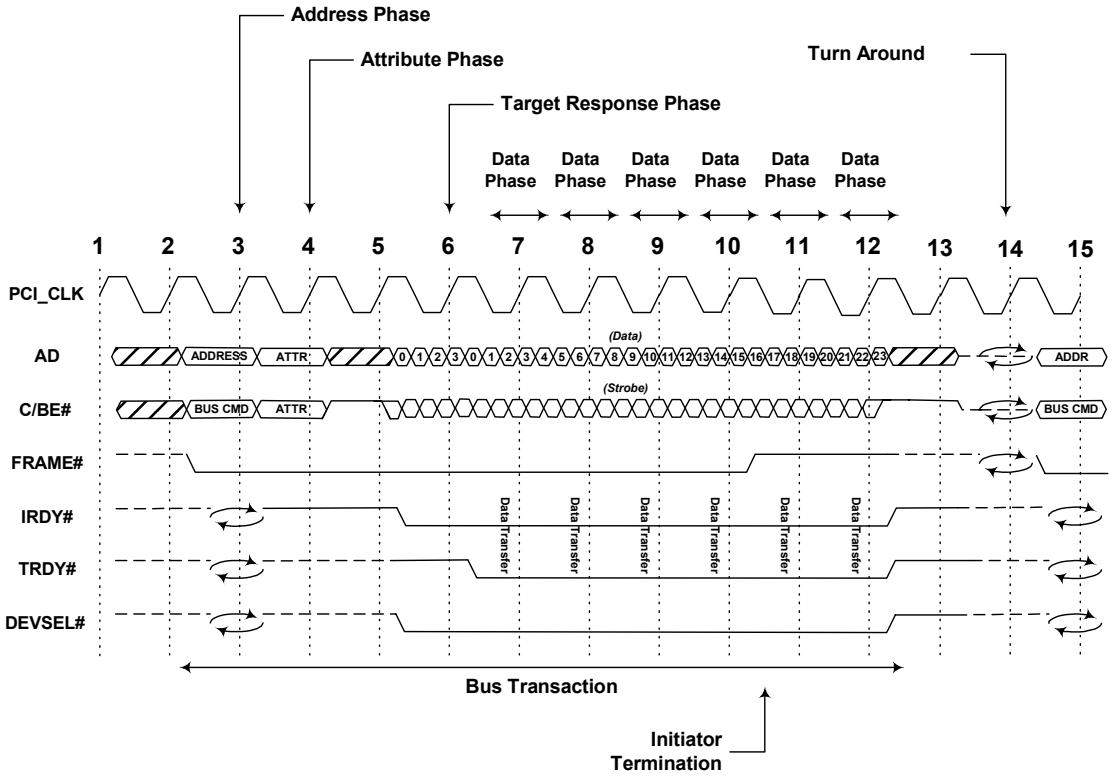

Figure 1-6: Typical PCI-X 533 (Mode 2) Burst Write Transaction

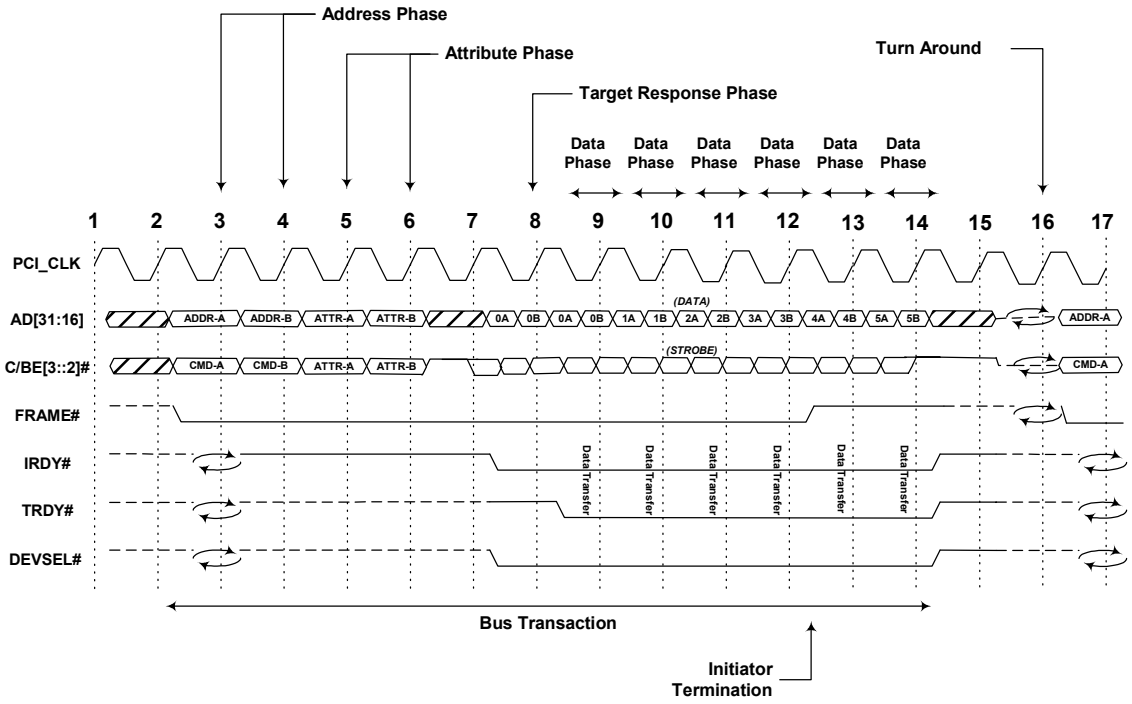

Figure 1-7: Typical 16-Bit PCI-X 266 (Mode 2) Burst Write Transaction

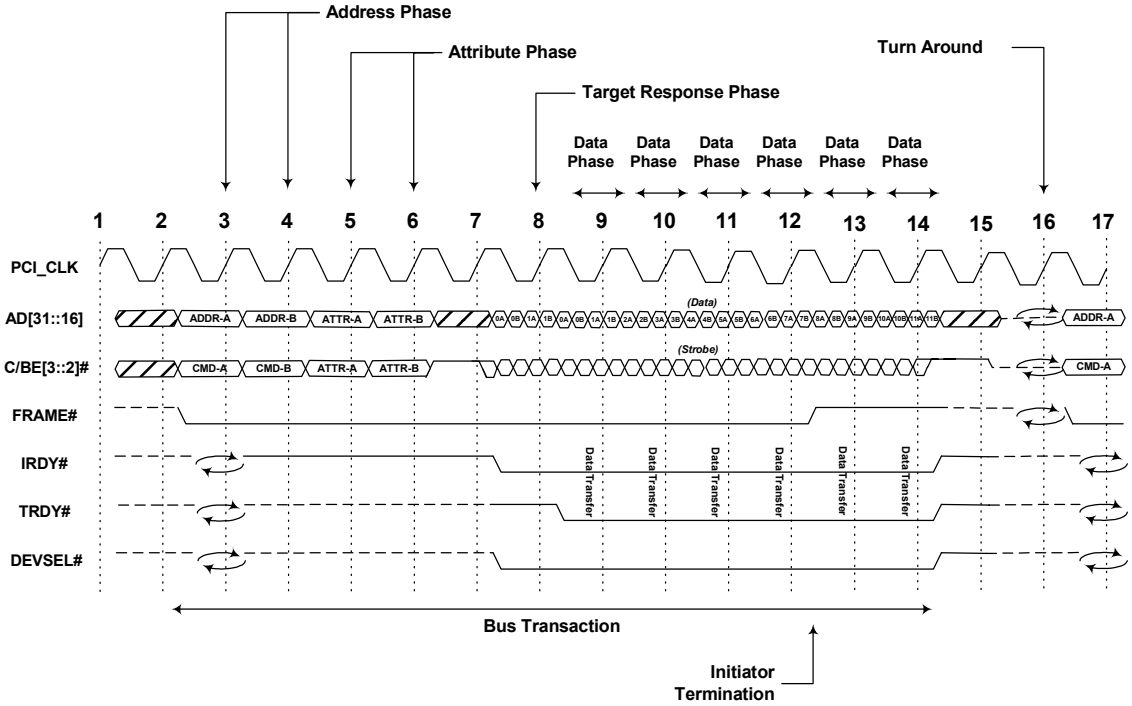

Figure 1-8: Typical 16-Bit PCI-X 533 (Mode 2) Burst Write Transaction

Figure 1-7 shows a typical 16-bit PCI-X 266 transaction and Figure 1-8 shows a typical 16-bit PCI-X 533 transaction. Both figures show write transactions with the same DEVSEL# decode timing, wait states, and number of data phases as the previous two Mode 2 examples. Notice that in 16-bit transactions, there are twice as many address and attribute phases as in 64- or 32-bit transactions.

Table 1-4 lists the transaction phases of PCI-X as shown in Figure 1-4 through Figure 1-8.

Table 1-4: PCI-X Transaction Phase Definitions

PCI-X Transaction Phases	Description
Address Phase	One clock for single address cycle, two clocks for dual address cycle for 64- and 32-bit buses. Twice that many for 16-bit buses. (See Section 2.12.1.1 for more information about dual address cycles. See Section 2.12.2 for more information about 16-bit buses.)
Attribute Phase	One clock for 64- and 32-bit buses, two clocks for 16-bit buses. 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.

The previous figures illustrate the similarities among conventional PCI, and PCI-X Mode 1 and Mode 2 (64-, 32-, and 16-bits wide). 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 the conventional PCI mode (Figure 1-3) and the parity-protected PCI-X Mode 1 (Figure 1-4) 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 parity-protected PCI-X Mode 1. The fastest allowable DEVSEL# decode speed in PCI-X Mode 1 with optional ECC protection and in PCI-X Mode 2 (which requires ECC protection) is one clock longer, to allow time for the device to correct ECC errors in the address. Mode 2 transactions also require one additional idle clock between transactions. So those transactions take 12 clocks. 16-bit transaction require twice as many clocks for the address and attribute phases, so those transactions take 14 clocks.

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), Split

Completion, and Device ID Message 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- or 16-bit transaction width (REQ64# must be deasserted).

Table 1-5 compares burst and DWORD transactions.

Table 1-5: 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 Device ID Message	Commands: Interrupt Acknowledge Special Cycle I/O Read I/O Write Configuration Read Configuration Write Memory Read DWORD
64- or 32- or 16-bit data transfers.	32- or 16-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 exactly one data phase for 32-bit transfer and exactly two data phases for 16-bit transfer.
In PCI-X Mode 1, the C/BE# bus contains valid byte enables for data phases for Memory Write transactions. Any byte enable pattern is permitted (between the starting and ending address, inclusive), including no byte enables asserted. In all other cases, the C/BE# bus is reserved and driven high by the initiator during data phases. In PCI-X Mode 2, the C/BE# bus contains valid byte enables for data phases for Memory Write transactions and is used for data strobes for data phases of source-synchronous transactions. In all other cases the C/BE# bus is reserved and driven high by the initiator during data phases.	During the attribute phase, the Requester Attributes contain valid byte enables. Any byte enable pattern is permitted, including no byte enables asserted. (See Section 2.3 for limitations on byte enable patterns for certain starting addresses.) During the data phase, the C/BE# bus is reserved and driven high by the initiator.

The starting address and byte count of some burst transactions are such that they have only a single data phase.

A Split Completion is a burst transaction, even if the Split Request was a DWORD transaction.

In PCI-X Mode 2, transactions using the Memory Write and Memory Read Block commands use common clock techniques to transfer data, the same as Mode 1. Other burst transactions use source-synchronous clocking techniques at a rate faster than the common clock. See Section 2.6 for details.

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 burst push transactions and 16-bit DWORD write transactions must come in pairs for transactions that successfully transfer data. In ECC mode, a 32-bit target of a burst push transaction from a 64-bit initiator must insert a minimum of two wait states 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 DWORD transactions (except Special Cycle) and all read transactions are permitted to be executed as Split Transactions. If a target cannot complete one of these transactions 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 DWORD or read transaction (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 required to support a 32-bit interface. They optionally support a 64-bit interface.

2. Addresses greater than 4 GB and certain device ID message transactions require a dual address cycle on 64- and 32-bit buses.
3. Data transfer width on 64-bit buses is negotiated for each transaction. The initiator and target are permitted to be either 64 or 32 bits wide 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.
3. In ECC mode, the upper bus is only used during 64-bit data phases. See Section 2.12.1.3 for details.
4. All Mode 2 devices (other than bridges) are required to support a 16-bit interface. The 16-bit interface is intended for embedded applications, so no width negotiation protocol or add-in card connector pin-out is specified, and bridge hierarchies are not allowed. The 16-bit bus always operates in Mode 2, and all transactions are 16 bits wide.

1.9. Compatibility and System Initialization

This document defines two PCI-X Mode 1 devices, PCI-X 66 and PCI-X 133, and two PCI-X Mode 2 devices, PCI-X 266 and PCI-X 533. Both kinds of PCI-X Mode 1 devices have identical requirements except for electrical differences specified in Section 2.1.2, "3.3V Signaling Environment," in PCI-X EM 2.0 such as the minimum clock period (maximum clock frequency) and the add-in card identification requirements specified in Section 3.2.2, "Add-in Card Requirement," in PCI-X EM 2.0. PCI-X 66 devices operate in PCI-X Mode 1 with clock frequencies from 50 MHz to 66 MHz. PCI-X 133 devices operate in PCI-X Mode 1 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 1 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 1 and the clock operates up to 66 MHz.

Both kinds of PCI-X Mode 2 devices have identical requirements except for electrical differences specified in Section 2.1.3, "1.5V Signaling Environment," in PCI-X EM 2.0 such as the minimum source-synchronous strobe timing (maximum data transfer rate) and the add-in card identification requirements specified in Section 3.2.2, "Add-in Card Requirement," in PCI-X EM 2.0. Both kinds of PCI-X Mode 2 devices are required to operate with clock frequencies from 50 MHz to 133 MHz. If only PCI-X 533 devices are installed on a bus, the bus operates in PCI-X Mode 2 with four data subphases in each source-synchronous data phase. If all devices on the bus are PCI-X 533 and PCI-X 266, the bus operates in PCI-X Mode 2 with two data subphases per source-synchronous data phase.

All PCI-X Mode 2 devices and systems also support PCI-X 133 mode. All PCI-X 133 devices naturally support PCI-X 66 mode (PCI-X 66 is a natural subset of PCI-X 133). 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 mode, PCI-X Mode 1, or PCI-X Mode 2 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 1 or Mode 2. Since PCI-X Mode 2 uses a different I/O signaling voltage than PCI-X Mode 1, the system switches the I/O supply voltage to the appropriate value when power is applied to the slot. Devices use the value of the I/O supply voltage to select between Mode 2 and Mode 1 I/O buffer characteristics for the Category 1 signals, as defined in Section 2.1.1, "Driver and Receiver Categories," in PCI-X EM 2.0.

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 1.1. 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. Device ID Messaging

Device ID messaging is intended for direct peer-to-peer communication between devices. Device ID message transactions are burst transactions that use the Device ID Message command and use explicit routing (i.e., the Completer ID—the completer bus number, completer device number, and completer function number) or implicit routing (described in Section 2.16) to address the completer. Bridges forward device ID messages based on the completer bus number. Device ID messages are permitted to be of any length from one byte to 4096 bytes.

1.11. PCI-X Mode 2 Bus Drive and Turn Around

The electrical drive requirements of PCI-X Mode 2 require that all input receivers for signals that have electrical terminators (see Category 1 in Table 2-1, "Driver and Receiver Categories," in PCI-X EM 2.0) must be disabled when the bus is not being driven. The lower 32-bit portion of the buses must be driven at all times that the bus is powered-up and operating, except during precisely defined bus turn-around clocks (see Section 4.1.2). The input receivers for the upper 32-bit portion of the buses are disabled except during 64-bit data phases. Turn-around of the AD bus for data phases of immediate read transactions is also precisely controlled.

1.12. 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.

1.12.1. 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, Special Cycle, or Device ID Message) and the C/BE# bus contains the command. (See Section 2.12.1.1 for dual address cycles.) ECC mode on 64- and 32-bit buses uses only AD[31::00], and C/BE[3::0]# for the address phase. On 16-bit buses, the addresses are driven in twice as many address phases on 16 bits of the AD bus and two bits of the C/BE# bus.
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. Device ID Message transactions use the Completer ID or implicit routing for an address.
3. The attribute phase follows the address phase(s). For 64- and 32-bit transfers, 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 in parity mode, and they are optionally driven in ECC mode. For 16-bit transfers, the attributes are driven in two attribute phases on two bits of the C/BE# bus and 16 bits of the AD bus. The attributes include additional information about the transaction.
4. The C/BE# bus is reserved (driven high) the clock after the attribute phase. In PCI-X Mode 2 source-synchronous transactions, the C/BE# bus begins to toggle as data strobes 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(s) 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 never initiated as 64-bit transactions (REQ64# must be 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 (two for 16-bit transactions). Transactions using the Memory Write, Memory Read Block, Memory Write Block, Alias to Memory Read Block, Alias to Memory Write Block, Split Completion, and Device ID Message commands 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. In PCI-X Mode 2, transactions using the Memory Write Block, Alias to Memory Write Block, Split Completion, and Device ID Message commands have two or four data subphases in each data phase, as determined by the PCI-X initialization pattern.
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. In PCI-X Mode 1, 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. In PCI-X Mode 2, the C/BE# bus is:
        1) Used for source-synchronous data strobes for each data subphase of Memory Write Block, Split Completion, and Device ID Message transactions.
        2) Used for explicit byte enables for data phases of Memory Write transactions.
        3) Reserved and driven high during the data phases of Memory Read Block and all DWORD transactions.
    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 2.12.1.3 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.) PCI-X Mode 2 devices and system must also ignore the assertion of PERR# except when it is asserted the appropriate number of clocks after a data phase as described in Section 5.1.1.3 for parity mode and Section 5.1.2.4 for ECC mode.
13. Like conventional PCI, no device is permitted to drive and receive a bus signal at the same time. (See Section 3.1.)

1.12.2. 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# one or two clocks after GNT# is asserted and the bus is idle. If the transaction uses a configuration command, the initiator must assert FRAME# five or six clocks after GNT# is asserted and the bus is idle. (See Section 2.12.2 for 16-bit buses.)
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 (second attribute phase for 16-bit buses). It deasserts IRDY# 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 is if the transaction has three or more data phases. The second case 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 burst push and all DWORD write transactions in Mode 1 and for Memory Write and all DWORD write transactions in Mode 2, 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 2.12.1.3 for requirements for a 64-bit initiator writing to 32-bit targets.

   For Memory Write Block, Alias to Memory Write Block, Split Completion, and Device ID Message transactions (source-synchronous transactions) in Mode 2, the initiator must:

   a. Drive data strobes on the C/BE# bus as described in Section 2.1.3.5.1, "Source-Synchronous Data Strobes," in PCI-X EM 2.0.
   b. Drive data on the AD bus between the first and second clocks after the attribute phase (second attribute phase for 16-bit buses). The initiator drives the data bus beginning at the first data phase boundary that is less than or equal to the starting address of the transaction and advances through all data subphases of each data phase.
   c. If the transaction is a burst with more than one data phase, the initiator advances to the second data phase value between the first and second clocks after the target asserts DEVSEL# (the same as in Mode 1 except that it includes all data subphases), in anticipation of the target asserting TRDY#.
   d. If the target also inserts wait states, the initiator must toggle between its first and second data phase values (including all data subphases) until the target asserts TRDY# (or terminates the transaction). See Section 2.12.1.3 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 of a transaction that starts less than four data phases from an ADB, 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. In PCI-X Mode 1, 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. In PCI-X Mode 2, the initiator floats the C/BE# bus one clock after it deasserts IRDY#. There are always at least two idle clocks between transactions in Mode 2 (including the bus turn-around).

10. In PCI-X Mode 1, 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. In PCI-X Mode 2, the initiator floats the AD bus one clock after it deasserts IRDY#. There are always at least two idle clocks between transactions in Mode 2 (including the bus turn-around).
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.

1.12.3. 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 (parity mode only), B, C, or Subtractive, as given in Table 2-11.
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 2-21. For common-clock data phases on 16-bit buses, the target always drives its data phase action for two clocks for transactions that transfer data. (Common-clock data phases on 16-bit buses always come in DWORD-aligned pairs for transactions that transfer data. See Section 2.12.2.3.3.)
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# for 64- and 32-bit transfers. 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# for 64- and 32-bit transfers. All PCI-X targets (including the host bridge) are subject to the same target initial latency limits. See Section 2.9.1 for target initial latency for 16-bit transfers.
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) for 64- and 32-bit transfers. 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.) If the target signals Single Data Phase Disconnect, the transaction ends after a single data phase for 64- and 32-bit transfers. For Memory Write Block transactions in PCI-X Mode 2, the data phase still includes the appropriate number of data subphases, but only the data of the subphase that includes the starting address is transferred. See Section 2.9.1 for requirements on 16-bit buses.
7. The target is permitted to signal Target-Abort on any data phase regardless of its relationship to an ADB. See Section 2.12.2.3.3 for requirement on 16-bit buses.
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.

1.12.4. 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 4.2.1).
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.) 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.
4. In PCI-X Mode 1, 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 on 64- and 32-bit buses.)

In PCI-X Mode 2, exactly one device must own the bus at all times when RST# is deasserted and the bus is not in interface low-power state. Bus ownership changes in a precisely controlled manner with a bus turn-around alert described in Rule 10 below. If GNT# is asserted after the bus turn-around alert, 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#.

5. If only one REQ# is asserted, it is recommended that the arbiter keep GNT# asserted to that device.
6. If the arbiter deasserts GNT# to one device, in PCI-X Mode 1 it must wait until the next clock to assert GNT# to another device. In PCI-X Mode 2, it asserts the next GNT# on the same clock it deasserts the previous GNT#. (The first clock GNT# is asserted in Mode 2 is the bus-turn-around alert. See Rule 10 below.)
7. An initiator is permitted to start a new transaction (drive the AD bus, etc.) on any clock N in which one of the following is true:
   a. In PCI-X Mode 1, 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 4.2.1).
   b. In PCI-X Mode 2, GNT# was asserted and the bus was idle (FRAME# and IRDY# were both deasserted) on clock N-2 and GNT# was asserted on clock N-3 (could be the bus turn-around alert).

   An initiator is permitted to start a new transaction on clock N even if GNT# is deasserted on clock N-1.
8. All fast back-to-back transactions as defined in PCI 2.3 are not permitted in PCI-X mode.
9. 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.
10. The following rules apply only to PCI-X Mode 2:
    a. The present bus owner actively drives AD[31::00], C/BE[3::0]#, and ECC[6::0] to a stable logic level whenever the bus is idle, except for the bus turn-around clock defined below. (It drives 16 AD bits, two C/BE# bits, and four ECC bits, as shown in Table 2-22, on 16-bit buses.)
    b. To turn the bus around in PCI-X Mode 2, the arbiter deasserts GNT# to the present bus owner and asserts GNT# to all other devices simultaneously. This is the bus turn-around alert. On the clock after the turn-around alert, the arbiter deasserts GNT# to all devices except the new bus owner.
    c. If the bus is idle in the clock after the turn-around alert, the bus turns around on the second clock after the alert.

d. If the bus is not idle in the clock after the turn-around alert, the bus turns around at the end of the transaction. The second clock after the end of every transaction (the clock following the first clock in which both FRAME# and IRDY# are deasserted) is reserved for a bus turn-around, whether there is an actual change of bus ownership or not.

e. Multiple bus turn-around alerts are permitted during a single transaction. In this case, there is still only a single bus turn-around cycle, and it occurs in the same place at the end of the transaction. If the second alert occurs on clock N that is the first idle clock at the end of the transaction, the bus turn-around for the first change of ownership corresponds to clock N+1 of the second alert. In this case, the bus turns around twice in a row (clock N+1 and N+2).

f. The previous bus owner stops driving the bus during the bus turn-around, and the new bus owner starts driving the bus the clock after the bus turn-around. All devices disable their receivers for Category 1 signals during the bus turn-around.

g. The arbiter places all devices in the interface low-power state by signaling a bus turn-around alert and then signaling another bus turn-around alert two clocks later. After the second alert, the arbiter deasserts all GNT# signals. To awaken the devices from the interface low-power state, the arbiter signals a bus turn-around alert and leaves GNT# asserted to the new bus owner.

1.12.5. 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 on a 64- or 32-bit bus.
2. In addition to the information required in conventional PCI for a Type 0 configuration transaction, in PCI-X mode 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.
5. In PCI-X Mode 2, AD[27::24] of Type 0 Configuration Space addresses are used as an additional four bits of configuration register addresses, enabling the addressing of 4096 bytes for each function of each device.

1.12.6. Parity and ECC Error Protection Rules

The following rules apply to parity and ECC generation and checking:

1. By default after RST#, PCI-X Mode 1 uses the same parity bits as conventional PCI to detect errors on the bus. (See Section 6.2 for exceptions for some PCI-X initialization patterns.) Optional configuration bits change the error protection in PCI-X Mode 1 to ECC. PCI-X Mode 2 uses ECC exclusively.
2. In ECC mode:
   a. ECC is used to detect and automatically correct (if enabled) any single bit error and to detect any double bit error in any address, attribute, or data phase.
   b. A seven-bit ECC protects the lower bus during the address and attribute phases, and during all 32-bit data phases or subphases.
   c. The same seven-bit ECC plus an additional check bit protect pairs of 16-bit phases or subphases.
   d. An eight-bit ECC protects a QWORD of data during all 64-bit data phases or subphases.
   e. A 32-bit target of a burst push transaction from a 64-bit initiator must insert a minimum of two target initial wait states, to allow the initiator to convert from eight-bit to seven-bit ECC.
   f. The ECC check bits share PAR, PAR64, REQ64#, and ACK64# pins defined in parity mode plus additional pins assigned only in ECC mode.
   g. ECC timing requirements are identical to parity timing requirements in all phases except the source-synchronous data phases. In the common-clock cases, the value of the ECC check bits appears on the bus delayed one clock from the information they protect. ECC timing in source-synchronous data subphases is coincident with the information it protects.
   h. Unless correction is disabled, correctable ECC errors are corrected by the device that detects them and the transaction proceeds as if no error occurred. The error information is latched for diagnostic purposes.
   i. Uncorrectable ECC errors in ECC mode are treated the same as parity errors in parity mode.
3. If a device receiving data (i.e., the target of a write Split Completion, or device ID message and the initiator of a read) detects an uncorrectable data error, it must do one of the following:
   a. For parity mode, assert PERR# (if enabled) on the second clock after PAR64 and PAR are driven (one clock later than conventional PCI).
   b. For ECC mode common-clock transfers, assert PERR# (if enabled) on the second clock after ECC[7] (for 64-bit transfers) and ECC[6::0] are driven (one clock later than conventional PCI).
   c. For source-synchronous transfers, assert PERR# (if enabled) on clock N+5 for data and ECC driven between clocks N and N+1 (two clocks later than conventional PCI).
4. During parity-protected 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. On ECC-protected transactions, the target drives ECC[7] (if responding as a 64-bit device) and ECC[6::0] on clock N+1 for the read data it drives on clock N (the byte enables are not included in the ECC). During write transactions in parity mode, 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. During Memory Write transactions in ECC mode, the initiator drives ECC[7] (if initiating as a 64-bit device) and ECC[6::0] on clock N+1 for the write data and the byte enables it drives on clock N. During source-synchronous transactions, the initiator drives ECC[7] (if initiating as a 64-bit device) and ECC[6::0] coincident with the data it protects. On 16-bit buses, the ECC check bits for each DWORD are multiplexed on two data phases or subphases (see Section 5.1.3).

5. All PCI-X devices are required to service uncorrectable data error conditions for their transactions. See Section 5.2.1.
6. If a device detects an uncorrectable error on an attribute phase, the device asserts SERR# (if enabled), independent of whether the device decodes its address during the address phase.
7. For Split Transactions, the requester sets the Master Data Parity Error bit in the Status register for uncorrectable data errors on either the Split Request or the Split Completion.
8. If uncorrectable data error recovery is disabled, the device asserts SERR# when an uncorrectable data error occurs (see Section 5.2.1).
9. Other requirements for asserting SERR# and setting status bits for address-phase and data-phase errors are the same as for conventional PCI.

1.12.7. Bus Width Rules

The following rules apply to the width of the transaction:

1. As in conventional PCI, PCI-X devices are required to implement a 32-bit interface and optionally implement a 64-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 two address phases on 64- and 32-bit buses.
6. The attribute phase is always a single clock long for both 64-bit and 32-bit buses.
7. In parity mode, only burst transactions (Split Completions, device ID messages, and 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.

In ECC mode, the upper buses (AD[63::32] and C/BE[7::4]#) are only used in data phases and are not used during the address and attribute phases. The width of each transaction is determined with a handshake protocol similar to the one used in parity mode. See Section 2.12 for details.

8. In PCI-X Mode 2, a 32-bit target of a burst push transaction from a 64-bit initiator must insert a minimum of two wait states.
9. All Mode 2 devices are required to support a 16-bit interface.
    a. The 16-bit interface is intended for embedded applications, so no width negotiation protocol or add-in card connector pin-out is specified, and bridge hierarchies are not allowed.
    b. The 16-bit bus always operates in Mode 2 and all transactions are 16-bits wide.
    c. In 16-bit transfers, there are twice as many address and attribute phases as 64-and 32-bit transfers.
    d. As in 64- and 32-bit transfers, the minimum data granularity (ignoring byte enables) is one DWORD. So 16-bit common-clock data phase and source-synchronous subphases always come in pairs that are aligned to a DWORD boundary.
    e. 16-bit transfers exclusively use seven-bit ECC plus an additional special check bit.

1.12.8. 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 on a 64- or 32-bit bus (two data phases on a 16-bit bus). 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.

1.13. PCI-X Transaction Flow

The following two figures illustrate how transactions flow through a PCI-X system. Figure 1-9 illustrates transaction flow from a requester directly to a completer (no intervening bridge). Figure 1-10 illustrates transaction flow across a bridge.

As shown in Figure 1-9, 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 1-10 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.

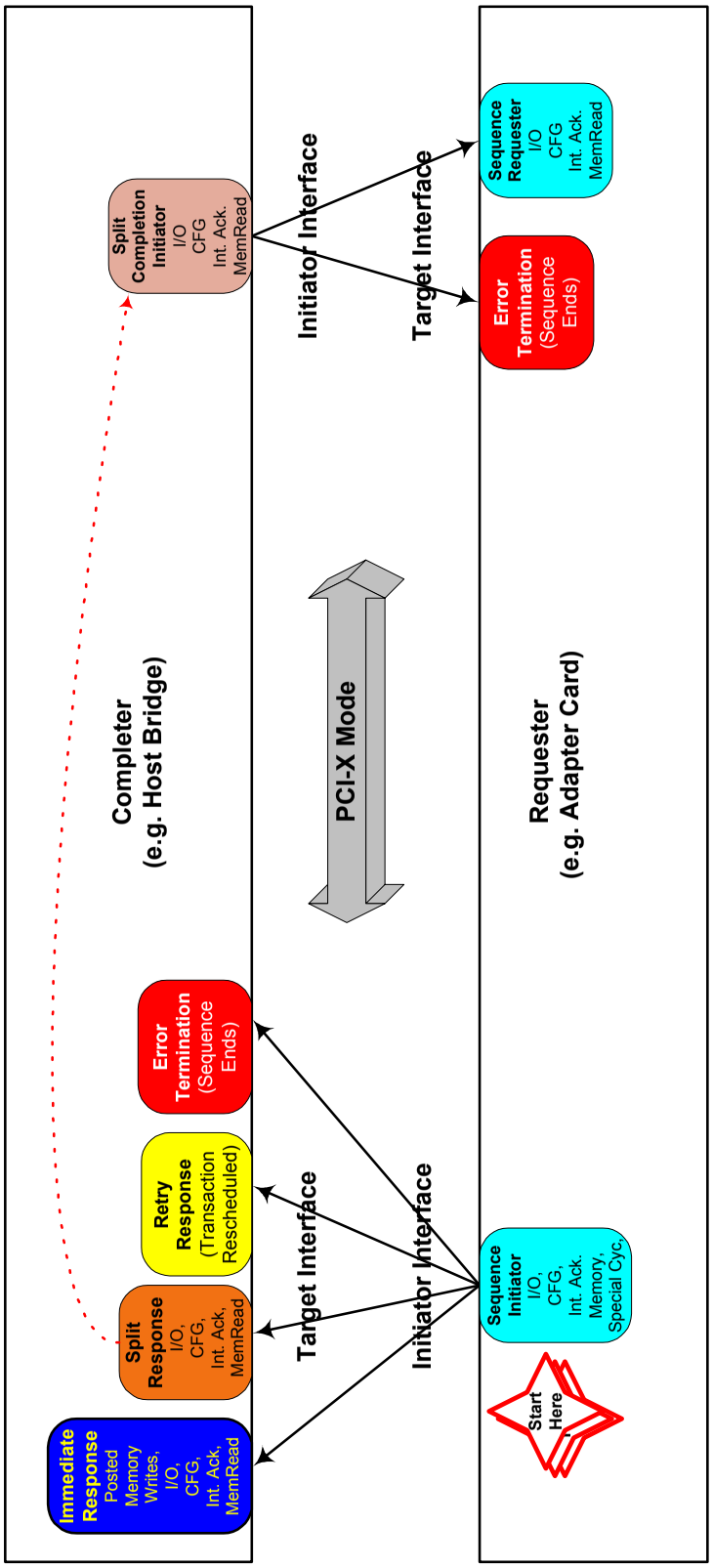

Figure 1-9: Transaction Flow without Crossing a Bridge

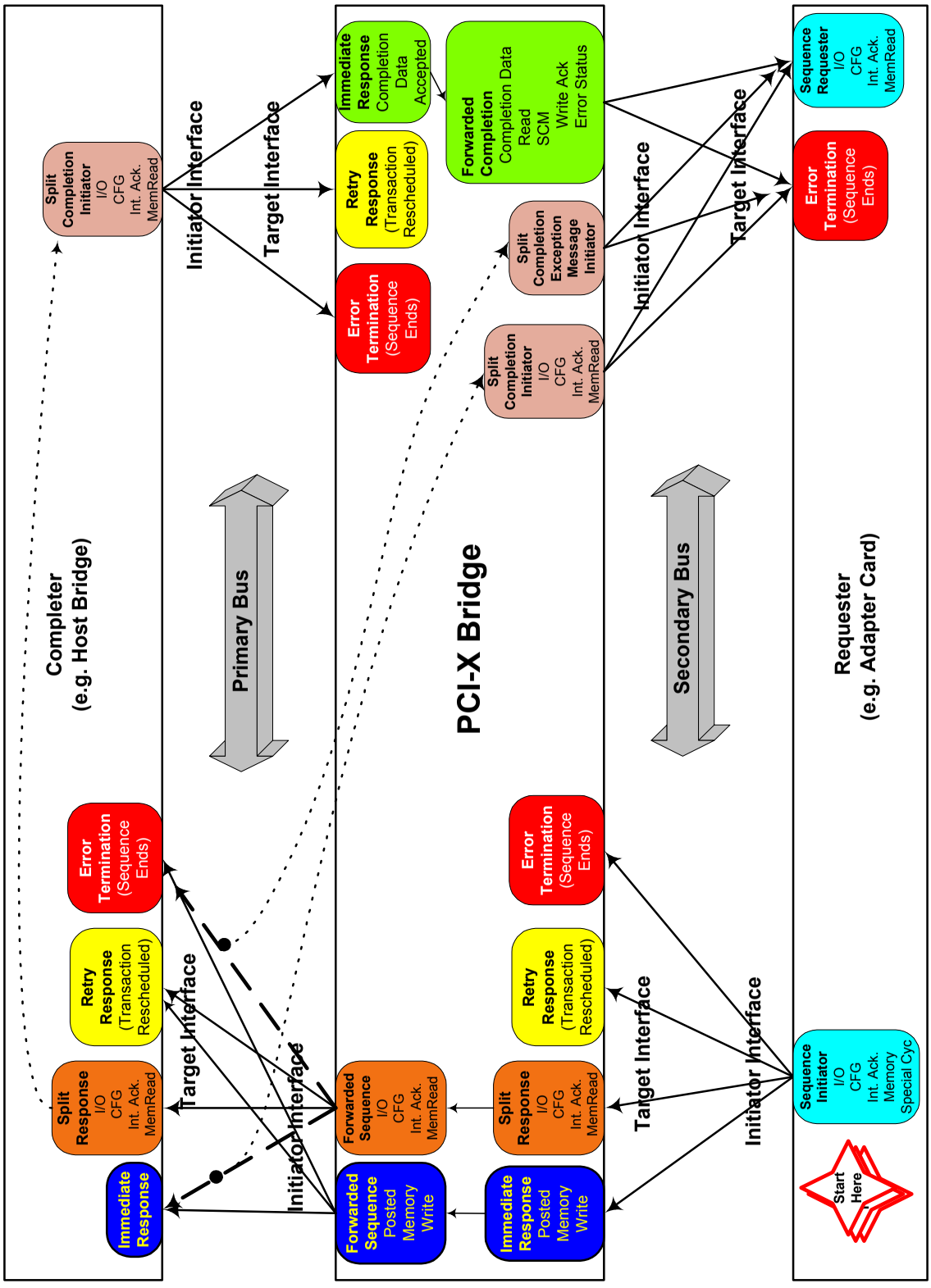

Figure 1-10: Transaction Flow Across a Bridge

2

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 memory 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 memory 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. Transactions using the Device ID Message command have the same requirements as burst memory writes, except they have no address to adjust (see Section 2.16.1).

If the Sequence is a burst memory write or device ID message 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 memory write or device ID message 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 memory write or device ID message, 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 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 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 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 2.3 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 with a message that the Sequence is complete or 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. The bridge keeps all memory write and device ID messages in the order that they arrived.

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 seven bits are zeros. After a burst data transfer starts and the target signals that it will accept more than a single data phase (two data phases for 16-bit common-clock bursts), 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 and regardless of whether each data phase includes multiple data subphases or not. Table 2-1 shows the number of data phases a burst transaction would require to go from one ADB to the next for each combination of transfer width and number of data subphases per data phase.

Table 2-1: Number of Data Phases Between Two ADBs

		Number of Data Phases Between Two ADBs		
Mode	**Number of Data Subphases**	**64-Bit Transfer**	**32-Bit Transfer**	**16-Bit Transfer**
PCI-X 66, PCI-X 133	none	16	32	na
PCI-X 266	2	8	16	32
PCI-X 533	4	4	8	16

Burst transactions other than device ID messages (see Section 2.16.1) are permitted to start on any byte address. Both the initiator and the target are permitted to disconnect a burst transaction on any ADB. (See Section 2.10.5 for limitations on disconnecting Split Completion transactions.) A target is permitted also to disconnect a burst other than a device ID message after a single data phase (two data phases for 16-bit common-clock bursts). (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:

- The transaction length is one ADQ.
- The completer does not signal Single Data Phase Disconnect.
- 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, transaction data bytes 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 and the number of data subphases in each data phase). Table 2-2 shows how bytes are assigned to byte lanes. (See Table 2-22 for the pin assignment on a 16-bit interface.)

PCI-X Mode 2 uses the same byte lane assignments as PCI-X Mode 1. Furthermore, for source-synchronous transactions, data phase boundaries are naturally aligned. That is, bytes appear in data subphases according to their individual byte address and the width of the transfer (independent of the starting address). (See Section 2.10.2 for the requirement for the address to be set to zero for Split Completions resulting from DWORD Split Requests.) Table 2-3 and Table 2-4 show the mapping of the naturally aligned DWORDs to the physical bus (AD[63::0]) and the data subphases. In PCI-X 266 mode, a 64-bit transaction transfers up to 16 bytes per data phase, a 32-bit transaction transfers up to eight bytes per data phase, and a 16-bit transaction transfers up to four bytes per data phase. In PCI-X 533 mode, a 64-bit transaction transfers up to 32-bytes per data phase, a 32-bit transaction

transfers up to 16 bytes per data phase, and a 16-bit transaction transfers up to eight bytes per data phase.

Table 2-2: Byte Lane Assignments

	Data Byte Lane		
Byte Address AD[2::0]	**64-Bit Transfer**	**32-Bit Transfer**	**16-Bit Transfer**
000b	**AD[7::0]**	**AD[7::0]**	**AD[23::16]**
001b	**AD[15::08]**	**AD[15::08]**	**AD[31::24]**
010b	**AD[23::16]**	**AD[23::16]**	**AD[23::16]**
011b	**AD[31::24]**	**AD[31::24]**	**AD[31::24]**
100b	**AD[39::32]**	**AD[7::0]**	**AD[23::16]**
101b	**AD[47::40]**	**AD[15::08]**	**AD[31::24]**
110b	**AD[55::48]**	**AD[23::16]**	**AD[23::16]**
111b	**AD[63::56]**	**AD[31::24]**	**AD[31::24]**

Table 2-3: Address Assignment to Data Lanes and Subphases, PCI-X 266

	64-Bit Transfer		32-Bit Transfer		16-Bit Transfer	
AD[3::1]	**Data Lanes**	**Data Subphase**	**Data Lanes**	**Data Subphase**	**Data Lanes**	**Data Subphase**
000b	**AD[15::00]**	0	**AD[15::00]**	0	**AD[31::16]**	0
001b	**AD[31::16]**	0	**AD[31::16]**	0	**AD[31::16]**	1
010b	**AD[47::32]**	0	**AD[15::00]**	1	**AD[31::16]**	0
011b	**AD[63::48]**	0	**AD[31::16]**	1	**AD[31::16]**	1
100b	**AD[15::00]**	1	**AD[15::00]**	0	**AD[31::16]**	0
101b	**AD[31::16]**	1	**AD[31::16]**	0	**AD[31::16]**	1
110b	**AD[47::32]**	1	**AD[15::00]**	1	**AD[31::16]**	0
111b	**AD[63::48]**	1	**AD[31::16]**	1	**AD[31::16]**	1

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. (See Section 2.16.1 for special addressing requirements of Device ID Message transactions.) 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 (other than inclusion in the parity or ECC calculations). For burst transactions, the starting address and ending address are not required to have any alignment to the bus width or data phase boundaries.

Table 2-4: Address Assignment to Data Lanes and Subphases, PCI-X 533

	64-Bit Transfer		32-Bit Transfer		16-Bit Transfer	
AD[4::1]	Data Lanes	Data Subphase	Data Lanes	Data Subphase	Data Lanes	Data Subphase
0000b	**AD[15::00]**	0	**AD[15::00]**	0	**AD[31::16]**	0
0001b	**AD[31::16]**	0	**AD[31::16]**	0	**AD[31::16]**	1
0010b	**AD[47::32]**	0	**AD[15::00]**	1	**AD[31::16]**	2
0011b	**AD[63::48]**	0	**AD[31::16]**	1	**AD[31::16]**	3
0100b	**AD[15::00]**	1	**AD[15::00]**	2	**AD[31::16]**	0
0101b	**AD[31::16]**	1	**AD[31::16]**	2	**AD[31::16]**	1
0110b	**AD[47::32]**	1	**AD[15::00]**	3	**AD[31::16]**	2
0111b	**AD[63::48]**	1	**AD[31::16]**	3	**AD[31::16]**	3
1000b	**AD[15::00]**	2	**AD[15::00]**	0	**AD[31::16]**	0
1001b	**AD[31::16]**	2	**AD[31::16]**	0	**AD[31::16]**	1
1010b	**AD[47::32]**	2	**AD[15::00]**	1	**AD[31::16]**	2
1011b	**AD[63::48]**	2	**AD[31::16]**	1	**AD[31::16]**	3
1100b	**AD[15::00]**	3	**AD[15::00]**	2	**AD[31::16]**	0
1101b	**AD[31::16]**	3	**AD[31::16]**	2	**AD[31::16]**	1
1110b	**AD[47::32]**	3	**AD[15::00]**	3	**AD[31::16]**	2
1111b	**AD[63::48]**	3	**AD[31::16]**	3	**AD[31::16]**	3

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 2-5. 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.

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. No other transactions include byte enables. In PCI-X Mode 1, the C/BE# bus is reserved and driven high by the initiator throughout all data phases of all other transactions.

Table 2-5: AD[1::0] and Byte Enable Encodings for I/O and DWORD Memory Transactions

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

Note:
"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.

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 2-6 and Table 2-7. See Section 2.12.1.3 for exceptions and additional requirements when a 64-bit initiator addresses a 32-bit target and AD[2] is 1.

Table 2-6: 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)
00b	xxxxb
01b	xxx1b
10b	xx11b
11b	x111b

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

Table 2-7: 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 2.12.1.3.

2.4. PCI-X Command Encoding

Table 2-8 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 except for logging of address-phase errors) any transactions using a reserved command.

Table 2-8: 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	Device ID Message	Burst	none	
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 are 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 The transaction does not use byte enables. All bytes between the starting and ending address inclusive are affected. In common-clock data phases, the C/BE# bus is reserved and driven high by the initiator. In source-synchronous data phases, the C/BE# bus is used for data strobes.

Notes:

1. For all commands other than Dual Address Cycle, the transaction command appears on C/BE[3::0]# during the (single) address phase on 64- and 32-bit buses. 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 in parity mode, C/BE[7::4]# contain the transaction command in both address phases. In ECC mode, C/BE[7::4]# are used only in

data phases. In 16-bit transaction, commands are multiplexed on two C/BE# bits in two or four address phases, see Table 2-22.

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. See Section 2.12.2.2 for differences in 16-bit buses. 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 for 64- and 32-bit buses and two clocks for 16-bit buses, regardless of the width of the address (single or dual address cycle). In parity mode, the upper buses (AD[63::32] and C/BE[7::4]#) of 64-bit devices are reserved and driven high during the attribute phase. In ECC mode, the upper buses of 64-bit devices (AD[63::32], C/BE[7::4]#, and ECC[7]) are optionally driven by the initiator, and the receivers in the target are disabled during the attribute phase.

There are four 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, Completer Attributes for Split Completions in Section 2.10.4, and DIM Attributes for device ID messages in Section 2.16.2.

Figure 2-1 and Figure 2-2 show the bit assignments for the Requester Attributes for burst and DWORD transactions, respectively. Table 2-9 describes the bit definitions.

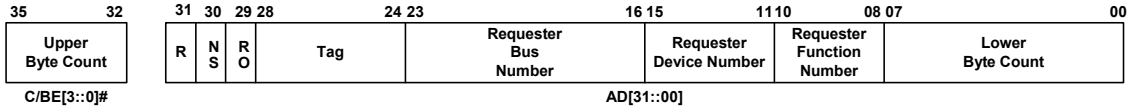

Figure 2-1: Burst Transaction Requester Attribute Bit Assignments

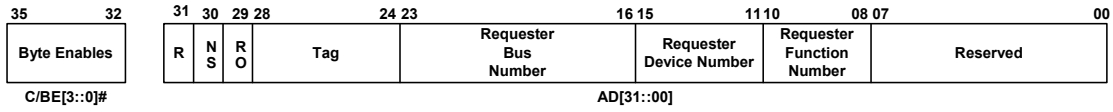

Figure 2-2: DWORD Transaction Requester Attribute Bit Assignments

Table 2-9: 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 or ECC 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 2.3). The requester must not set this bit if the transaction is a Message Signaled Interrupt, I/O, Special Cycle, or Device ID Message 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 Appendix B 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

Attribute	Function
	set. 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 2.3), and for device ID messages. 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, Section 2.10.4 for Split Completions, and Section 2.16.2 for device ID messages.) 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 requester. The requester 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 one 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
- Device ID Message

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-SIG. All requirements stated throughout this document for Memory Read Block also apply to Alias to Memory Read Block and all requirements stated throughout this document for Memory Write Block also apply to Alias to Memory Write Block.

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 on 64-bit buses.

Burst memory transactions use the full address bus (including AD[2::0]) to specify the starting byte address of the transaction. (In ECC mode, AD[63::32] are not used during the address phase. See Section 2.12.1.1 for the case in which the address is greater than 4 GB.)

There are no restrictions on the starting address. The transaction is permitted to begin on any byte address. (Byte lanes and data phase boundaries are always naturally aligned to the address. See Section 2.3.) Split Completions and device ID messages have special addressing requirements specified in Sections 2.10.3 and 2.16.1, respectively.

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 2.12.1.1). 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 in PCI-X Mode 1, and Memory Write, Memory Write Block, Split Completion, and Device ID Message in Mode 2. See Section 2.6.1 for the behavior of the C/BE# bus for a Memory Write transaction. See Section 2.1.3.5.1, "Source-Synchronous Data Strobes," in PCI-X EM 2.0 for the behavior of the C/BE# bus for carrying data strobes during the data subphases of Memory Write Block, Split Completion, and Device ID Message transactions.

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 Push Transactions

Burst push transactions use the Memory Write, Memory Write Block, Alias to Memory Write Block, Split Completion, or Device ID Message commands. 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 memory 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 permitted to respond to a Device ID Message transaction with any of the following:

- Target-Abort
- Wait State
- Data Transfer
- Retry
- Disconnect at Next ADB (The same restrictions apply to bridges as for burst memory write transactions. See Section 8.4.6.)

The target is not permitted to respond to burst push transactions with Split Response. Burst push transactions 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.

All bytes between the starting and ending address inclusive are included in these transactions. In PCI-X Mode 1, the C/BE# bus is reserved and driven high after the attribute phase of all burst push transactions except Memory Write, which is described below. See Section 2.12.1.3 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 2.12.1.3, 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 or ECC, 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 2.3 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 are included 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.

IMPLEMENTATION NOTE

Don't-Care Clock on Mode 1 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 in Mode 1 as a turn-around clock by multi-package host bridges that source the address from one package and the data from another. (In Mode 2, the data must be driven by the initiator after the attribute phase, so multi-package host bridges are not possible.)

2.6.1.1. Common-Clock Burst Push Transactions

All data phases in PCI-X Mode 1 and data phases for Memory Write transactions in PCI-X Mode 2 are common-clock data phases. For common-clock data phases for burst push transactions, the initiator must drive data on the AD bus two clocks after the attribute phase. (The initiator is not permitted to insert wait states.) If the burst push transaction has 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 2.9.2 for a complete discussion of the effects of different DEVSEL# decode times and target initial wait states. See Section 2.12.1.3 for requirements for a 64-bit initiator writing to 32-bit targets.

The next two figures show burst push transactions using the Memory Write command. Memory Write Block, Split Completion, and Device ID Message transactions in PCI-X Mode 1 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 (in parity mode) and target initial latency and eight data phases. Minimum DEVSEL# decode time in ECC mode would be one clock later.

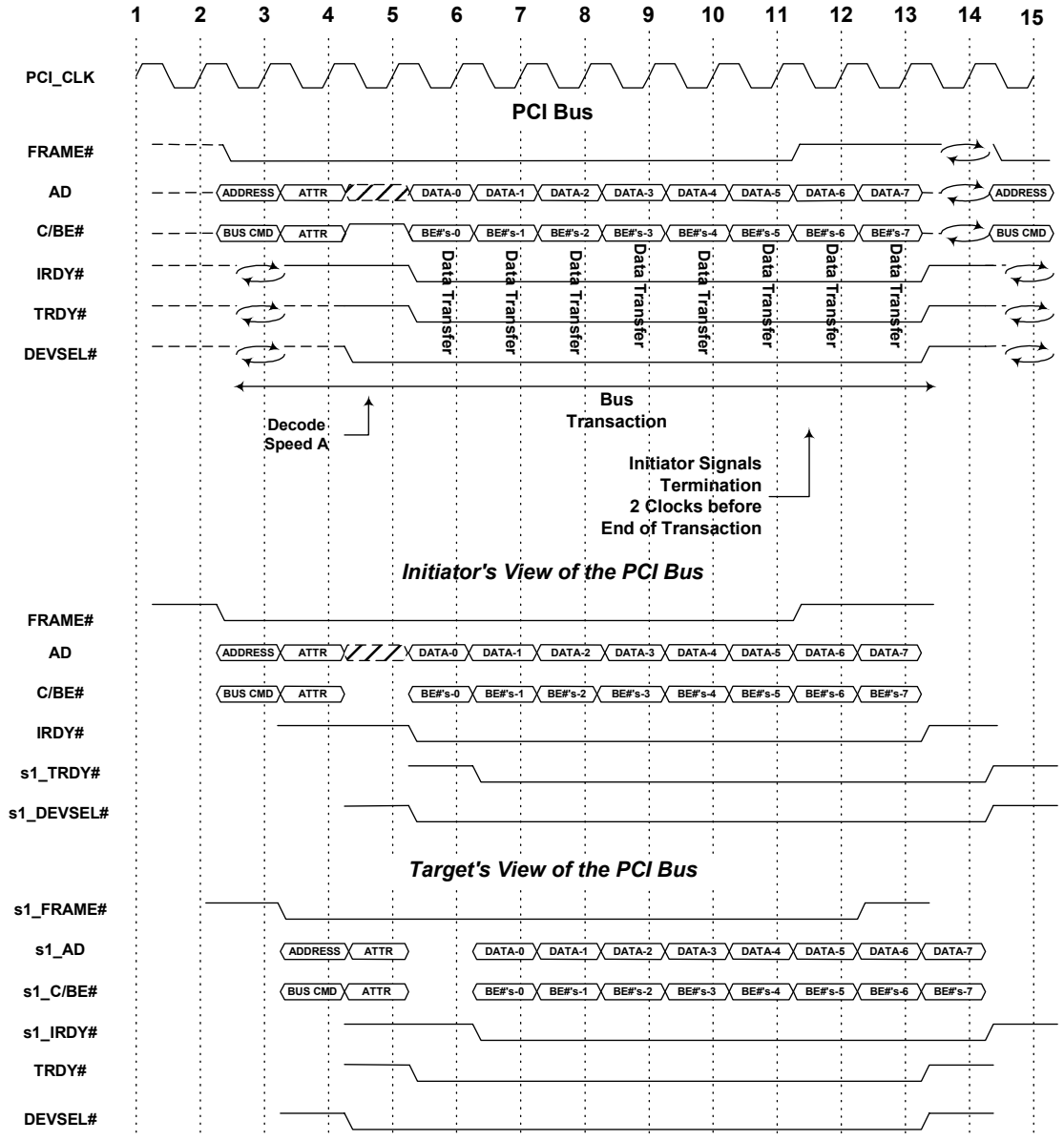

Figure 2-3: Common-Clock Burst Memory Write Transaction with No Target Initial Wait States, Mode 1

Figure 2-4 shows a similar transaction with minimum DEVSEL# decode timing (in parity mode) 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 with the first data value one clock after the target asserts DEVSEL# (clock 5), advances to the second data value two clocks after the target asserts DEVSEL#, and repeats until the target asserts TRDY# (clock 8). (Note that this data toggling is why target initial wait states must occur in pairs for burst push transactions.) See Section 2.9.2 for a complete discussion of the effects of different DEVSEL# decode times and target initial wait states.

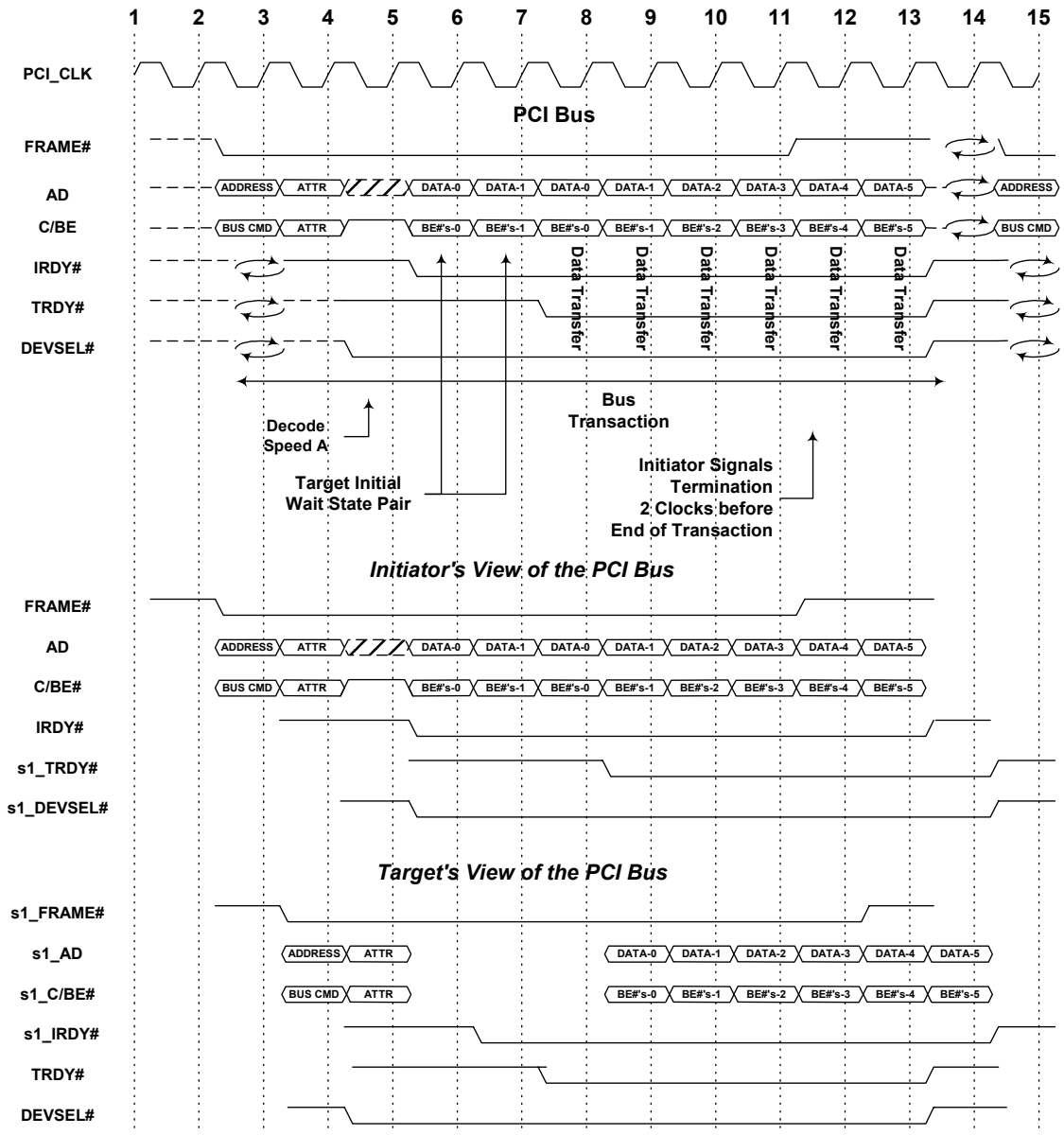

Figure 2-4: Common-Clock Burst Memory Write Transaction with Two Target Initial Wait States, Mode 1

Figure 2-5 and Figure 2-6 show the same cases in PCI-X Mode 2 as the previous two figures in PCI-X Mode 1. Note that the minimum DEVSEL# decode time is one clock longer in Mode 2 than the previous examples, to allow the target to check ECC on the address. Also, note that there is a minimum of two idle clocks between all transactions in PCI-X Mode 2.

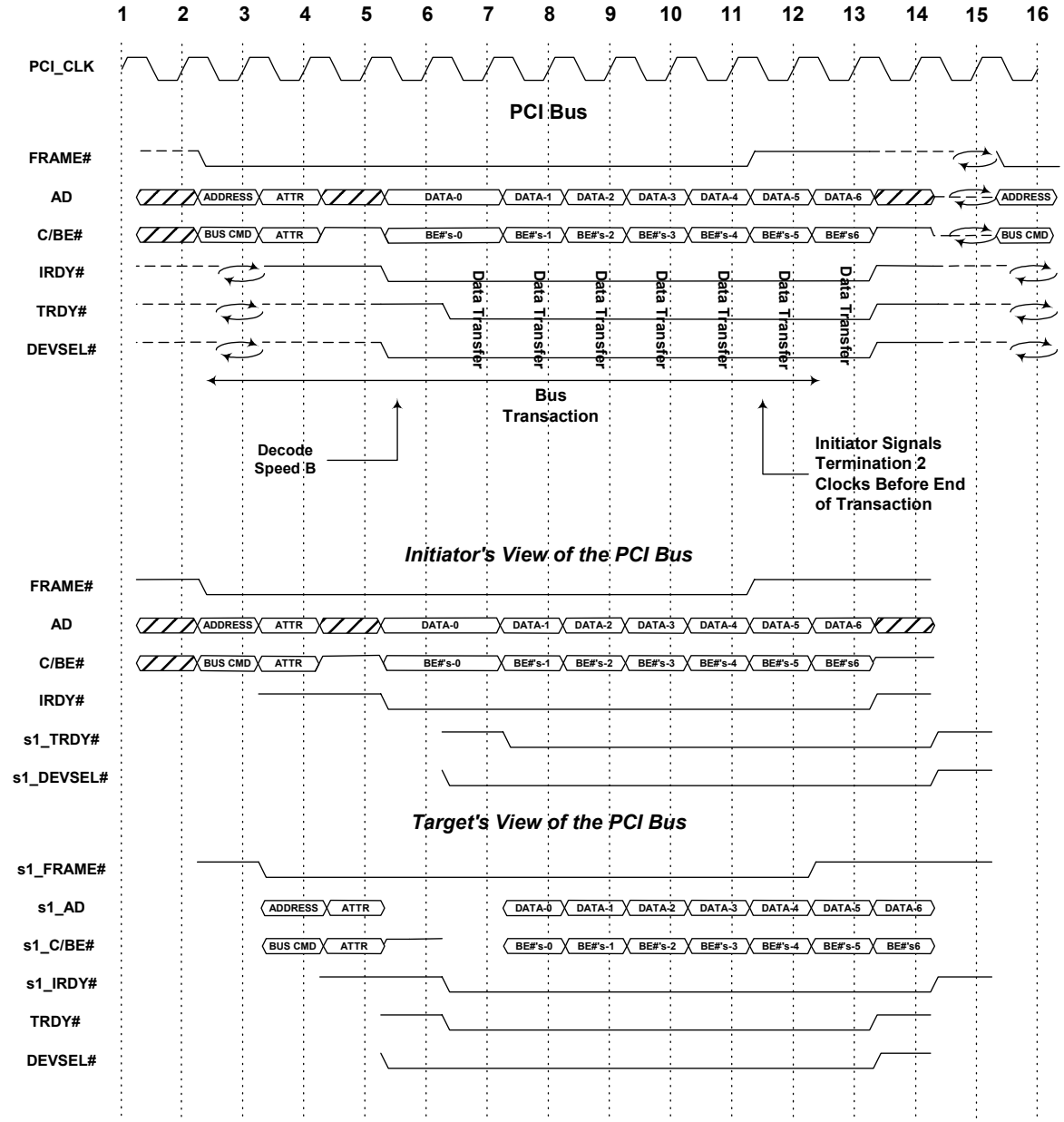

Figure 2-5: Common-Clock Burst Memory Write Transaction with No Target Initial Wait States, Mode 2

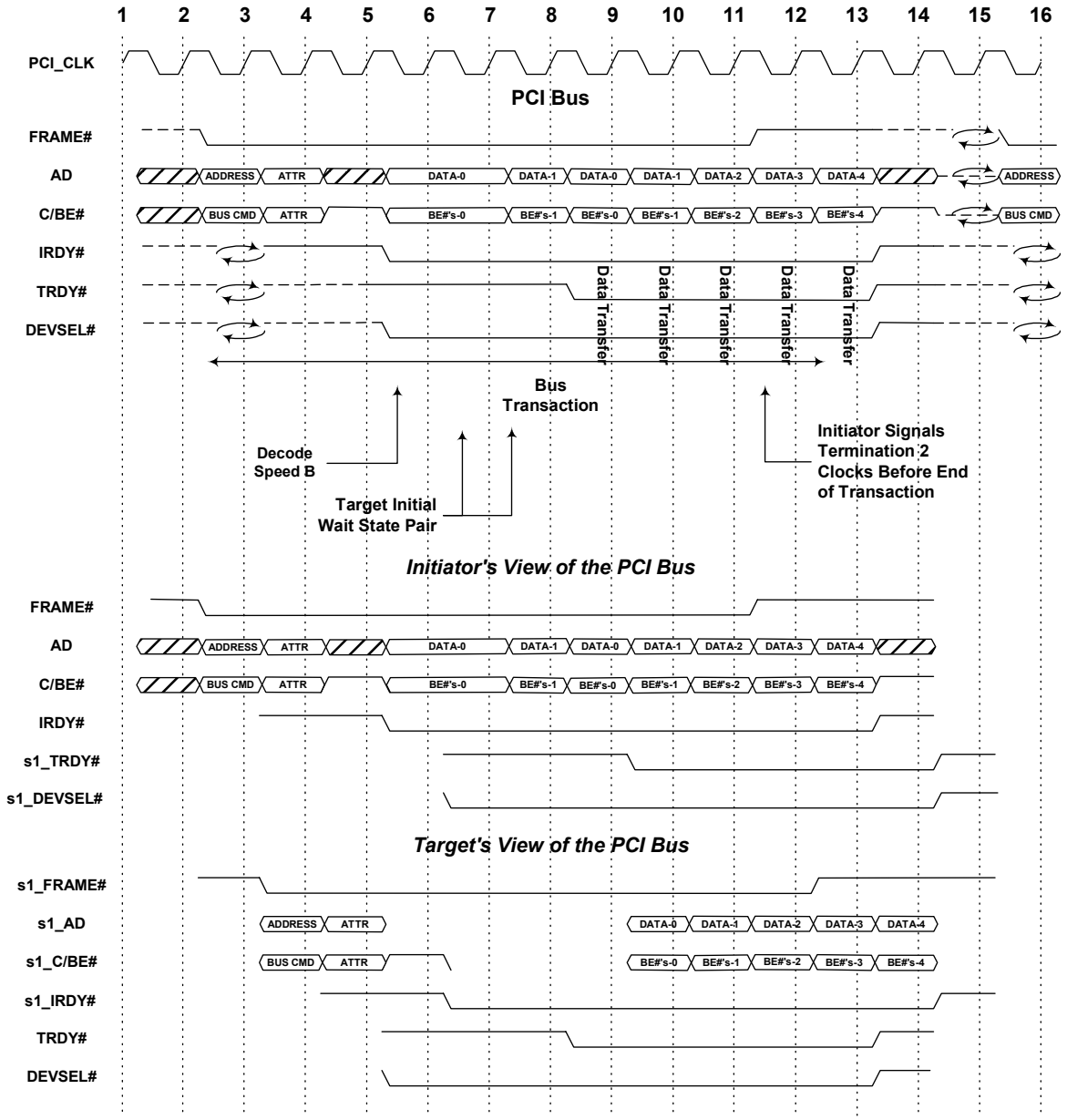

Figure 2-6: Common-Clock Burst Memory Write Transaction with Two Target Initial Wait States, Mode 2

2.6.1.2. Source-Synchronous Burst Push Transactions

In PCI-X Mode 2 all burst push transactions other than Memory Write are source-synchronous transactions. Source-synchronous transactions use common-clock sampling for the control signal during all phases and for the AD and ECC buses during address and attribute phases. They use source-synchronous sampling for the AD and ECC buses during data phases, as described in Section 2.1.3.5.2, "1.5V Environment Source-Synchronous Timing Parameters," in PCI-X EM 2.0. For source-synchronous data phases, the initiator

must drive data on the AD bus and ECC check bits on the ECC bus (see Section 5.1.2.4) between the first and second clocks after the attribute phase. (The initiator is not permitted to insert wait states.) The initiator must also drive the data strobes as described in Section 2.1.3.5.1, "Source-Synchronous Data Strobes," in PCI-X EM 2.0.

The initiator drives the data bus beginning at the first data phase boundary that is less than or equal to the starting address of the transaction and advances through all data subphases of each data phase. If the burst push transaction has more than one data phase, the initiator advances to the second data phase value between the first and second clocks after the target asserts DEVSEL# (the same as in Mode 1 except including all data subphases), in anticipation of the target asserting TRDY#. If the target also inserts wait states, the initiator must toggle between its first and second data phase values (including all data subphases) until the target asserts TRDY# (or terminates the transaction). See Section 2.9.2 for a complete discussion of the effects of different DEVSEL# decode times and target initial wait states. See Section 2.12.1.3 for requirements for a 64-bit initiator writing to 32-bit targets.

The next two figures show source-synchronous burst push transactions with four subphases per data phase (PCI-X 533). The top portions of the diagrams show 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.

Figure 2-7 shows the minimum DEVSEL# decode time, target initial latency, and seven data phases. On clock 5, the initiator presents the data for a full data phase (four times the width of the bus) to its interface state machines that multiplex it onto the AD bus between clocks 5 and 6, one data subphase at a time. Since DEVSEL# is not asserted in clock 5, the initiator repeats the first data phase value between clocks 6 and 7. Since TRDY# is asserted on clock 7, data capture and demultiplexing logic in the target captures the subphases and resynchronize it to CLK internal to the target on clock 9, making it ready for use within the target at clock 10.

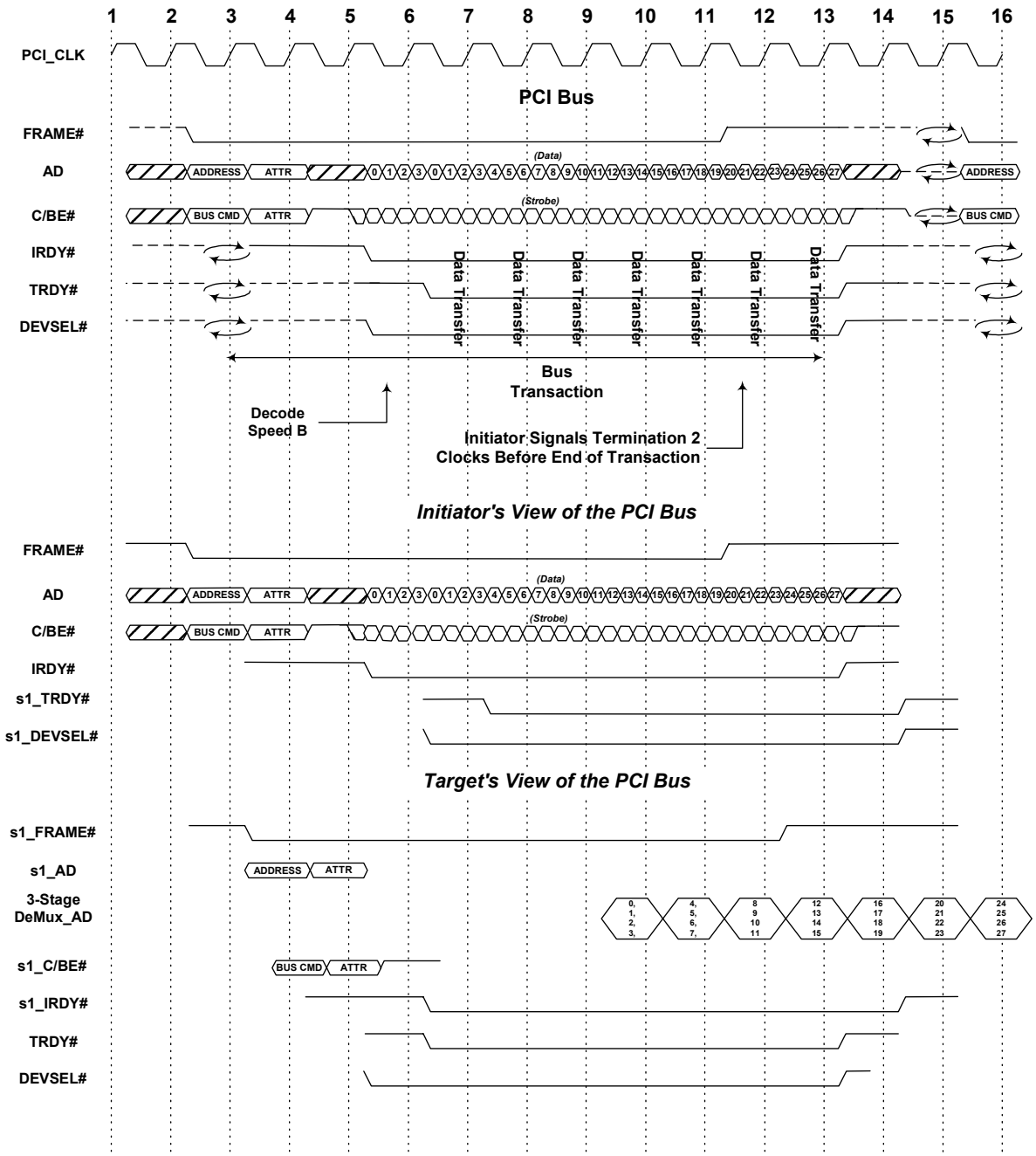

Figure 2-7: Source-Synchronous Burst Push Transaction with No Target Initial Wait States

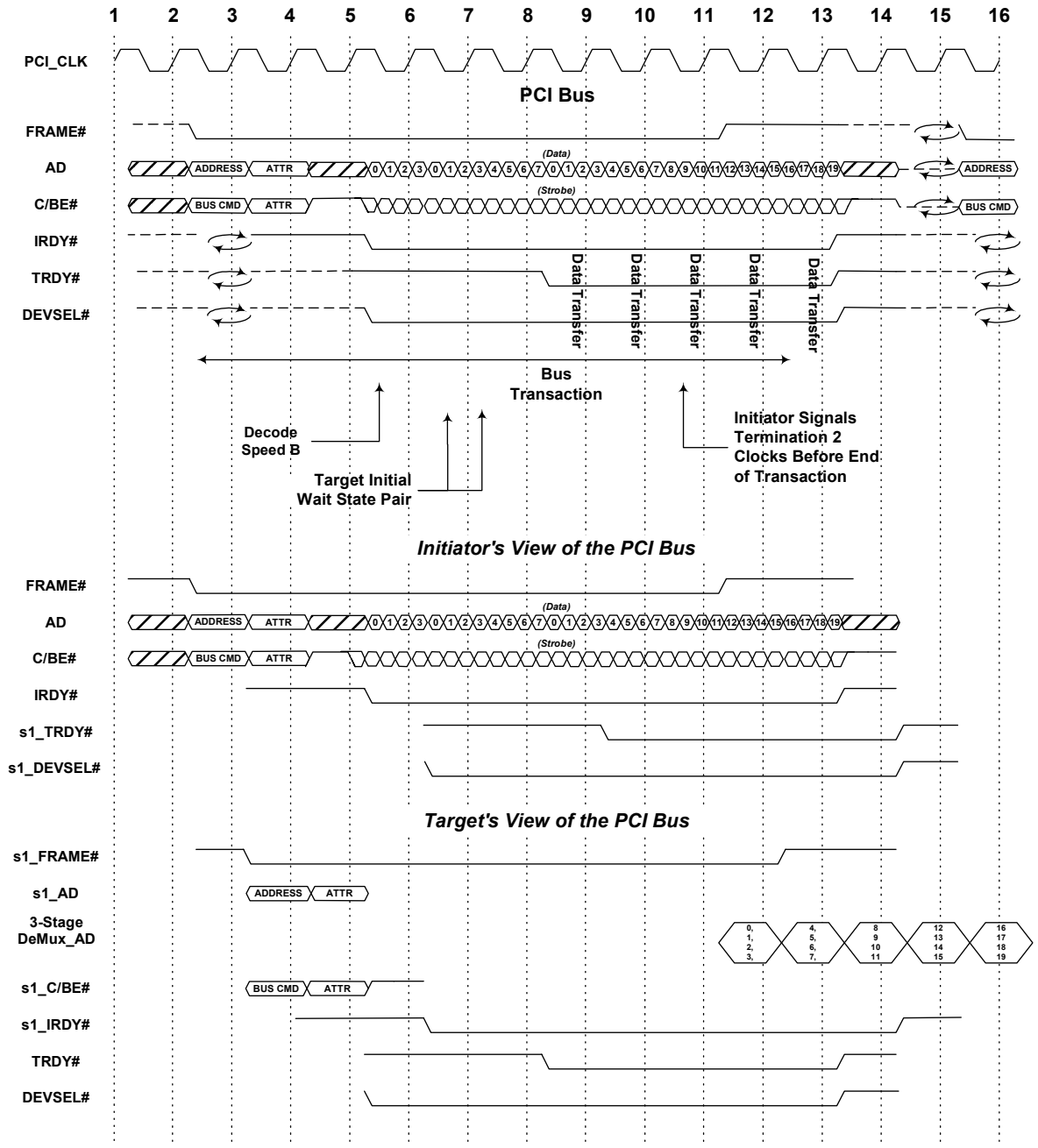

Figure 2-8: Source-Synchronous Burst Push Transaction with Two Target Initial Wait States

Figure 2-8 shows a similar transaction with minimum DEVSEL# decode timing but with only five data phases and a target initial latency of six clocks (two wait states). Notice that the initiator drives the first phase values between clocks 5 and 6 and then repeats it between clocks 6 and 7 because DEVSEL# was not asserted until clock 6. The initiator advances to the second data phase values between clocks 7 and 8, and then repeats the first two data phase values between clocks 8 and 9 and 9 and 10 because the target inserted wait states. (Note that this data toggling is the same as for common-clock transactions, except that

source-synchronous toggling includes all the subphases.) In Figure 2-8 the starting address is aligned to the data phase boundary. If the starting address is not aligned to the data phase boundary, the pattern driven on the AD bus prior to the starting address is not specified except that it must be included in the ECC calculation. When the pattern repeats, there is no requirement for the data pattern before the starting address to match earlier patterns, but if the data pattern changes, ECC must be recalculated.

See Section 2.9.2 for a complete discussion of the effects of different DEVSEL# decode times and 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 protocol for burst read transactions in Mode 1 and Mode 2 is identical (all common-clock) except that there are a minimum of two idle clocks after all transactions in Mode 2 (sometimes there is only one in Mode 1), and ECC is required in Mode 2 and optional in Mode 1.

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 2-9 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.

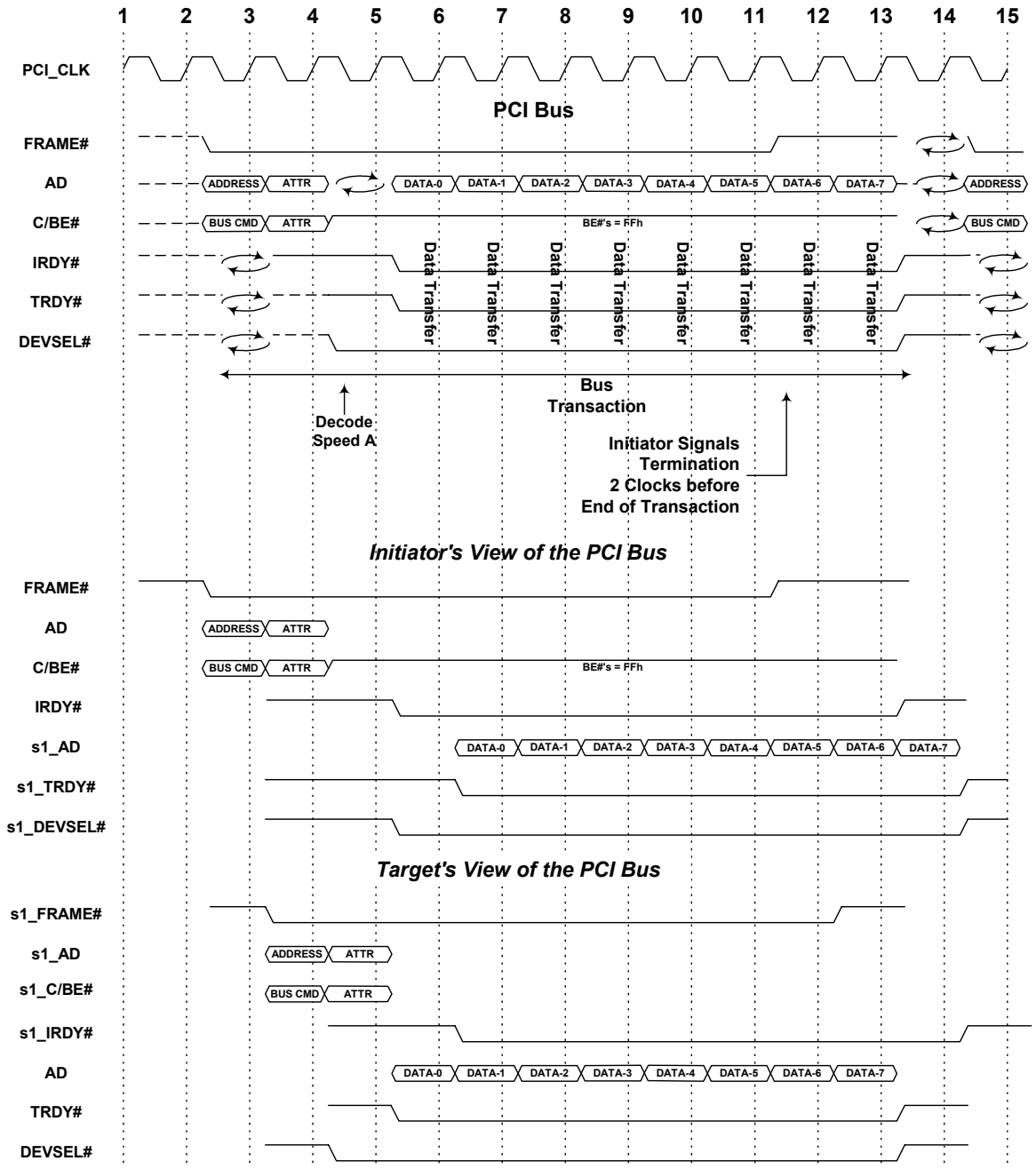

Figure 2-9: Burst Memory Read Transaction with No Target Initial Wait States, Mode 1

Figure 2-10 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).

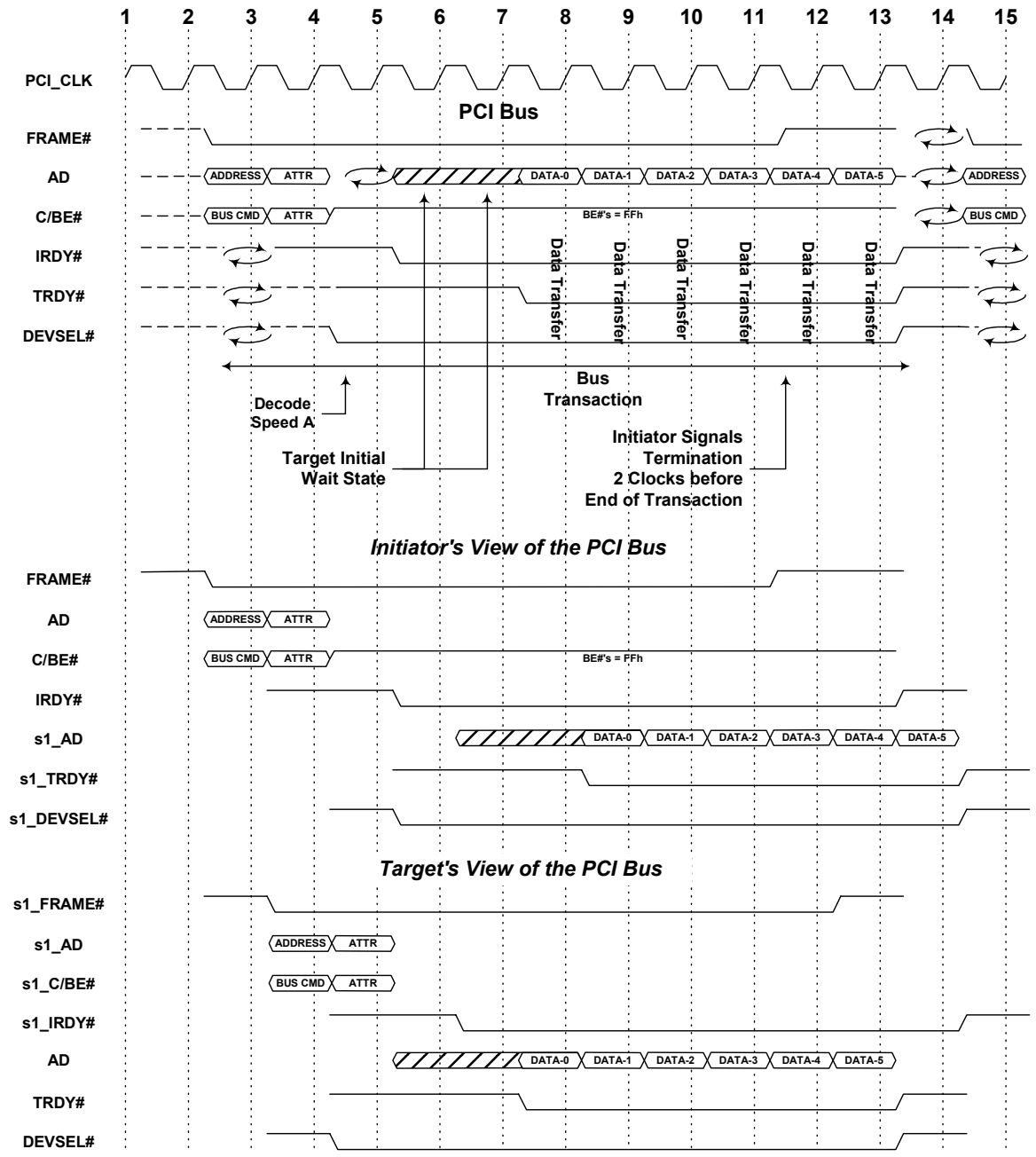

Figure 2-10: Burst Memory Read Transaction with Target Initial Wait States, Mode 1

Figure 2-11 and Figure 2-12 show the same cases in PCI-X Mode 2 as the previous two figures in PCI-X Mode 1.

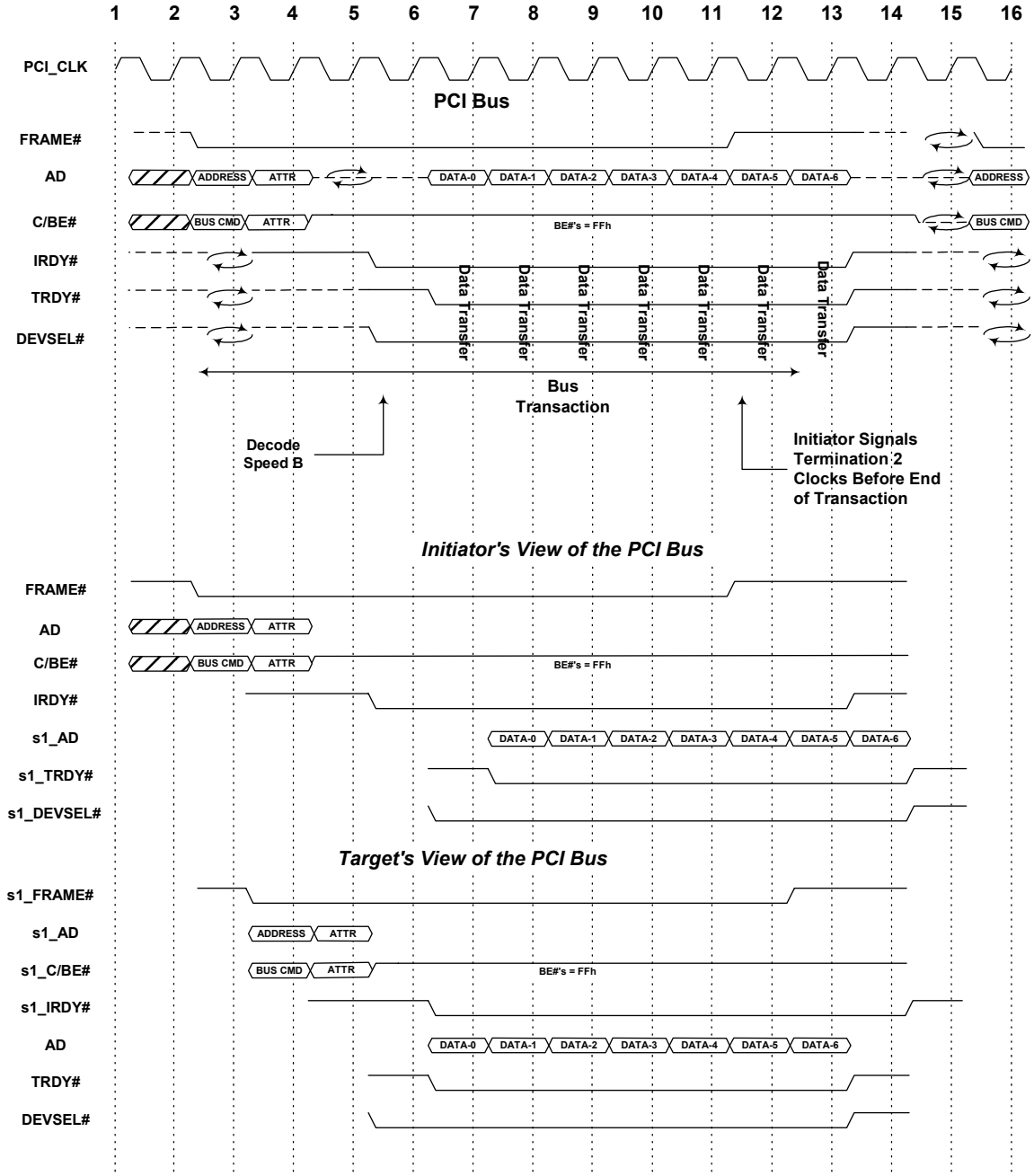

Figure 2-11: Burst Memory Read Transaction with No Target Initial Wait States, Mode 2

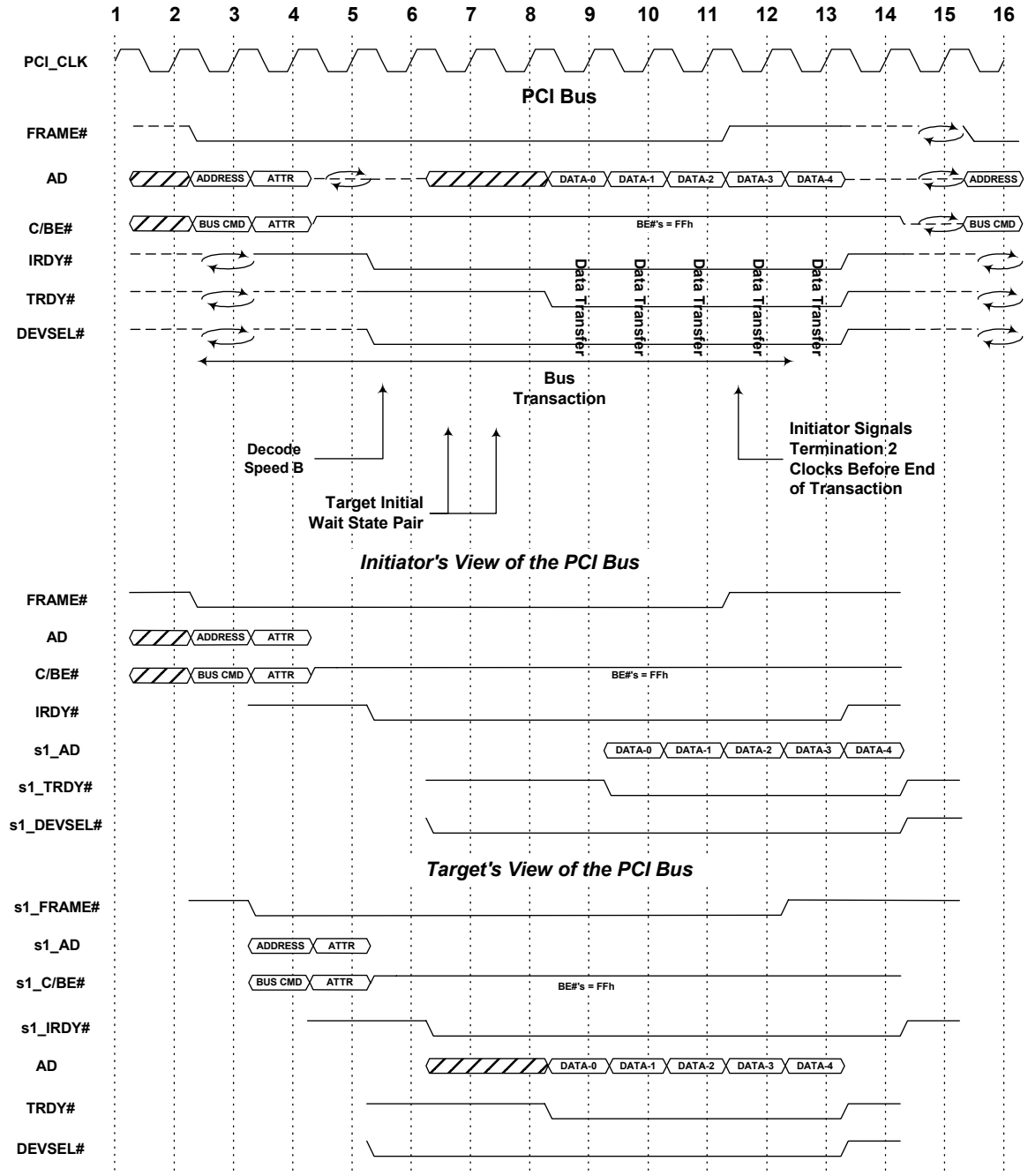

Figure 2-12: Burst Memory Read Transaction with Target Initial Wait States, Mode 2

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

A DWORD transaction always has a single data phase, except it has two data phases if all of the following are true:

- The bus is 16 bits wide.
- The transaction is not a Special Cycle
- The target signals Data Transfer, Single Data Phase Disconnect, Disconnect at Next ADB, or Split Response.

DWORD transactions always affect no more than a single DWORD. DWORD transactions do not include a byte count. The protocol for DWORD transactions in Mode 1 and Mode 2 is identical (all common-clock) except that ECC is required in Mode 2 and optional in Mode 1.

On a 64- or 32-bit interface, 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. In PCI-X Mode 2, all upper-bus receivers are disabled throughout a DWORD transaction. The upper buses are permitted to be driven or floated by the device that drives the lower buses.

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 (16 bits during each of the two data phases of a 16-bit transfer, see Table 2-22), 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 or ECC, even if some of the byte enables are deasserted. The C/BE[3::0]# 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 2-13 shows an I/O write transaction in PCI-X Mode 1 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.)

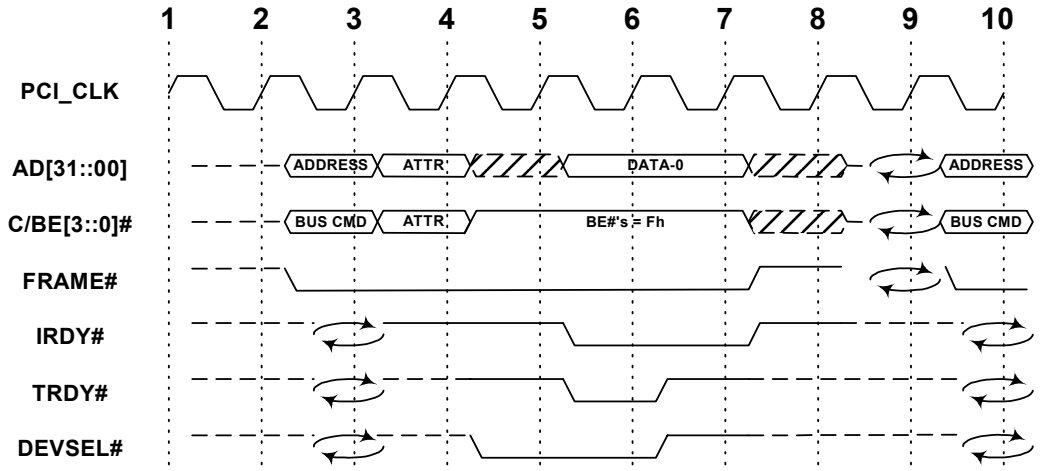

Figure 2-13: I/O Write Transaction with No Wait States and Data Transfer, Mode 1

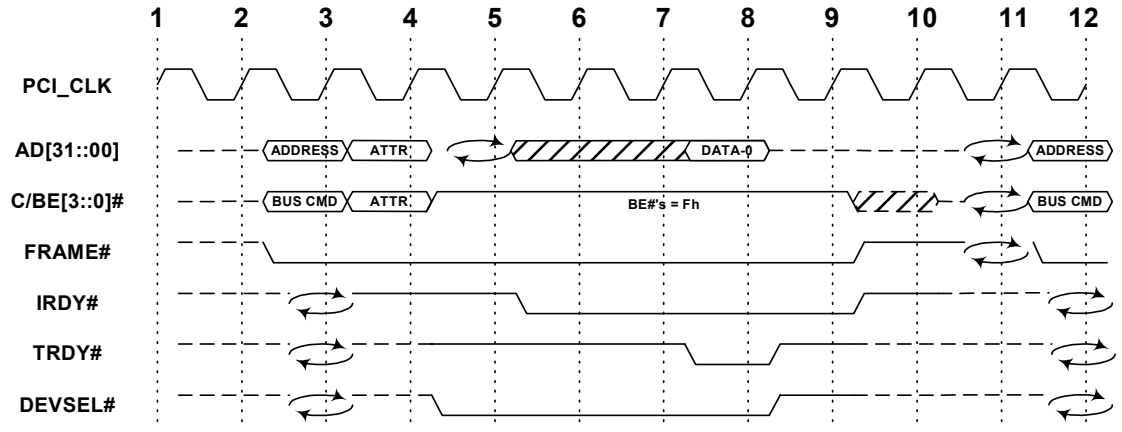

Figure 2-14: DWORD Memory or I/O Read with Two Target Initial Wait States and Data Transfer, Mode 1

Figure 2-14 shows a DWORD memory or I/O 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 2-15 and Figure 2-16 show the same examples in PCI-X Mode 2 as the previous two figures in PCI-X Mode 1.

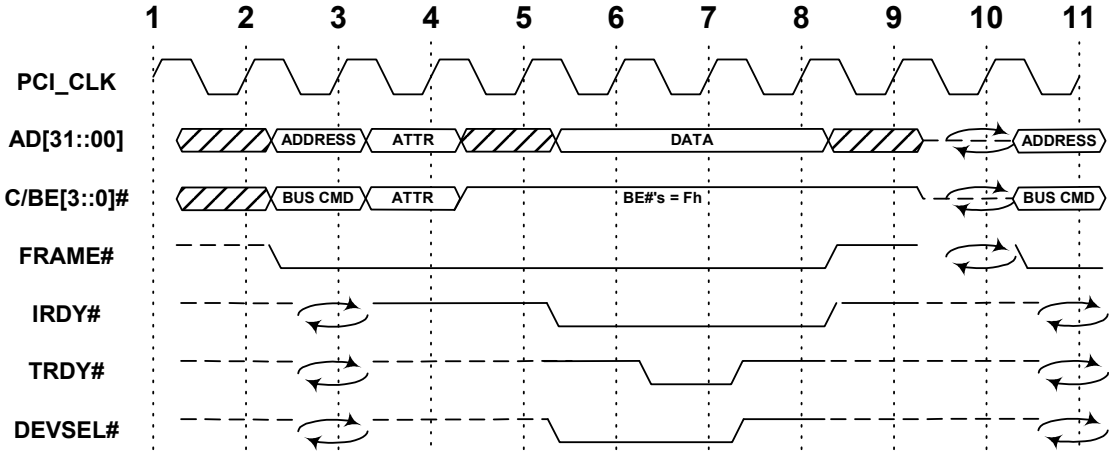

Figure 2-15: I/O Write Transaction with No Wait States and Data Transfer, Mode 2

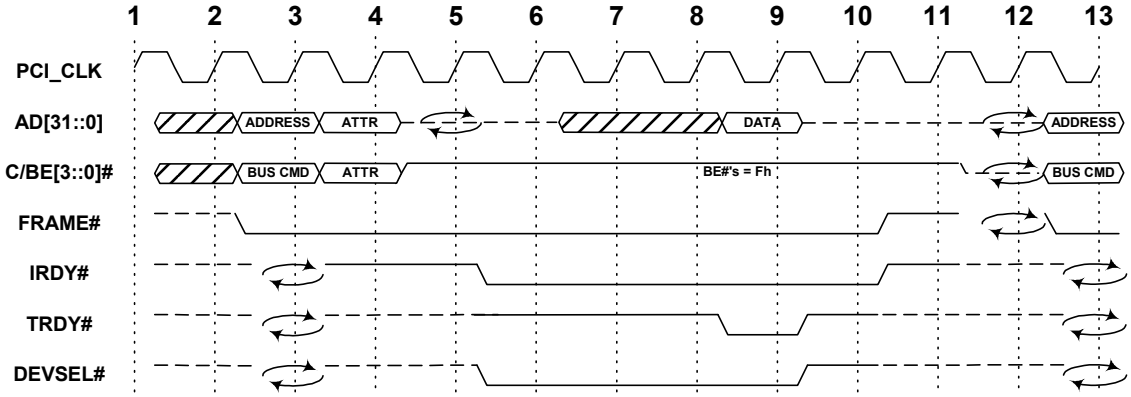

Figure 2-16: DWORD Memory or I/O Read with Two Target Initial Wait States and Data Transfer, Mode 2

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::00] bus before it asserts FRAME# on 64- and 32-bit buses. 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# on a 64- or 32-bit bus. Once FRAME# is asserted, the rest of the transaction proceeds like any other DWORD transaction.

The address predrive allows additional propagation time for the IDSEL input signal for Type 0 configuration transactions. In those 64- and 32-bit Mode 1 system boards 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. (See limitations in Section 2.3.5, "IDSEL Connection to AD Bus," in PCI-X EM 2.0.) In those system boards 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.

Electrical connection of IDSEL to an AD bit is not allowed for Mode 2 system boards for electrical reasons described in Section 2.3.5, "IDSEL Connection to AD Bus," in PCI-X EM 2.0. A 64- or 32-bit Mode 2 source bridge must provide separately decoded IDSEL output pins for each bus. See Section 2.12.2 for the use of IDSEL on a 16-bit bus.

Timing for Type 1 configuration transactions is the same as Type 0, even though IDSEL is used only for Type 0.

As with all DWORD transactions, configuration transactions (both read and write) must not affect registers if that register's byte enable is deasserted.

As for all transactions in Mode 1, 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 in Mode 1.) 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. Operation in Mode 2 is identical, except that GNT# must be asserted both in clock N-3 and N-2. (See Section 4.1.2 for a discussion of the arbiter in Mode 2.)

Figure 2-17 and Figure 2-18 illustrate a PCI-X Mode 1 Configuration Write transaction and Configuration Read transaction respectively on a 64- or 32-bit interface. Both figures show DEVSEL# decode timing A (which is allowed only in parity mode) with two target initial

wait states. All other DEVSEL# decode speeds and initial wait state combinations are also valid for configuration transactions in Mode 1.

Figure 2-17 and Figure 2-18 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.

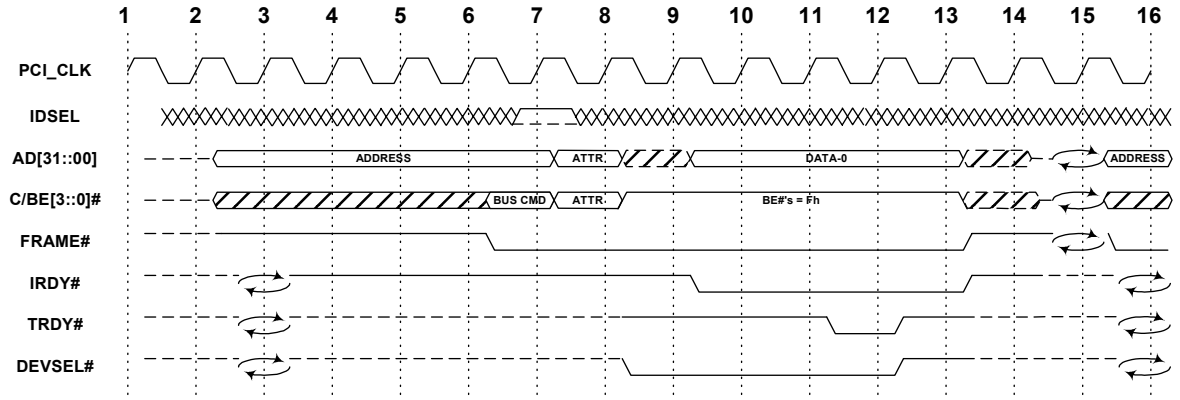

Figure 2-17: Configuration Write Transaction, Mode 1

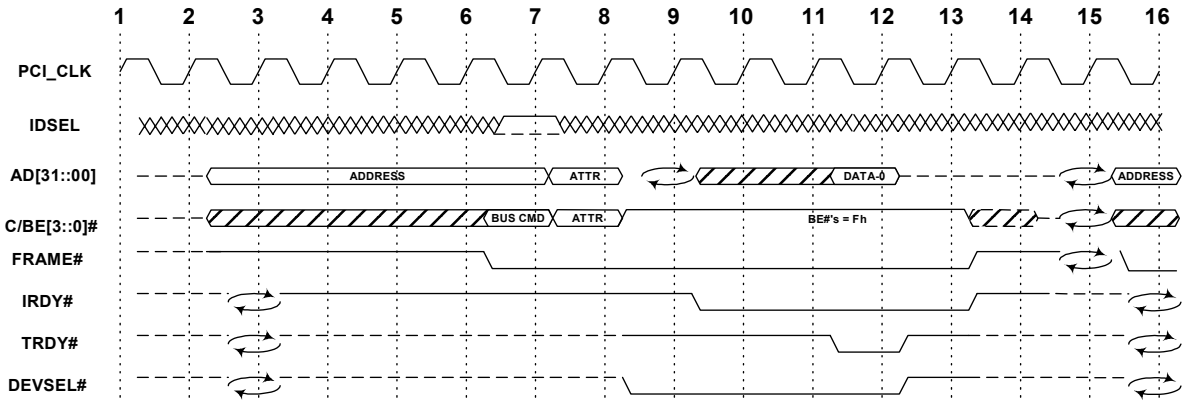

Figure 2-18: Configuration Read Transaction, Mode 1

Figure 2-19 and Figure 2-20 show the same examples in PCI-X Mode 2 as the previous two figures in PCI-X Mode 1. The Mode 2 examples use DEVSEL# decode timing B to allow for checking ECC on the address.

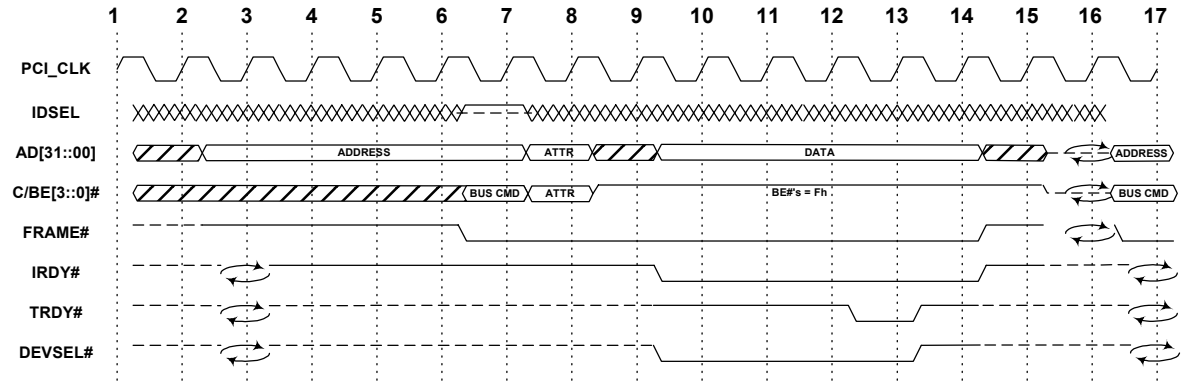

Figure 2-19: Configuration Write Transaction, Mode 2

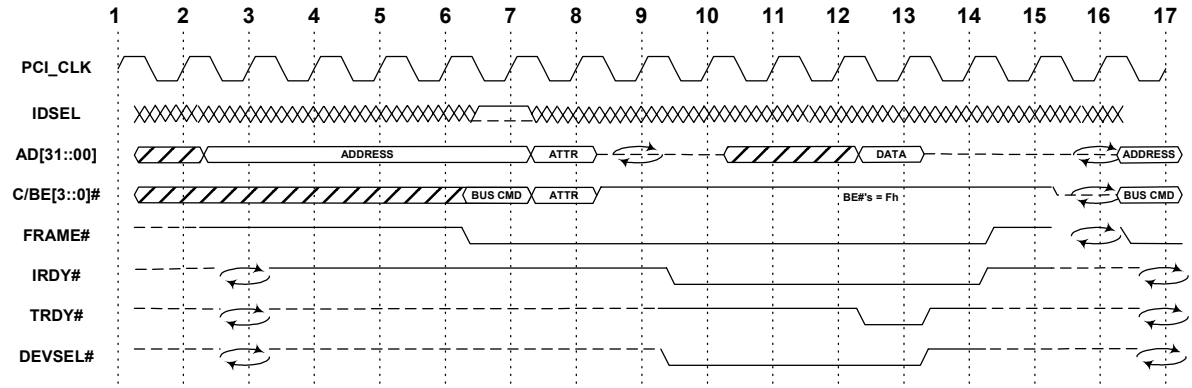

Figure 2-20: Configuration Read Transaction, Mode 2

2.7.2.2. Configuration Transaction Address and Attributes

PCI-X Mode 1 uses the same 256-byte Configuration Space as conventional PCI. PCI-X Mode 2 adds four additional bits to the Configuration Space address to expand the space to 4096 bytes.

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.

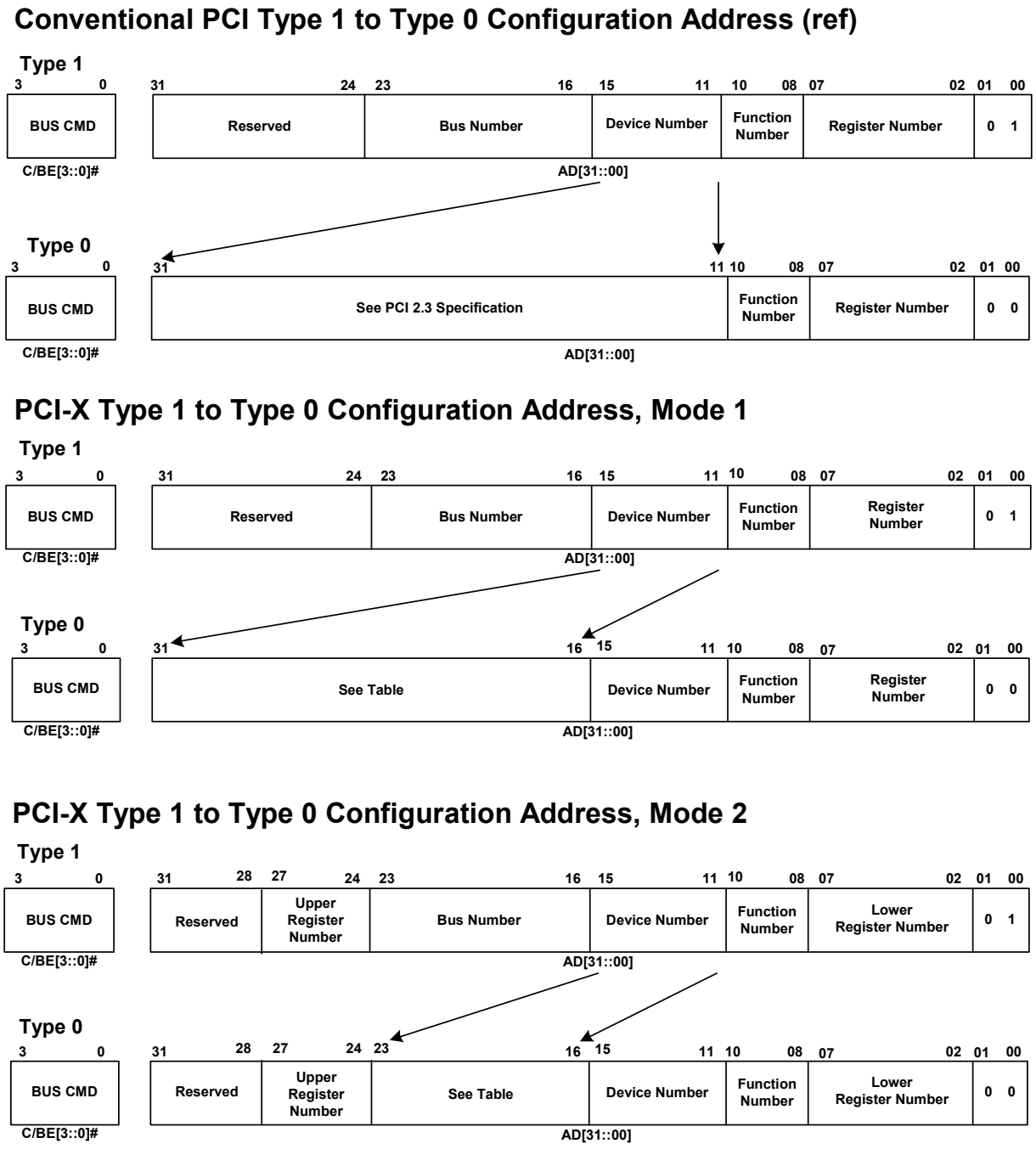

Figure 2-21: Configuration Transaction Address Format

Figure 2-21 shows the format of the address for both Type 0 and Type 1 configuration transactions in PCI-X Mode 1 and PCI-X Mode 2. 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 to 0 1111b for Mode 1 and 0 0000b-0 0111 for Mode 2) according to Table 2-10. (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, in 64- and 32-bit Mode 1 systems. (See Section 2.3.5, "IDSEL Connection to AD Bus," in PCI-X EM 2.0.) 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 2-21 also shows the additional register number bits in both Type 0 and Type 1 transactions in PCI-X Mode 2.

Table 2-10: IDSEL Generation

Device Number	Address AD[31::16], Mode 1	Address AD[23::16], Mode 2
0 0000b (Note)	0000 0000 0000 0001b	0000 0001b
0 0001b	0000 0000 0000 0010b	0000 0010b
0 0010b	0000 0000 0000 0100b	0000 0100b
0 0011b	0000 0000 0000 1000b	0000 1000b
0 0100b	0000 0000 0001 0000b	0001 0000b
0 0101b	0000 0000 0010 0000b	0010 0000b
0 0110b	0000 0000 0100 0000b	0100 0000b
0 0111b	0000 0000 1000 0000b	1000 0000b
0 1000b	0000 0001 0000 0000b	0000 0000b
0 1001b	0000 0010 0000 0000b	0000 0000b
0 1010b	0000 0100 0000 0000b	0000 0000b
0 1011b	0000 1000 0000 0000b	0000 0000b
0 1100b	0001 0000 0000 0000b	0000 0000b
0 1101b	0010 0000 0000 0000b	0000 0000b
0 1110b	0100 0000 0000 0000b	0000 0000b
0 1111b	1000 0000 0000 0000b	0000 0000b
1 xxxxb	0000 0000 0000 0000b	0000 0000b

Note:
Device number 0 is reserved for the source bridge.

PCI-X Mode 2 devices are required to support the 4096-byte Configuration Space described in Section 7.4. Mode 2 devices must respond to configuration transactions (must assert DEVSEL#) independent of the register that is being addressed within the 4096-byte Configuration Space. If the device does not implement any extended capabilities (i.e., the device does not include an Extended Capabilities List as described in Section 7.4.2), the

device responds to reads from register 100h in the 4096-byte Configuration Space as specified in Section 7.4.2.

A bridge is permitted to forward a configuration transaction (either Type 0 or Type 1) to a bus operating in Mode 1 only if the Upper Register Number bits are all zero. If a configuration transaction targets a bus operating in Mode 1 on the other side of a bridge and the Upper Register Number bits are not all zero, the bridge must treat the transaction as if it received a Master-Abort on the destination bus. That is, the bridge must do the following:

- Set the appropriate status bits for the destination bus as if the transaction had actually executed and received a Master-Abort.
- Generate a Split Completion Message indicating that the transaction was terminated with Master-Abort.
- Send the Split Completion Message to the requester on the originating bus.

Figure 2-22 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 and device ID message 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.

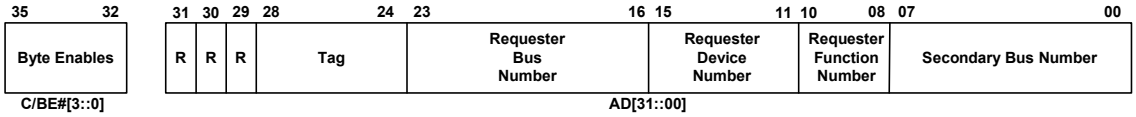

Figure 2-22: Type 0 Configuration Transaction Requester Attribute Bit Assignments

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 (two bits during each of the two address phases on a 16-bit bus, see Table 2-22) 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 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 (AD[31::16] during two address phases on a 16-bit bus) is not an address and no device asserts DEVSEL# (devices disable comparison to Base Address registers when the C/BE# bus 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 2-23 and Figure 2-24.

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.

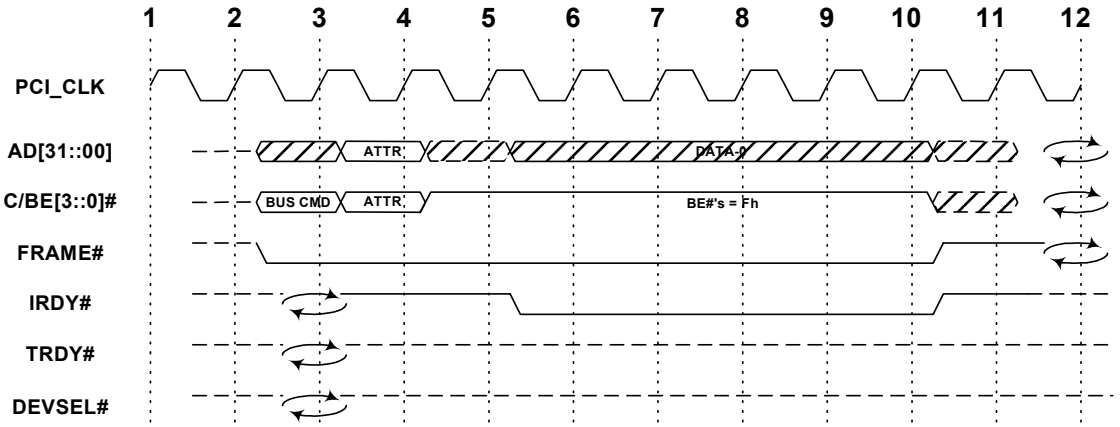

Figure 2-23: Special Cycle, Mode 1

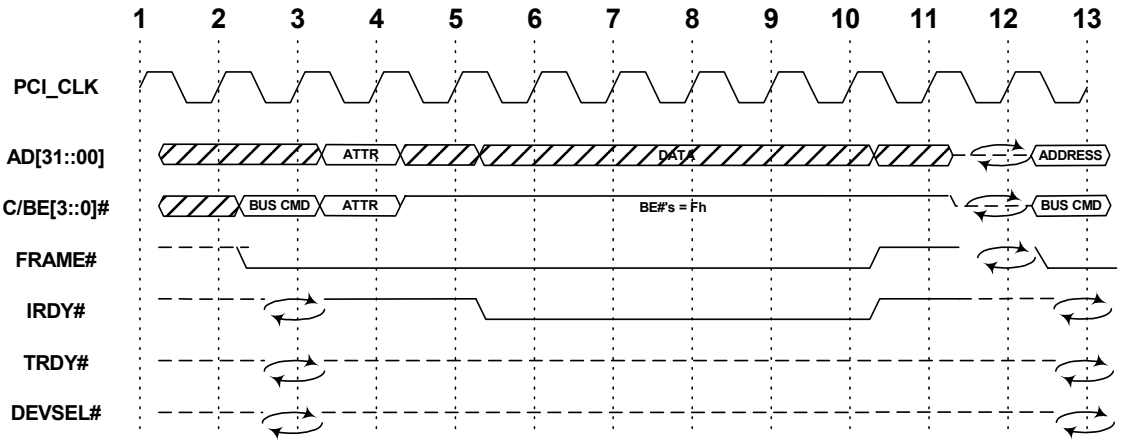

Figure 2-24: Special Cycle, Mode 2

Like conventional PCI, AD[31::00] during the data phase contain the message type and an optional data field on 64- and 32-bit buses. The message is encoded on AD[15::00] and the optional data field on AD[31::16]. On 16-bit buses only the message is used (the part that is driven on AD[15::00] on 64- and 32-bit buses), but it appears on AD[31::16] on a 16-bit bus.

The optional data field is not used on 16-bit buses. Message types for PCI-X are the same as for conventional PCI (see PCI 2.3 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). (16 AD bits and two C/BE# bits are used in two address phases on a 16-bit bus.)

The same optional methods for software to generate a Special Cycle transaction are available in PCI-X mode as defined in PCI 2.3 for conventional mode.

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 (16 bits of the AD bus during each of the two address phases on a 16-bit bus, as shown in Table 2-22). 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 (two bits of the C/BE during each of the two data phases on a 16-bit bus). 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# using decode A, B, C, or Subtractive as shown in Table 2-11 and Figure 2-25. DEVSEL# decode timing is measured the same way regardless of the transfer rate (PCI-X 66, PCI-X 133, PCI-X 266, or PCI-X 533) or width of the bus (64-, 32-, or 16-bits). (In 16-bit transactions, decode timing is measured from the second attribute phase.) Conventional DEVSEL# timing is shown in the table for reference. In ECC mode, the target is allowed to use only decode B, C, or Subtractive, to allow time to check the ECC check bits of the address before asserting DEVSEL#.

In Mode 1, the target leaves TRDY# and STOP# deasserted in the clock that it first asserts DEVSEL#. In Mode 2, the target leaves TRDY# deasserted in the clock that it first asserts

DEVSEL# and uses STOP# to indicate the transaction width as described in Section 2.12.1.3.

Table 2-11: DEVSEL# Timing

Decode Speed	PCI-X 64- and 32-Bit Bus	PCI-X 16-Bit Bus	Conventional PCI (ref)
1 clock after last address phase	Not Supported	Not Supported	Fast
2 clocks after last address phase	Decode A (Note)	Not Supported	Medium
3 clocks after last address phase	Decode B	Not Supported	Slow
4 clocks after last address phase	Decode C	Decode B	Subtractive
5 clocks after last address phase	Not Supported	Decode C	Not Supported
6 clocks after last address phase	Subtractive	Not Supported	Not Supported
7 clocks after last address phase	Not Supported	Subtractive	Not Supported

Note:
Decode speed A is not supported in ECC mode.

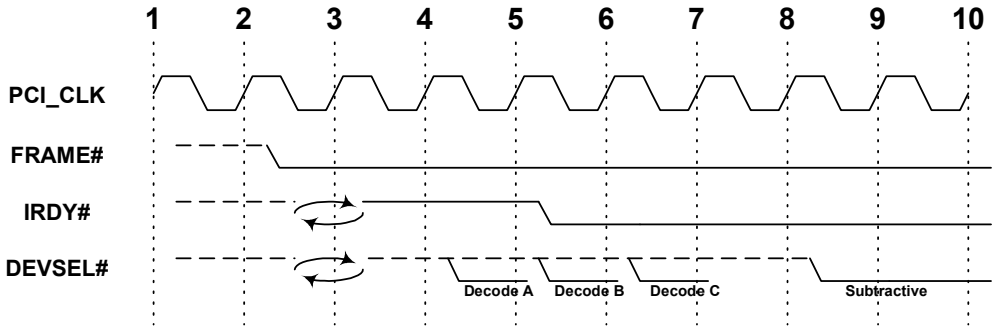

Figure 2-25: DEVSEL# Timing, Single Address Cycle, 64- or 32-Bit Bus

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. Common-Clock Burst Push and DWORD Write Transactions

The figures in this section illustrate device select timing on common-clock burst push and DWORD write transactions by using burst Memory Write transactions with four data phases. Memory Write Block, Split Completion, and Device ID Message transaction timing in PCI-X Mode 1 is identical to the transactions shown, except that the C/BE# bus is

reserved and driven high. Burst transactions of different lengths and DWORD write transactions use the same device select timing.

Most figures in this section show PCI-X Mode 1. Figure 2-26, showing DEVSEL# decode speed A, applies only to parity mode. PCI-X Mode 2 transactions differ only in the details related to floating the buses and that they use ECC (so DEVSEL# decode speed A is not supported). One Mode 2 figure is presented to illustrate the different bus drive and float requirements.

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 on a 64- or 32-bit bus, there is only one data value, which remains constant from the second clock after the attribute phase until the end of the transaction. For DWORD write transactions on a 16-bit bus, the initiator advances to the second data value in the same way as is would for a burst transaction. (See Section 2.12.2.3.2.)

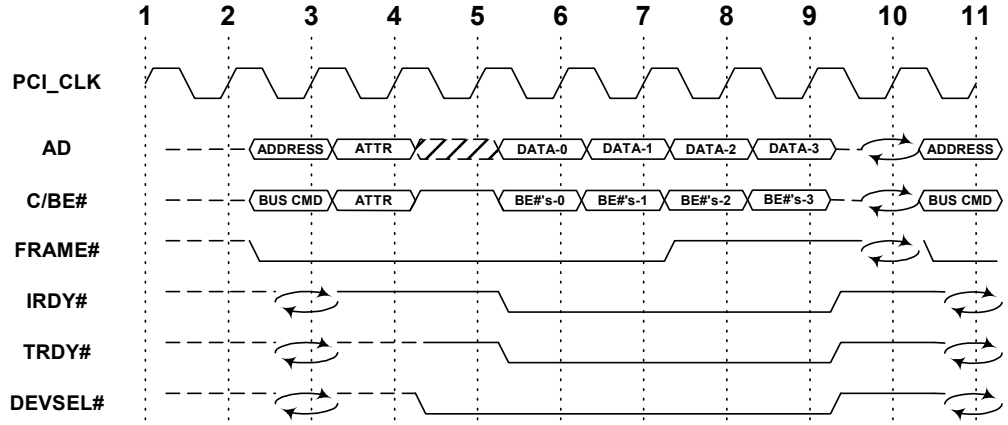

Figure 2-26: Common-Clock Burst Memory Write with DEVSEL# Decode A and No Initial Wait States, Mode 1

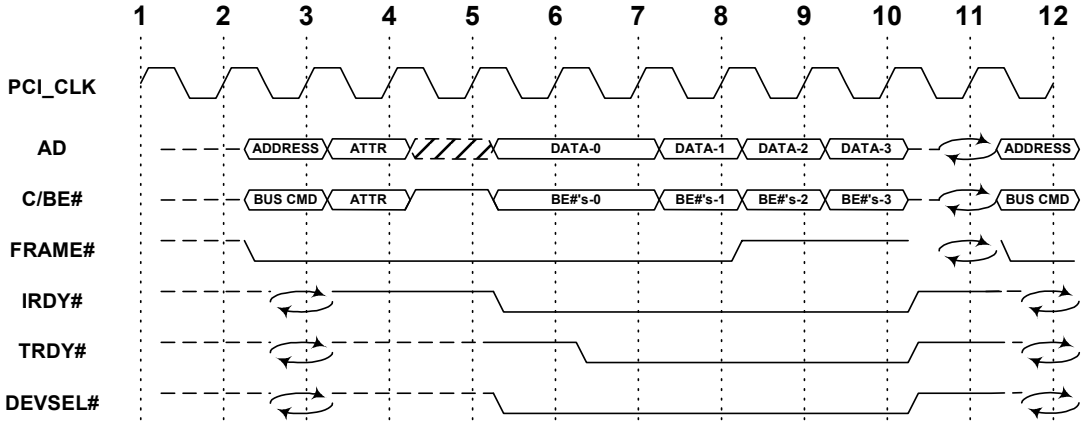

Figure 2-27: Common-Clock Burst Memory Write with DEVSEL# Decode B and No Initial Wait States, Mode 1

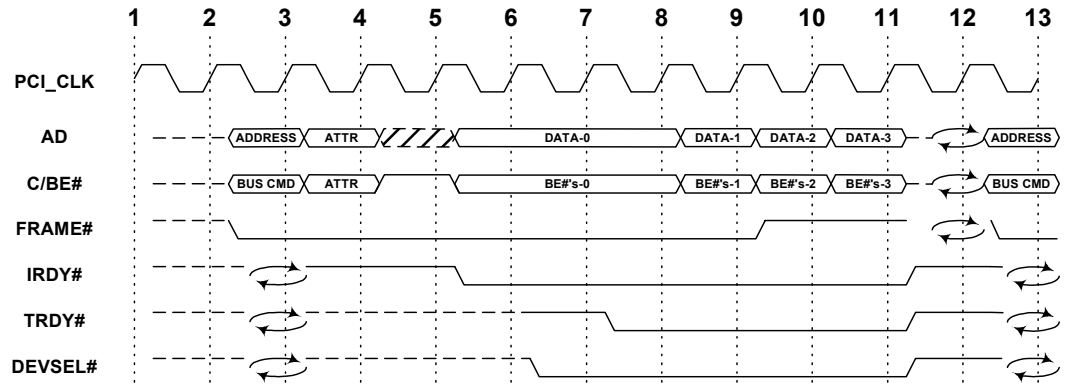

Figure 2-28: Common-Clock Burst Memory Write with DEVSEL# Decode C and No Initial Wait States, Mode 1

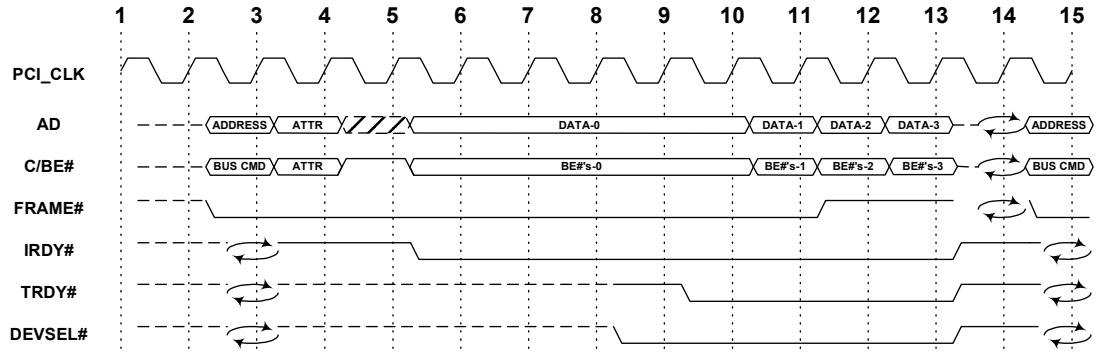

Figure 2-29: Common-Clock Burst Memory Write with Subtractive DEVSEL# Decode and No Initial Wait States, Mode 1

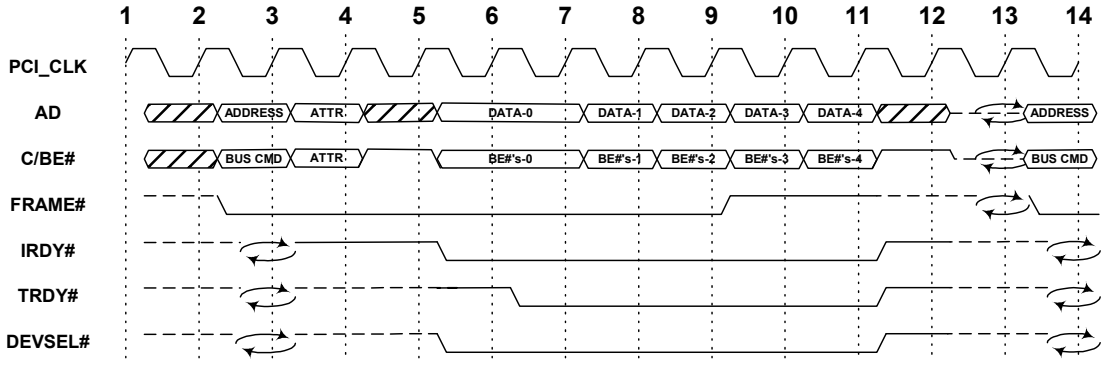

Figure 2-30: Common-Clock Burst Memory Write with DEVSEL# Decode B and No Initial Wait States, Mode 2

2.8.2. Source-Synchronous Burst Push Transactions

Transactions using the Memory Write Block, Split Completion, and Device ID Message commands in PCI-X Mode 2 are source-synchronous burst push transactions. For source-synchronous burst push, the initiator drives the first data phase value (with all its subphases) one clock after the attribute phase. The initiator advances to the second data phase value of the burst (with all of its subphases) two clocks after the target asserts DEVSEL#. Source-synchronous transactions use ECC and in ECC mode, decode speed A is not allowed. For decode speeds B (Figure 2-31), C (Figure 2-32), and Subtractive (Figure 2-33), the initiator repeats the subphases of the first data phase one, two, or four more times, respectively before advancing to the second data phase value. See Section 2.9.2 for the effects of wait states on burst push data.

The figures in this section illustrate device select timing using burst push transactions in PCI-X Mode 2 with four data phases and four subphases per data phase (PCI-X 533). In these illustrations, the initiator and target are the same width. Burst transactions of different length and burst transactions with a different number of subphases per data phase (PCI-X 266) use the same device select timing. See Section 2.12 for the case in which the initiator is 64-bits wide and the target is 32-bits wide. See Section 2.12.2 for 16-bit bus case.

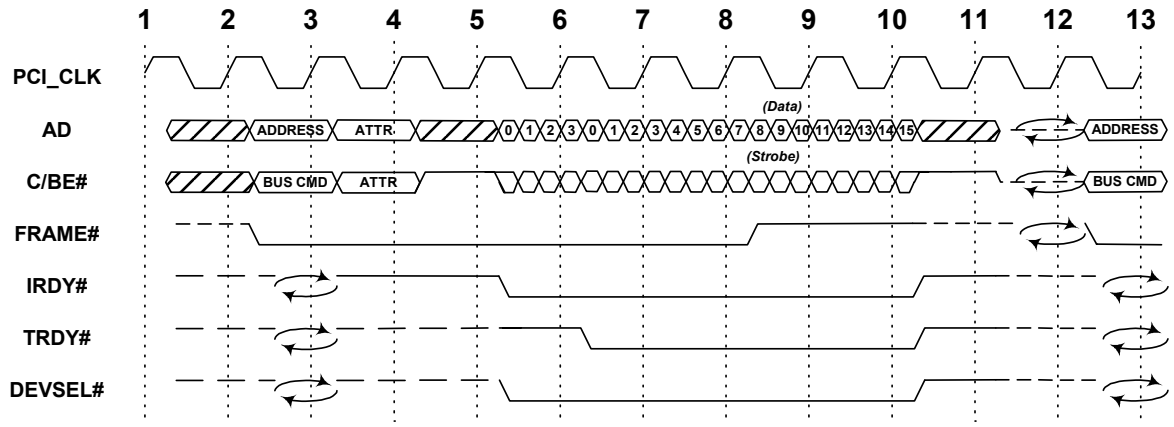

Figure 2-31: Source-Synchronous Burst Push Transaction with DEVSEL# Decode B and No Initial Wait States

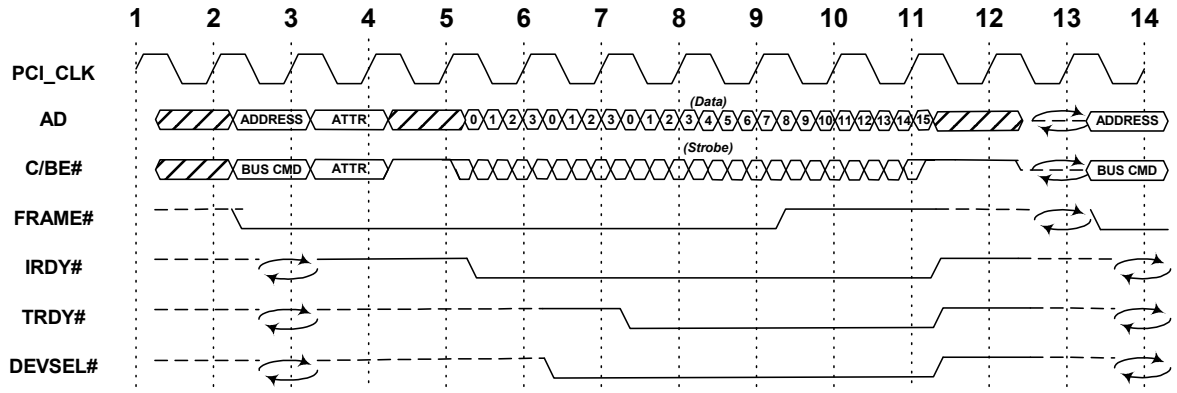

Figure 2-32: Source-Synchronous Burst Push Transaction with DEVSEL# Decode C and No Initial Wait States

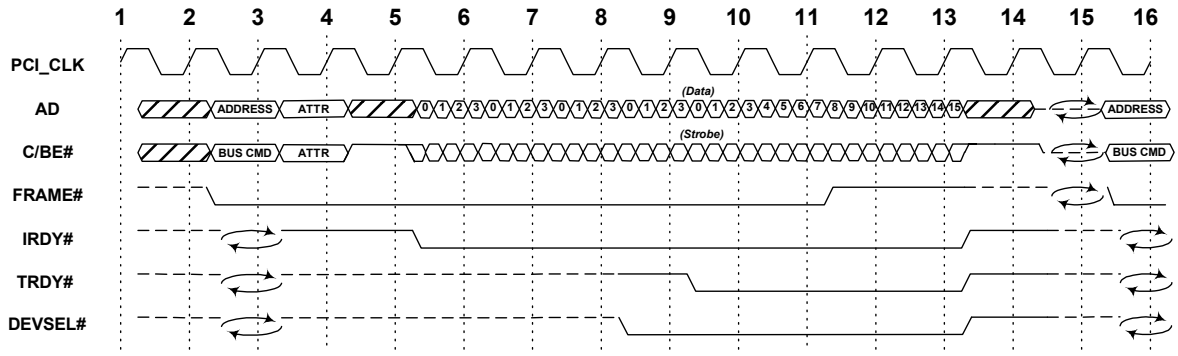

Figure 2-33: Source-Synchronous Burst Push Transaction with Subtractive DEVSEL# Decode and No Initial Wait States

2.8.3. Reads

For burst and DWORD read transactions, the C/BE# bus is reserved and driven high after the attribute phase. The target is required to begin driving the AD bus the clock after it asserts DEVSEL#. In PCI-X Mode 2, the initiator is not permitted to enable its receivers to sense the state of the AD bus until the clock after DESVEL# is asserted.

In ECC mode, the target is required to decode the address and check the address ECC before asserting DEVSEL# (decode speed A is not allowed).

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.

Most figures in this section show PCI-X Mode 1. Figure 2-34, showing DEVSEL# decode speed A, applies only to parity mode. PCI-X Mode 2 transactions differ only in the details related to floating the buses and that they use ECC (so DEVSEL# decode speed A is not

supported). One Mode 2 figure is presented to illustrate the different bus drive and float requirements. Other examples are shown in Section 2.9.3.

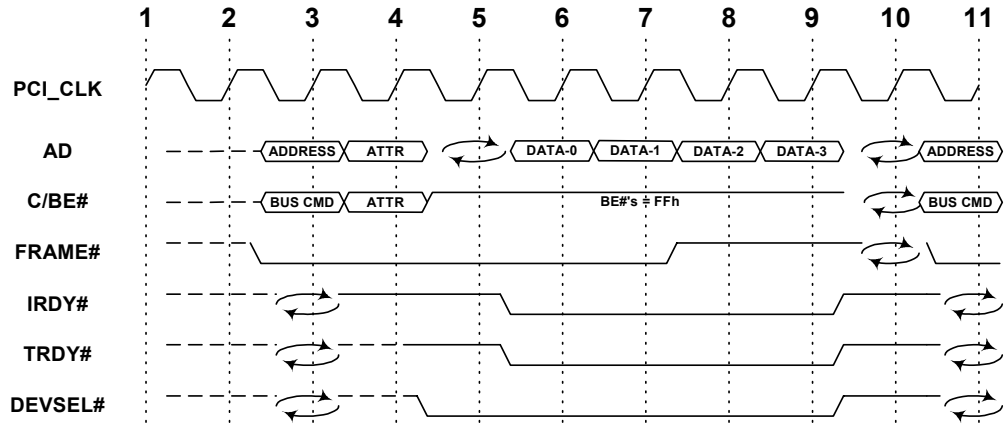

Figure 2-34: Burst Read with DEVSEL# Decode A and No Initial Wait States, Mode 1

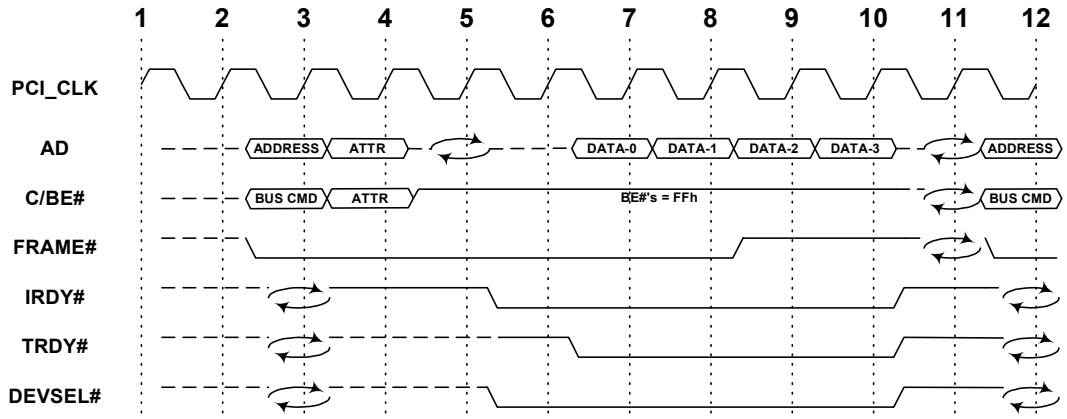

Figure 2-35: Burst Read with DEVSEL# Decode B and No Initial Wait States, Mode 1

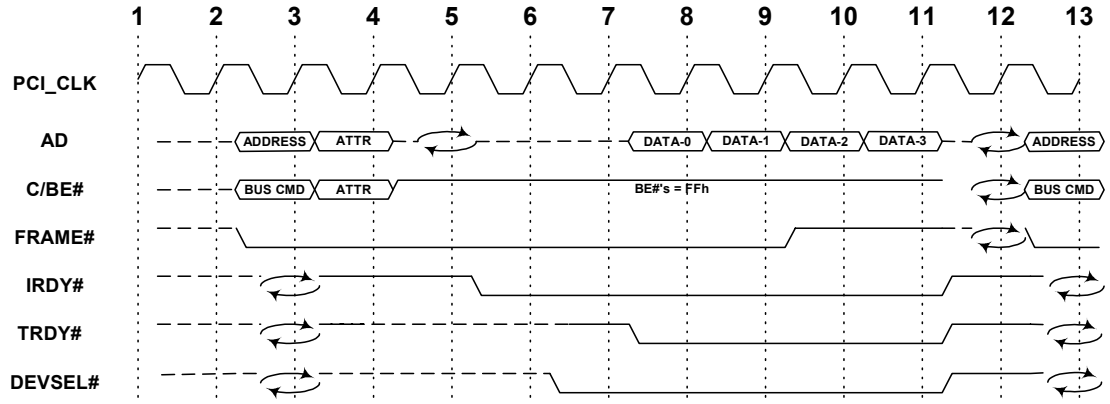

Figure 2-36: Burst Read with DEVSEL# Decode C and No Initial Wait States, Mode 1

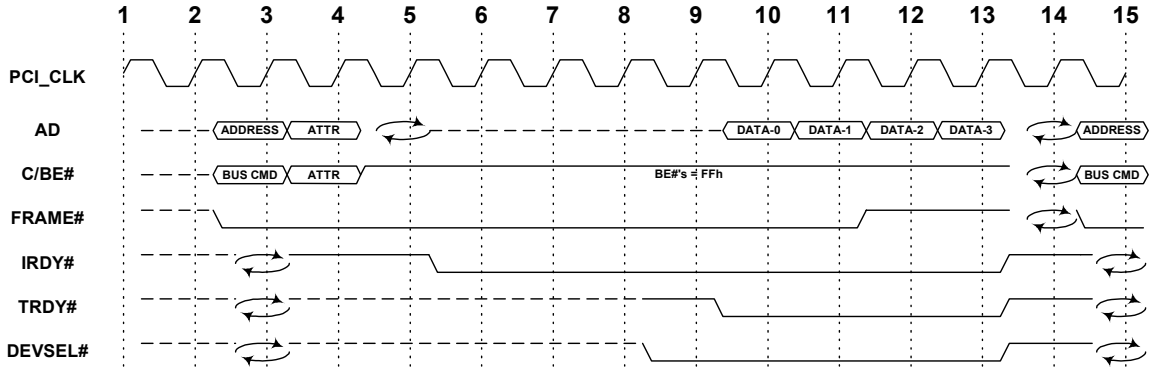

Figure 2-37: Burst Read with Subtractive DEVSEL# Decode and No Initial Wait States, Mode 1

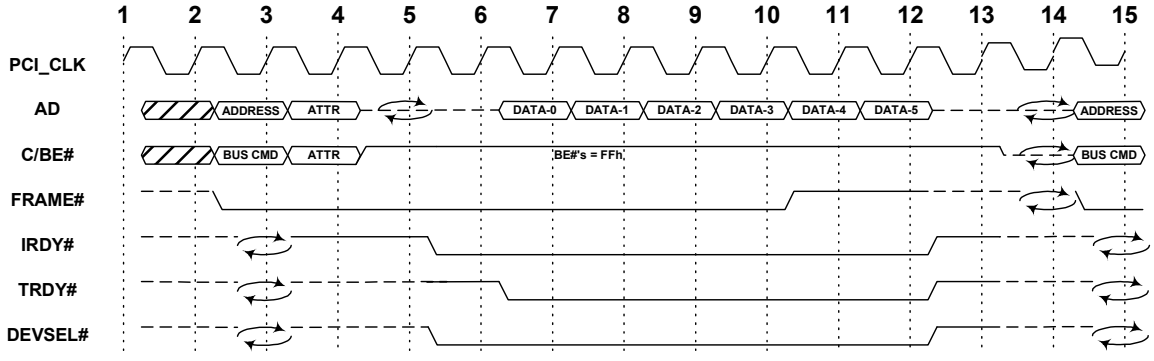

Figure 2-38: Burst Read with DEVSEL# Decode B and No Initial Wait States, Mode 2

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 (second attribute phase of a 16-bit transaction). The initiator must drive write data and in parity mode must be prepared to accept read data when IRDY# is asserted. In ECC mode, the minimum DEVSEL# decode speed is B, so the initiator must be able to accept read data one clock after 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 in the following cases:

- Burst write and Split Completion transactions for which data transfers (the target signals Data Transfer, Disconnect at Next ADB, or Single Data Phase Disconnect).
- On a 16-bit bus, DWORD write and burst push transactions for which data transfers (the target signals Data Transfer, Disconnect at Next ADB, or Single Data Phase Disconnect).

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 in Section 2.9.1). (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.)

In ECC mode, a 32-bit target of a burst push transaction from a 64-bit initiator must insert a minimum of two wait states for transactions in which data transfers (the target signals Data Transfer, Disconnect at Next ADB, or Single Data Phase Disconnect). (A 64-bit initiator uses this time to recalculate the ECC check bits for a 32-bit transfer, and to shift the data down to the lower bus, as described in Section 2.12.1.3.) For all other transactions (including burst push transactions, for initiators and targets of any size, 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 in Table 2-12).

2.9.1. Target Initial Latency

As defined in PCI 2.3, 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 2-12 shows the target initial latency for all the combinations of DEVSEL# timing, numbers of address phases, and numbers of wait states.

Table 2-12: Target Initial Latency

	64- and 32-Bit Transfers								16-Bit Transfers					
	Single Address Cycle DEVSEL# Timing				Dual Address Cycle DEVSEL# Timing				Single Address Cycle DEVSEL# Timing			Dual Address Cycle DEVSEL# Timing		
Wait States	A	B	C	Sub	A	B	C	Sub	B	C	Sub	B	C	Sub
0	3	4	5	7	4	5	6	8	6	7	9	8	9	11
1	4	5	6	8	5	6	7	9	7	8	10	9	10	12
2	5	6	7	9	6	7	8	10	8	9	11	10	11	13
3	6	7	8	10	7	8	9	11	9	10	12	11	12	14
4	7	8	9	11	8	9	10	12	10	11	13	12	13	15
5	8	9	10	12	9	10	11	13	11	12	14	13	14	16
6	9	10	11	13	10	11	12	14	12	13	15	14	15	17
7	10	11	12	14	11	12	13	15	13	14	16	15	16	18
8	11	12	13	15	12	13	14	16	14	15	17	16	17	N/A
9	12	13	14	16	13	14	15	N/A	15	16	18	17	18	N/A
10	13	14	15	N/A	14	15	16	N/A	16	17	N/A	18	N/A	N/A
11	14	15	16	N/A	15	16	N/A	N/A	17	18	N/A	N/A	N/A	N/A
12	15	16	N/A	N/A	16	N/A	N/A	N/A	18	N/A	N/A	N/A	N/A	N/A
13	16	N/A	N/A	N/A	N/A	N/A	N/A	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 and whether the bus is 16-bit wide. If the target signals Split Response or Retry, it must do so within eight clocks of the assertion of FRAME# for 64- and 32-bit transfers, and 11 clocks for 16-bit transfers. If the target signals Target-Abort in the first data phase for reasons other than an error that prevents transferring the first data, it must do so within eight clocks of the assertion of FRAME# (10 clocks for 16-bit transfers). These cases are outlined with a heavy line in Table 2-12. 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# for 64- and 32-bit transfers, and 18 clocks for 16-bit transfers. 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# (18 clocks for 16-bit transfers). 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 2-7, "3.3V General Timing Parameters," in PCI-X EM 2.0) 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 1.1, 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 2.3 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 2-12. (Same as PCI 2.3 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 2-12, 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 2-12. 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 2.3 except the latency limit is lower.)

Option 3: The device frequently cannot transfer data within the target initial latency limit in Table 2-12. 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 2-12, 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 Burst Push and DWORD Write Transactions

For common-clock burst push and DWORD write transactions, the initiator must drive data on the AD bus two clocks after the attribute phase. If the transaction is 16 bits wide or 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 2.12.1.3 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. For source-synchronous burst push transactions, the initiator drives, repeats, and advances data phases in the same manner as common-clock burst transactions, except that each data phase includes two or four subphases.

The starting address and byte count of some burst push transactions are 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 phase.

For DWORD write transactions on a 64- or 32-bit interface, 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. DWORD transactions on a 16-bit interface have two data phases and behave the same as a common-clock burst push transaction with two data phases.

The following figures illustrate transactions with a variety of combinations of DEVSEL# timing and wait states. PCI-X Mode 1 illustrations appear first, followed by Mode 2 illustrations. All of the Mode 2 illustrations in this section use source-synchronous transactions. Mode 2 common-clock transactions use the data pattern requirements illustrated in the Mode 1 examples and the signal drive and float requirements illustrated in the Mode 2 source-synchronous examples. The Mode 2 illustrations assume the initiator and

the target are the same width. See Section 2.12 for the requirements when a 64-bit initiator addresses a 32-bit target in PCI-X Mode 2.

Figure 2-39 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.

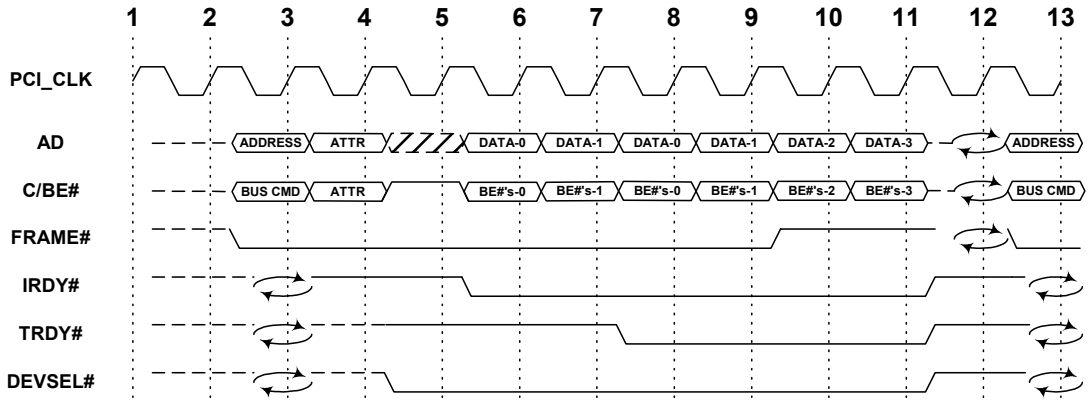

Figure 2-39: Burst Memory Write Transaction with DEVSEL# Decode A and Two Initial Wait States, Mode 1

Figure 2-40 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.

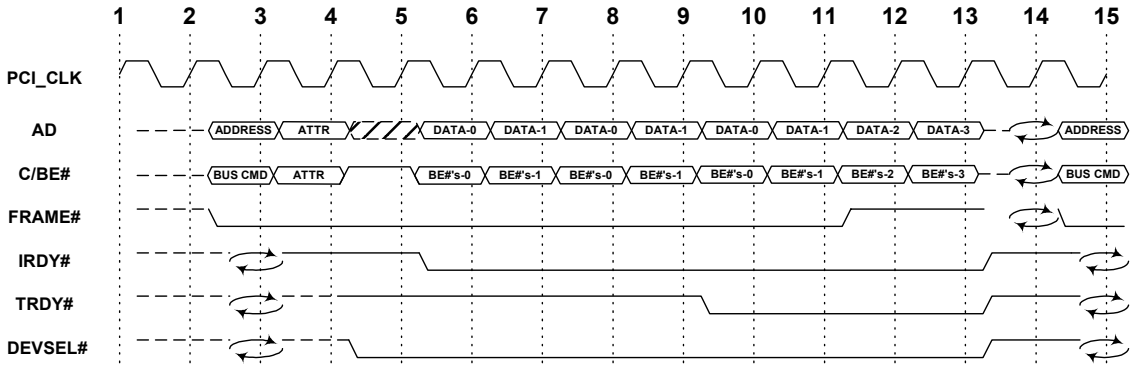

Figure 2-40: Burst Memory Write Transaction with DEVSEL# Decode A and Four Initial Wait States, Mode 1

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 2-41 and Figure 2-42 show device select timing B, and Figure 2-43 and Figure 2-44 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. Figure 2-47 through Figure 2-50 show the same set of cases in PCI-X Mode 2 with four subphases per data phase (PCI-X 533).

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 push 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 2-12.

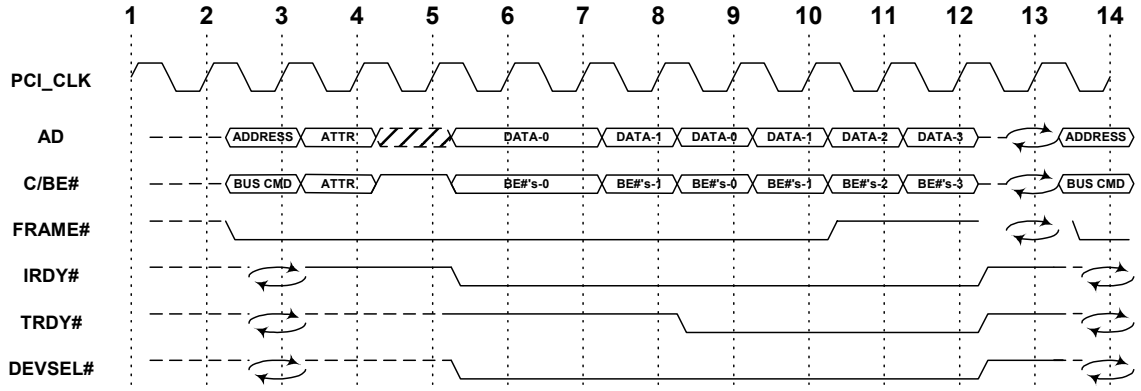

Figure 2-41: Burst Memory Write Transaction with DEVSEL# Decode B and Two Initial Wait States, Mode 1

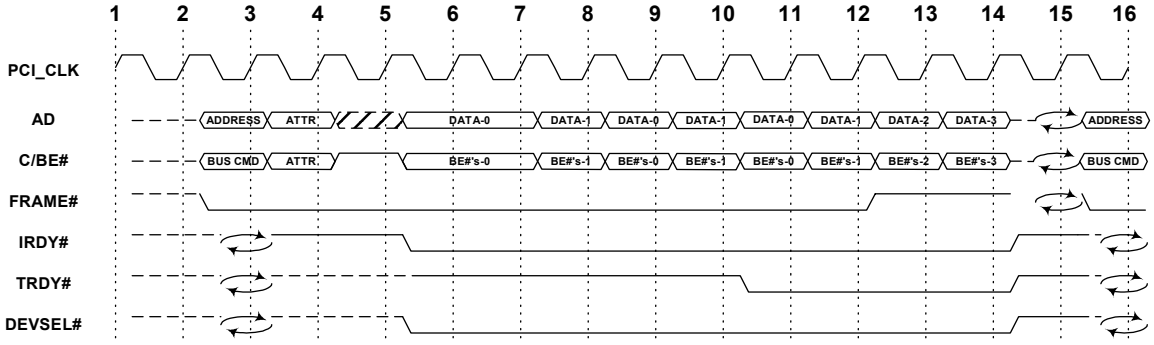

Figure 2-42: Burst Memory Write Transaction with DEVSEL# Decode B and Four Initial Wait States, Mode 1

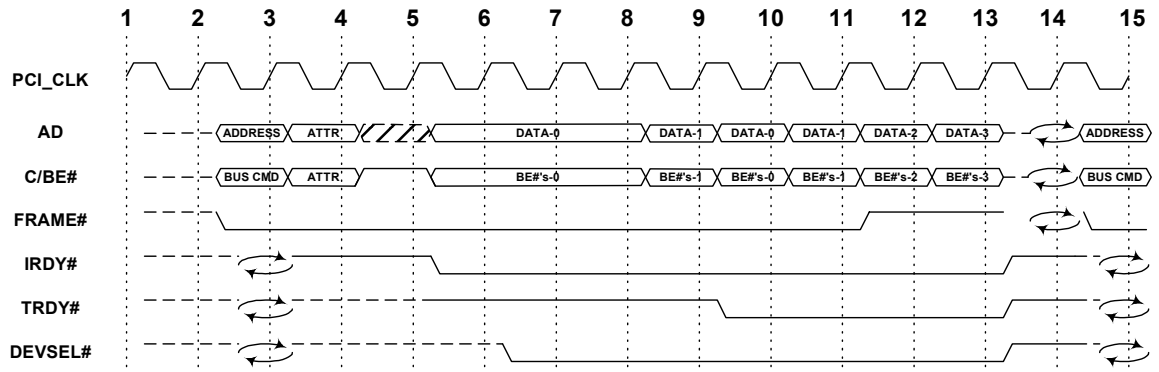

Figure 2-43: Burst Memory Write Transaction with DEVSEL# Decode C and Two Initial Wait States, Mode 1

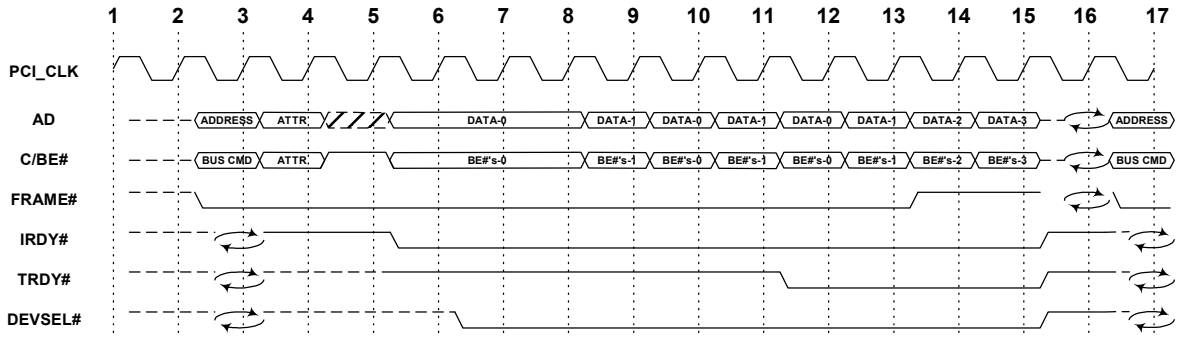

Figure 2-44: Burst Memory Write Transaction with DEVSEL# Decode C and Four Initial Wait States, Mode 1

Figure 2-45 and Figure 2-46 show the effects of device select timing and wait states on DWORD write transactions on a 64- or 32-bit bus. 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 2-51 shows the same case in Mode 2 for decode speed C. See Section 2.12.2.3.2 for 16-bit bus examples.

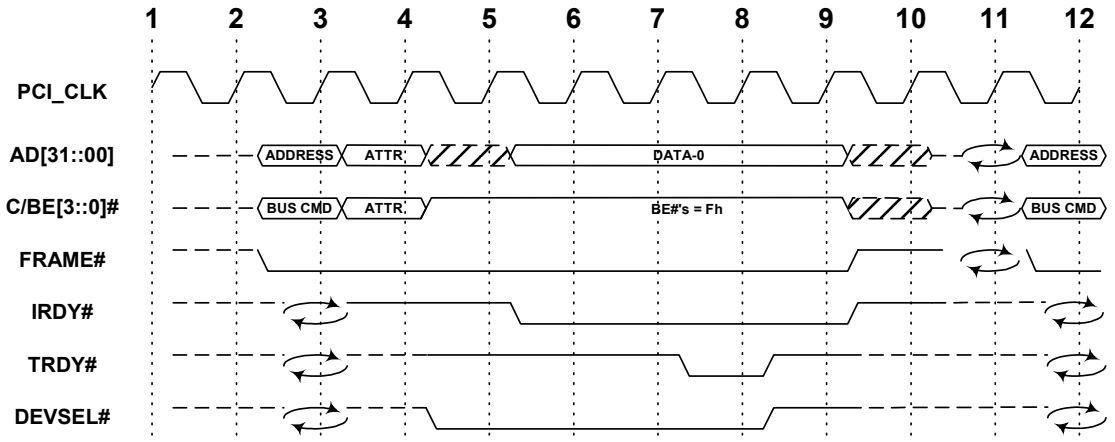

Figure 2-45: DWORD Write Transaction with DEVSEL# Decode A and Two Initial Wait States, Mode 1

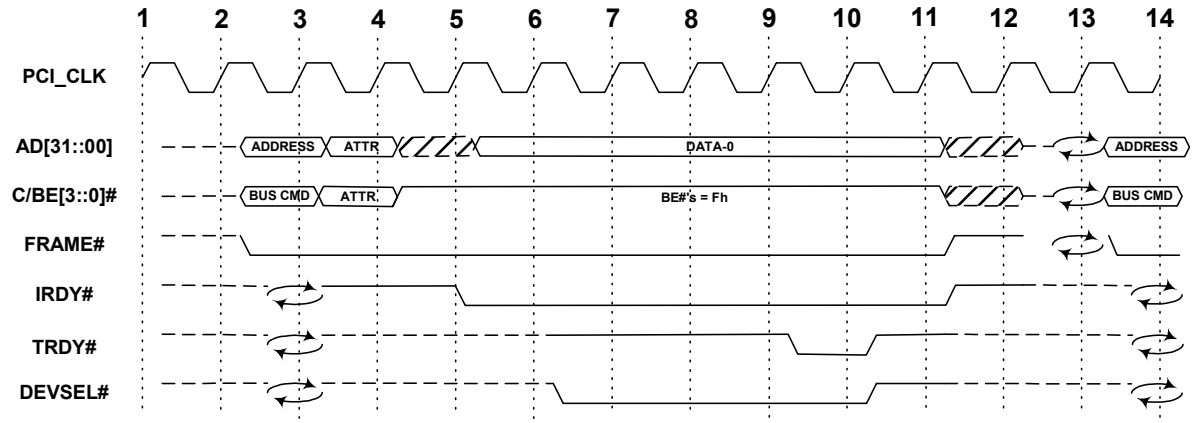

Figure 2-46: DWORD Write Transaction with DEVSEL# Decode C and Two Initial Wait States, Mode 1

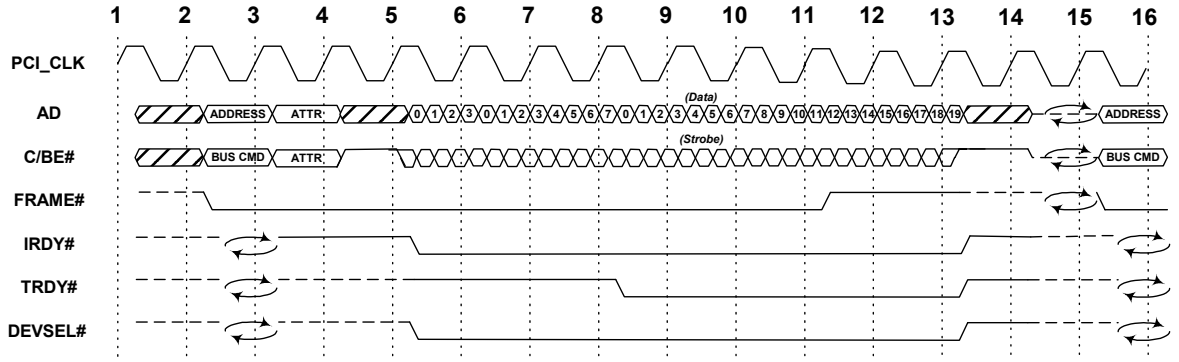

Figure 2-47: Source-Synchronous Burst Push Transaction with DEVSEL# Decode B and Two Initial Wait States, Mode 2

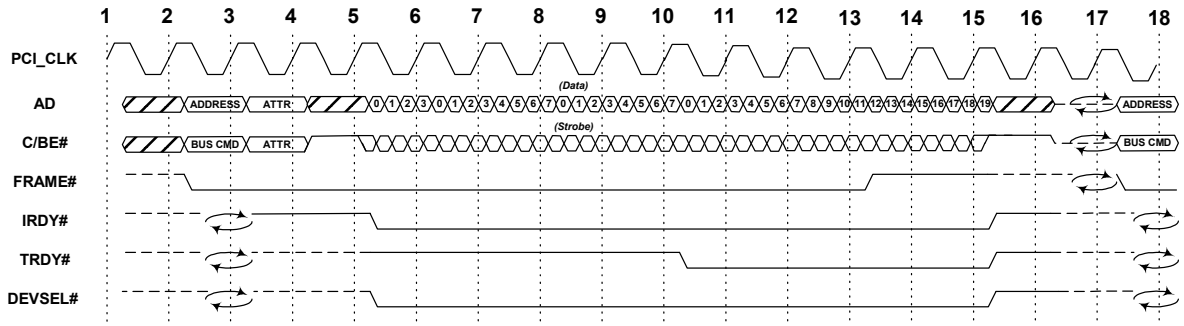

Figure 2-48: Source-Synchronous Burst Push Transaction with DEVSEL# Decode B and Four Initial Wait States, Mode 2

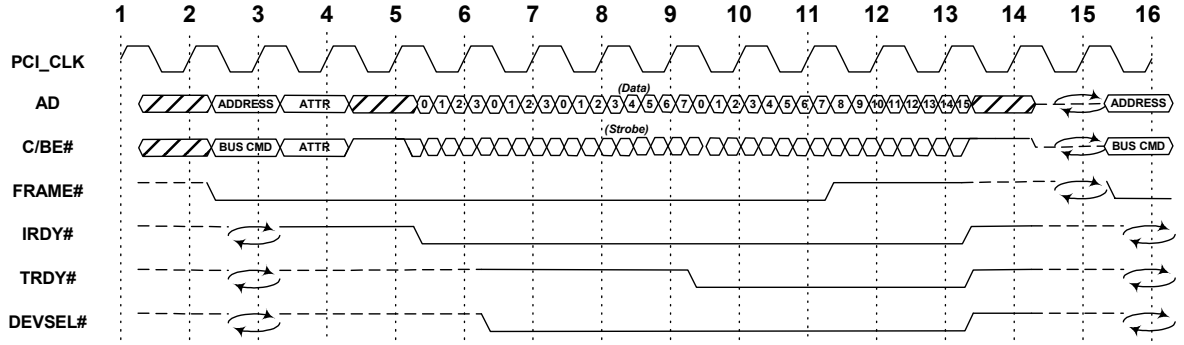

Figure 2-49: Source-Synchronous Burst Push Transaction with DEVSEL# Decode C and Two Initial Wait States, Mode 2

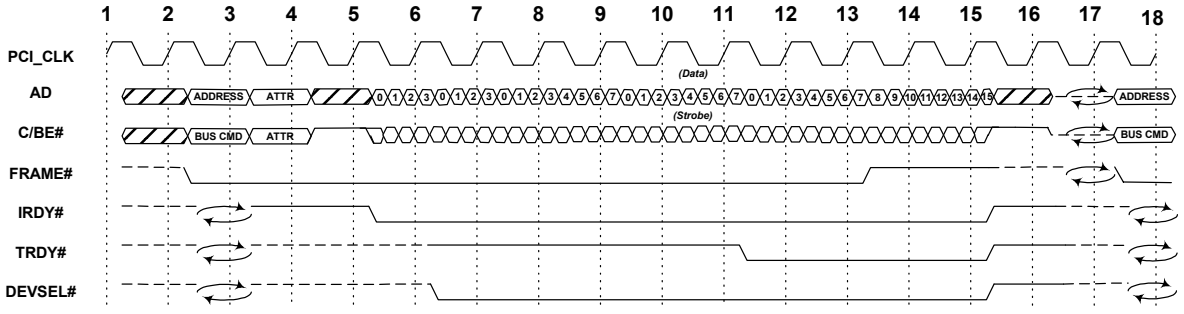

Figure 2-50: Source-Synchronous Burst Push Transaction with DEVSEL# Decode C and Four Initial Wait States, Mode 2

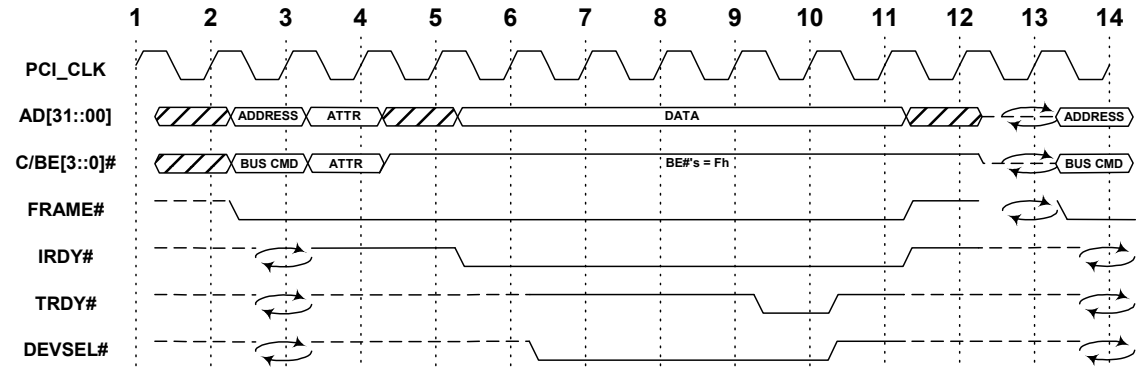

Figure 2-51: DWORD Write Transaction with DEVSEL# Decode C and Two Initial Wait States, Mode 2

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 target is required to begin driving the AD bus the clock after it asserts DEVSEL#, even though the data is not required to be valid until the clock in which TRDY# is asserted.

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 2-52 through Figure 2-56 show burst read transactions in which data is transferred (Immediate Transactions) in PCI-X Mode 1. Figure 2-57 and Figure 2-58 show DWORD read transactions in which data is transferred (Immediate Transactions) in PCI-X Mode 1 on a 64- or 32-bit bus. Figure 2-59 through Figure 2-63 show the analogous cases in PCI-X Mode 2. 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.

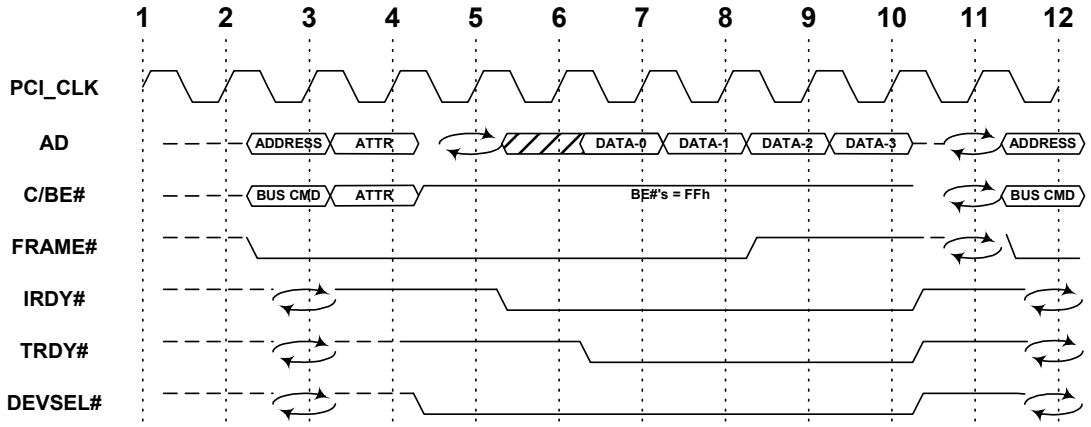

Figure 2-52: Burst Read Transaction with DEVSEL# Decode A and One Initial Wait State, Mode 1

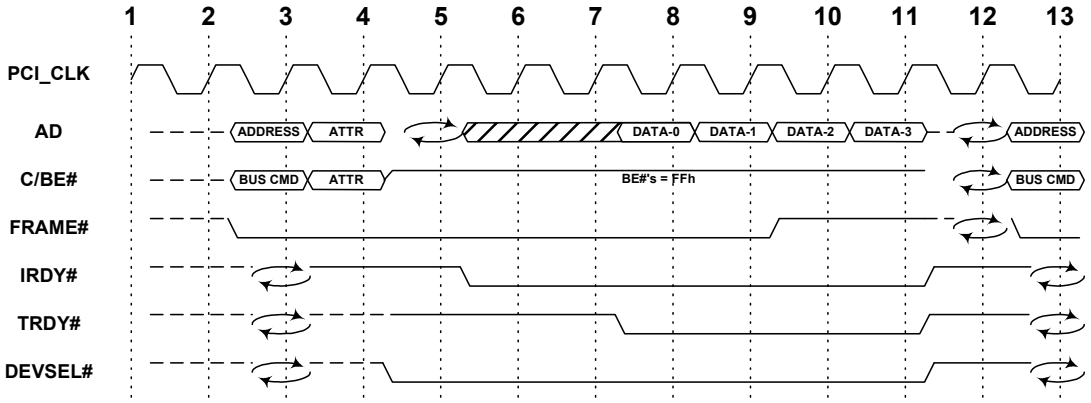

Figure 2-53: Burst Read Transaction with DEVSEL# Decode A and Two Initial Wait States, Mode 1

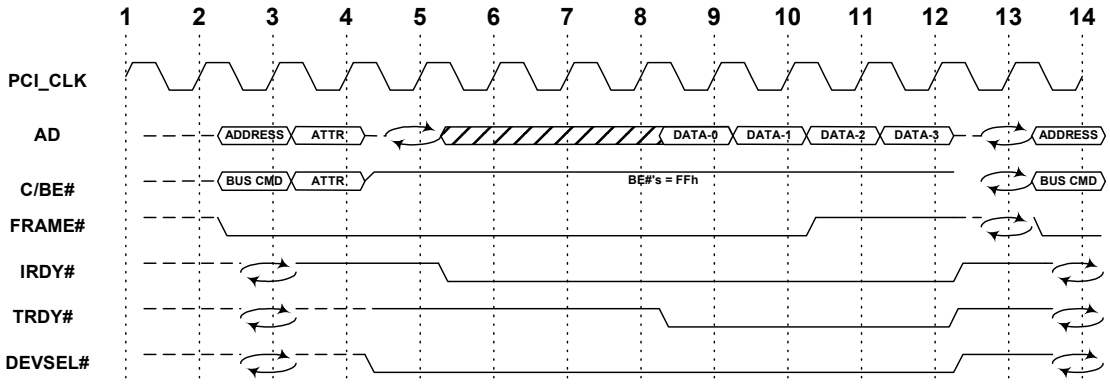

Figure 2-54: Burst Read Transaction with DEVSEL# Decode A and Three Initial Wait States, Mode 1

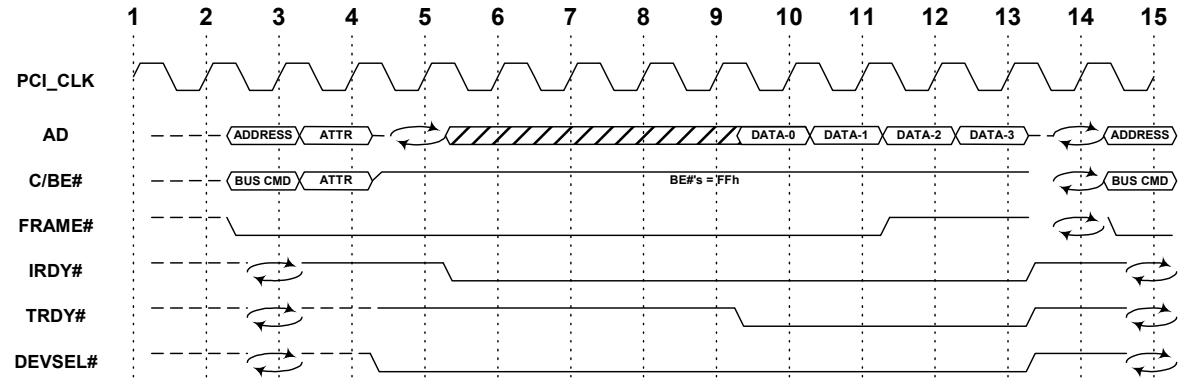

Figure 2-55: Burst Read Transaction with DEVSEL# Decode A and Four Initial Wait States, Mode 1

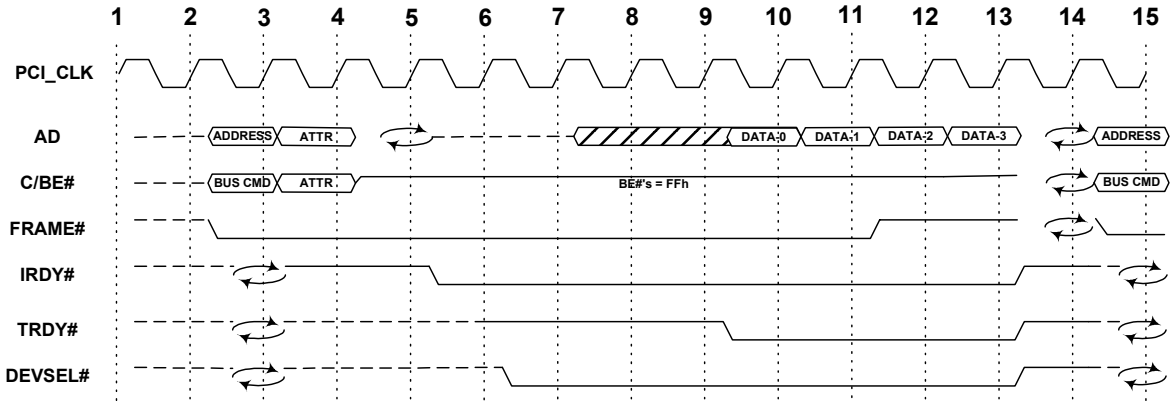

Figure 2-56: Burst Read Transaction with DEVSEL# Decode C and Two Initial Wait States, Mode 1

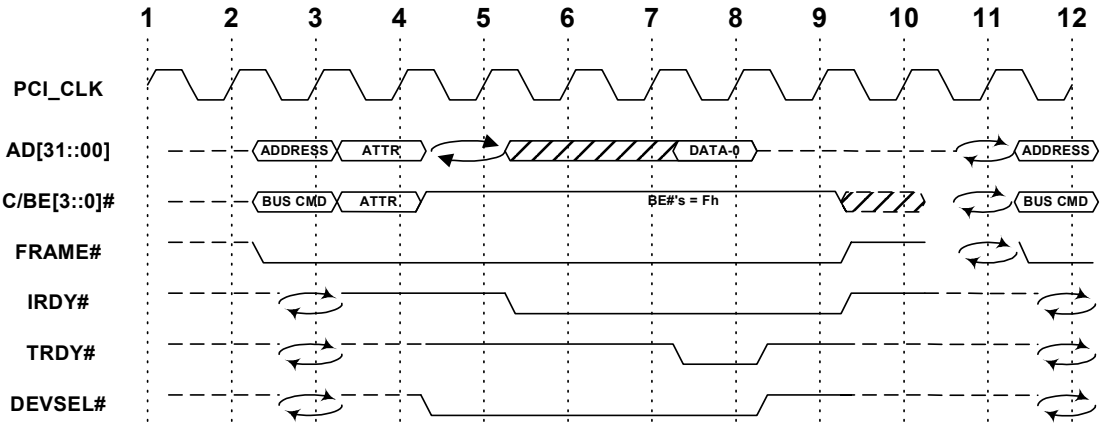

Figure 2-57: DWORD Read Transaction with DEVSEL# Decode A and Two Initial Wait States, Mode 1

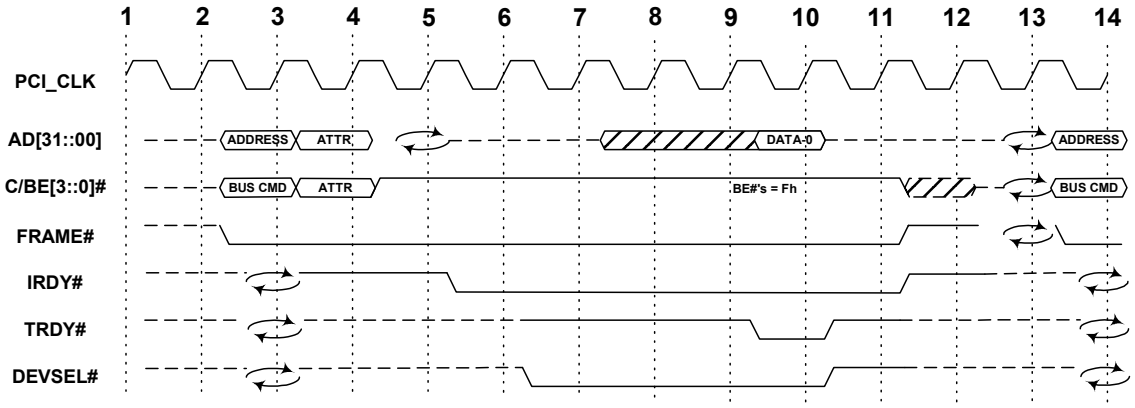

Figure 2-58: DWORD Read Transaction with DEVSEL# Decode C and Two Initial Wait States, Mode 1

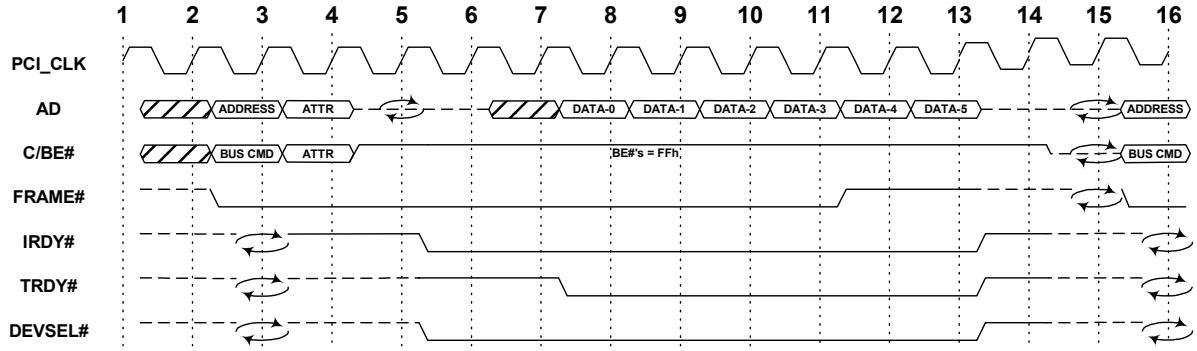

Figure 2-59: Burst Read Transaction with DEVSEL# Decode B and One Initial Wait State, Mode 2

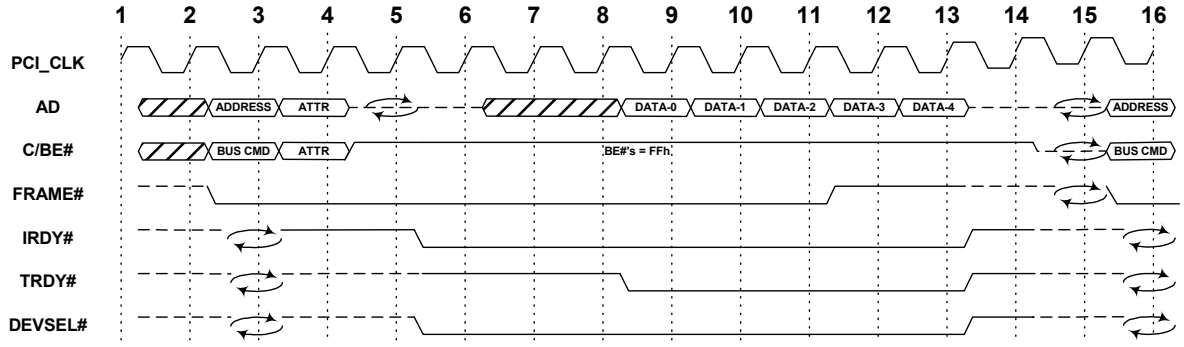

Figure 2-60: Burst Read Transaction with DEVSEL# Decode B and Two Initial Wait States, Mode 2

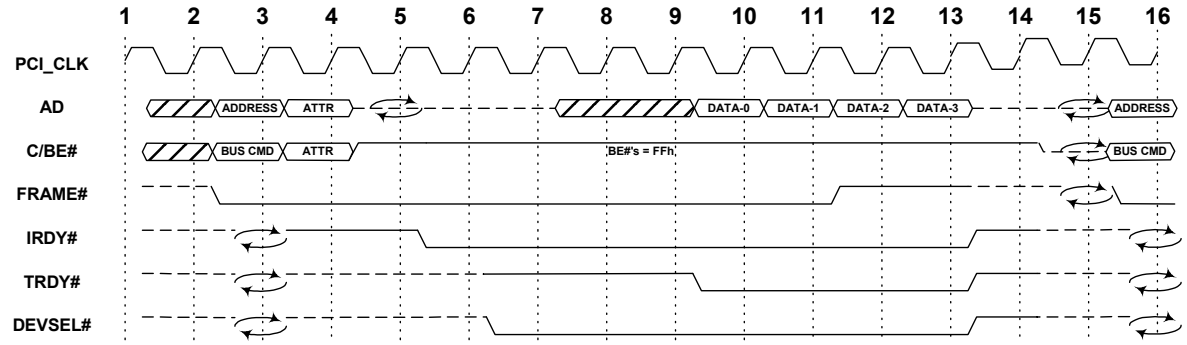

Figure 2-61: Burst Read Transaction with DEVSEL# Decode C and Two Initial Wait States, Mode 2

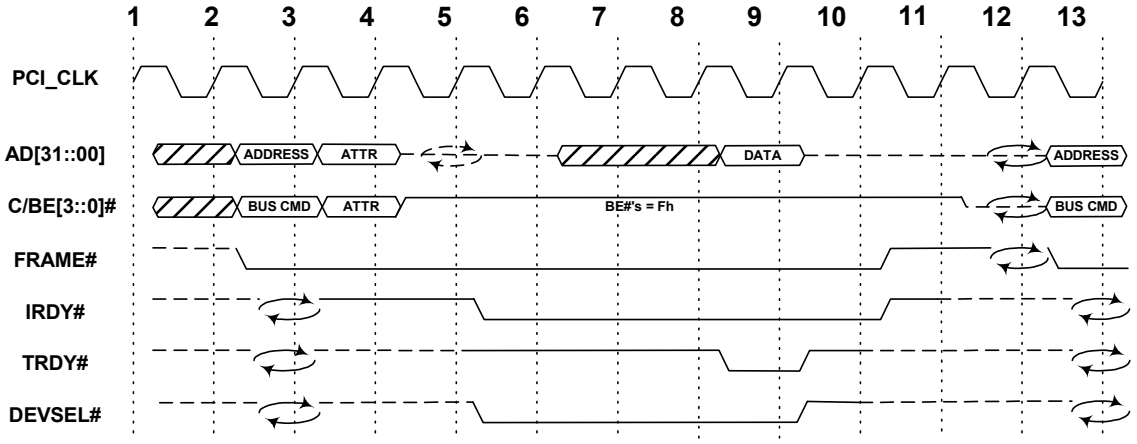

Figure 2-62: DWORD Read Transaction with DEVSEL# Decode B and Two Initial Wait States, Mode 2

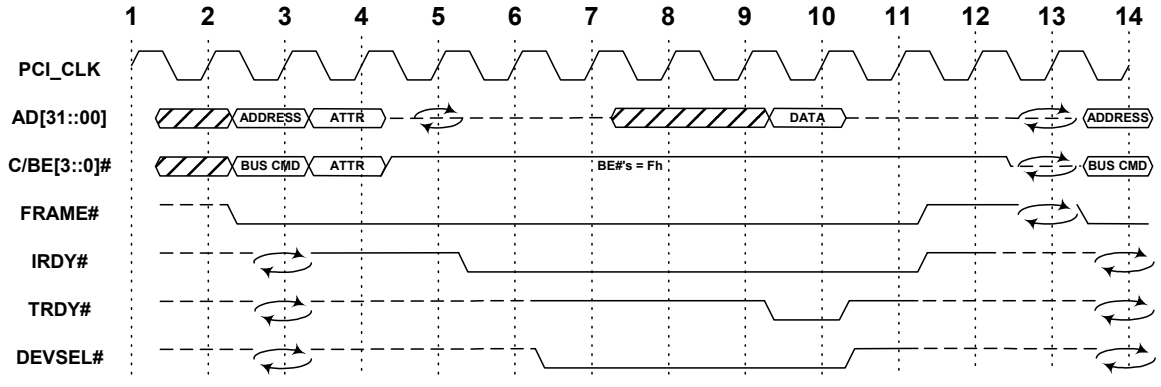

Figure 2-63: DWORD Read Transaction with DEVSEL# Decode C and Two Initial Wait States, Mode 2

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. A Split Completion is a burst transaction, even if the Split Request was a DWORD transaction. As for all burst transactions, Split Completions include the byte count in the attribute phase. In PCI-X Mode 1, the C/BE# bus is reserved and driven high during all data phases of a Split Completion. In PCI-X Mode 2, Split Completion transactions are source-synchronous regardless of the byte count of the transaction, and even if the Split Request was a DWORD transaction. In PCI-X Mode 2, the C/BE# bus carries the data strobes for each subphase of a Split Completion (see Section 2.1.3.5.1, "Source-Synchronous Data Strobes," in PCI-X EM 2.0).

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 (16 bits of the AD bus during each of the two data phases if the bus is 16 bits wide, see Table 2-22). In PCI-X Mode 1, 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). Since the Lower Address is set to zero regardless of the address of the Split Request, in PCI-X Mode 2, data is valid only in the first subphase (two subphases on a 16-bit bus) and only in the byte lanes corresponding to the enabled bytes in the Split Request. The requester must ignore the data in the other byte lanes and subphases (except for ECC checking, see Section 5.1.2). If the Split Request was a write transaction, a Split Completion Message is driven on AD[31::00] (16 bits of the AD bus during each of the two data phases if the bus is 16 bits wide, see Table 2-22) 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 on 64-bit buses, 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 2.12.1.3.) 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 2-64 shows the bit assignments for the Split Completion address. The Split Completion command is driven on C/BE[3::0]# (two bits of the C/BE# bus during each of the two data phases if the bus is 16 bits wide, see Table 2-22). Table 2-13 describes the bit definitions of the Split Completion address fields.

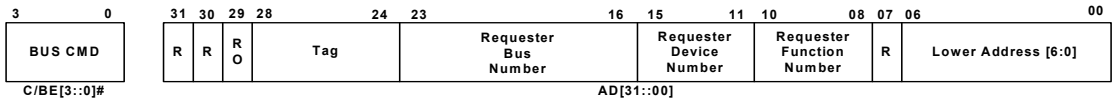

Figure 2-64: Split Completion Address

Table 2-13: Split Completion Address Field Definitions

Field	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.
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 secondary interface is operating in PCI-X 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.

Field	Function
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 2-65 shows the bit assignments for the Completer Attributes, and Table 2-14 describes the bit definitions.

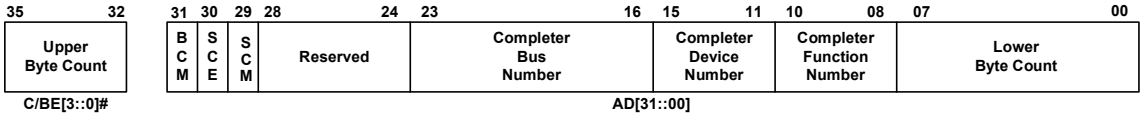

Figure 2-65: Completer Attribute Bit Assignments

Table 2-14: 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)
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.

Attribute	Function
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.
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 5.2.5 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 an uncorrectable error in the Split Completion address, which includes the Sequence ID. (See Section 5.2.3 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 5.2.4 and 8.7.1.3.) 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. In PCI-X Mode 1, 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. Note that in PCI-X Mode 2, the C/BE# bus carries data strobes for each subphase of the Split Completion (see Section 2.1.3.5.1, "Source-Synchronous Data Strobes," in PCI-X EM 2.0), and the Split Completion Message appears in the first subphase, as would be true for any Split Completion in which the Lower Address bits were 0 and the byte count was four.

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 2-66 shows the format of the message in the data phase of the Split Completion, and Table 2-15 shows the encoding of those messages.

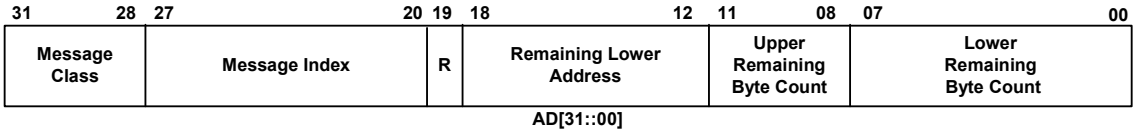

Figure 2-66: Split Completion Message Format

Table 2-15: 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 2-16 and Table 2-17.
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 2-16. All other indices are reserved.

Table 2-16: 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 Sequence.

Table 2-17 shows the index values defined for this message class. All other indices are reserved.

Table 2-17: 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	Uncorrectable Split Write Data Error. The completer sends this message if it terminated a DWORD write transaction with Split Response and detected an uncorrectable data error. (See Section 5.2.4.)
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

Transaction termination rules are based on number of data phases and are independent of the width of the transaction (64, 32, or 16 bits). That is, the amount of data transferred in each data phase varies, but the rules based on number of data phases are the same.

Most of the figures in this section that illustrate transaction termination show DEVSEL# decode A and no target initial wait states. Decode speed A is not allowed in ECC mode. All other DEVSEL# decodes and wait state combinations specified in Sections 2.8 and 2.9.1 are allowed.

In PCI-X Mode 1, when a transaction ends, the initiator deasserts and floats FRAME# as described in PCI 2.3 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 2-69 and Figure 2-70). 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 2-68 illustrates the other alternative.

In PCI-X Mode 2, the assertion, deassertion, and float requirements for control signals related to transaction termination are the same as in PCI-X Mode 1 except that there are a minimum of two idle clocks (also described as one idle clock and a bus turn-around clock) between two transactions. (In some cases in Mode 1, the next transaction begins after only one idle clock. See Section 4.2.1.)

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. In PCI-X Mode 1, 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

All of the figures in this section illustrate transaction termination using PCI-X Mode 1. Transaction termination in PCI-X Mode 2 would be the same, except that in PCI-X Mode 2, the initiator and target drive and float the AD and C/BE# buses as described in Sections 2.16 and 4.1.2.

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.

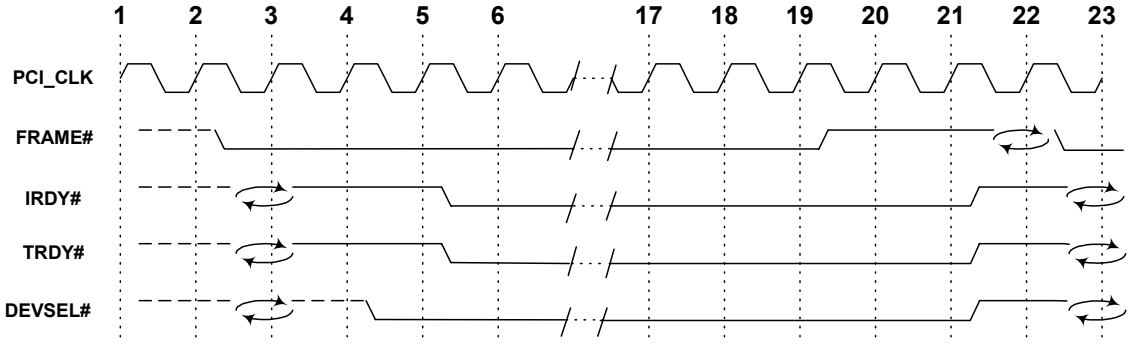

Figure 2-67: Initiator Termination of a Burst Transaction with Four or More Data Phases

Figure 2-67 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.

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 2-18 shows the number of data phases for PCI-X Mode 1 and common-clock PCI-X Mode 2 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. Data always transfers in an even number of data phases for common-clock Mode 2 transactions on a 16-bit bus.

Table 2-18: Data Phases Dependence on Starting Address and Bus Width, Mode 1 and Common-Clock Mode 2

	Data Phases			
	64-bit Transfers		32-bit Transfers	16-Bit Transfers
Transaction Length (DWORDs)	Starting on Even DWORD	Starting on Odd DWORD		
1	1	1	1	2
2	1	2	2	4
3	2	2	3	6
4	2	3	4	8
5	3	3	5	10
6	3	4	6	12
7	4	4	7	14
8	4	5	8	16
9	5	5	9	18
10	5	6	10	20
11	6	6	11	22
12	6	7	12	24

Table 2-19 and Table 2-20 show the number of data phases for PCI-X Mode 2 source-synchronous transactions up to 25 DWORDs long. (As in the previous table, the length of the transaction is measured from the starting address rounded down to the next DWORD address.) The headings under each transfer width indicate the DWORD address that contains the starting address. The DWORD address corresponds with AD[2], AD[3:2], or AD[4:2], as indicated in the tables.

Table 2-19: Data Phases Dependence on Starting Address and Bus Width, Source-Synchronous PCI-X 266

Transaction Length (DWORDs)	Data Phases							
	64-Bit Transfer AD[3::2]				32-Bit Transfer AD[2]		16-Bit Transfers AD[2]	
	0	1	2	3	0	1	0	1
1	1	1	1	1	1	1	1	1
2	1	1	1	2	1	2	2	2
3	1	1	2	2	2	2	3	3
4	1	2	2	2	2	3	4	4
5	2	2	2	2	3	3	5	5
6	2	2	2	3	3	4	6	6
7	2	2	3	3	4	4	7	7
8	2	3	3	3	4	5	8	8
9	3	3	3	3	5	5	9	9
10	3	3	3	4	5	6	10	10
11	3	3	4	4	6	6	11	11
12	3	4	4	4	6	7	12	12
13	4	4	4	4	7	7	13	13
14	4	4	4	5	7	8	14	14
15	4	4	5	5	8	8	15	15
16	4	5	5	5	8	9	16	16
17	5	5	5	5	9	9	17	17
18	5	5	5	6	9	10	18	18
19	5	5	6	6	10	10	19	19
20	5	6	6	6	10	11	20	20
21	6	6	6	6	11	11	21	21
22	6	6	6	7	11	12	22	22
23	6	6	7	7	12	12	23	23
24	6	7	7	7	12	13	24	24
25	7	7	7	7	13	13	25	25

Table 2-20: Data Phases Dependence on Starting Address and Bus Width, Source-Synchronous PCI-X 533

Transaction Length (DWORDs)	Data Phases															
	64-Bit Transfer AD[4::2]								32-Bit Transfer AD[3::2]				16-Bit Transfer AD[3::2]			
	0	1	2	3	4	5	6	7	0	1	2	3	0	1	2	3
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	2	1	1	1	2	1	2	1	2
3	1	1	1	1	1	1	2	2	1	1	2	2	2	2	2	2
4	1	1	1	1	1	2	2	2	1	2	2	2	2	3	2	3
5	1	1	1	1	2	2	2	2	2	2	2	2	3	3	3	3
6	1	1	1	2	2	2	2	2	2	2	2	3	3	4	3	4
7	1	1	2	2	2	2	2	2	2	2	3	3	4	4	4	4
8	1	2	2	2	2	2	2	2	2	3	3	3	4	5	4	5
9	2	2	2	2	2	2	2	2	3	3	3	3	5	5	5	5
10	2	2	2	2	2	2	2	3	3	3	3	4	5	6	5	6
11	2	2	2	2	2	2	3	3	3	3	4	4	6	6	6	6
12	2	2	2	2	2	3	3	3	3	4	4	4	6	7	6	7
13	2	2	2	2	3	3	3	3	4	4	4	4	7	7	7	7
14	2	2	2	3	3	3	3	3	4	4	4	5	7	8	7	8
15	2	2	3	3	3	3	3	3	4	4	5	5	8	8	8	8
16	2	3	3	3	3	3	3	3	4	5	5	5	8	9	8	9
17	3	3	3	3	3	3	3	3	5	5	5	5	9	9	9	9
18	3	3	3	3	3	3	3	4	5	5	5	6	9	10	9	10
19	3	3	3	3	3	3	4	4	5	5	6	6	10	10	10	10
20	3	3	3	3	3	4	4	4	5	6	6	6	10	11	10	11
21	3	3	3	3	4	4	4	4	6	6	6	6	11	11	11	11
22	3	3	3	4	4	4	4	4	6	6	6	7	11	12	11	12
23	3	3	4	4	4	4	4	4	6	6	7	7	12	12	12	12
24	3	4	4	4	4	4	4	4	6	7	7	7	12	13	12	13
25	4	4	4	4	4	4	4	4	7	7	7	7	13	13	13	13

Figure 2-68 through Figure 2-70 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. Note that the three- and one-data phase cases described are not possible in 16-bit common-clock transactions because 16-bit common-clock transactions that transfer data always have an even number of data phases.

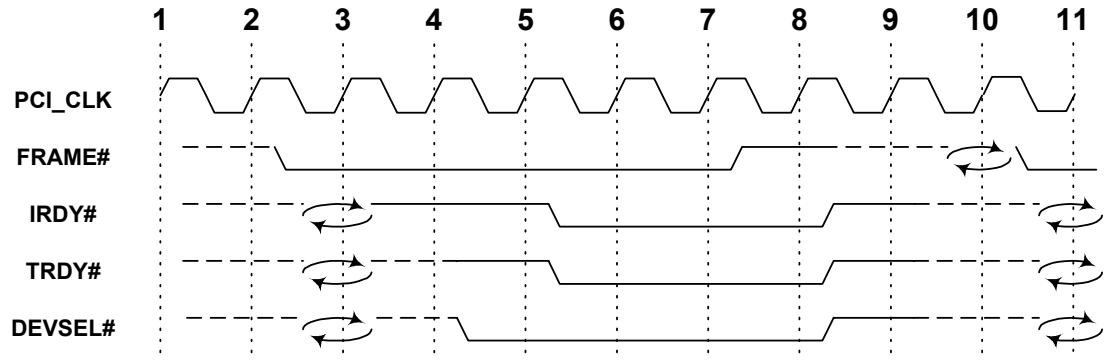

Figure 2-68: Initiator Termination of a Burst Transaction with Three Data Phases

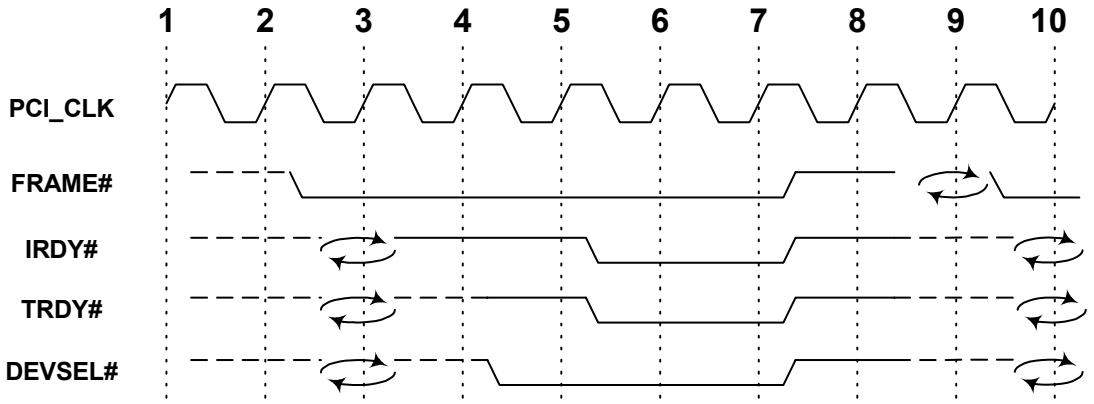

Figure 2-69: Initiator Termination of a Burst Transaction with Two Data Phases

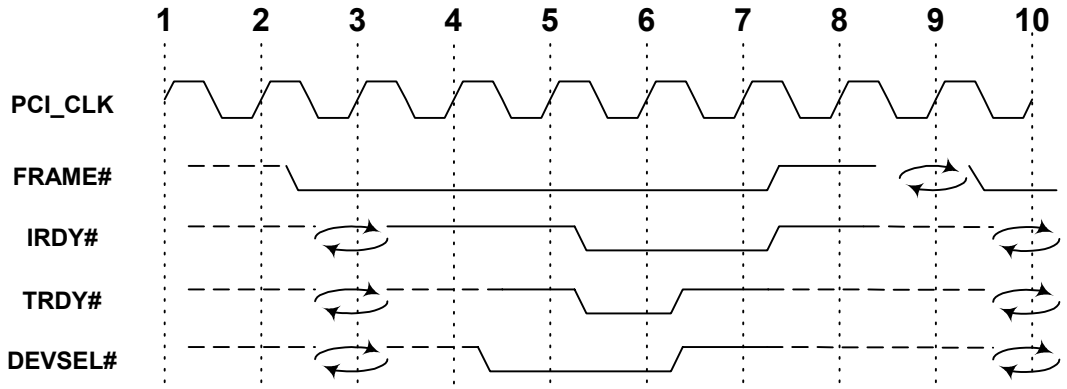

Figure 2-70: Initiator Termination of a Burst Transaction with One Data Phase

2.11.1.2. Master-Abort Termination

If no target asserts DEVSEL# within five clocks after the attribute phase (the second attribute phase in 16-bit transactions), the initiator deasserts FRAME# and IRDY# seven

clocks after the attribute phase(s). In Mode 1, the initiator floats the bus one clock later. In Mode 2, the initiator continues to drive the bus until a turn-around cycle (see Section 4.1.2). The initiator sets bits in its Status register the same as for conventional PCI.

Figure 2-71 shows a Master-Abort termination of a transaction.

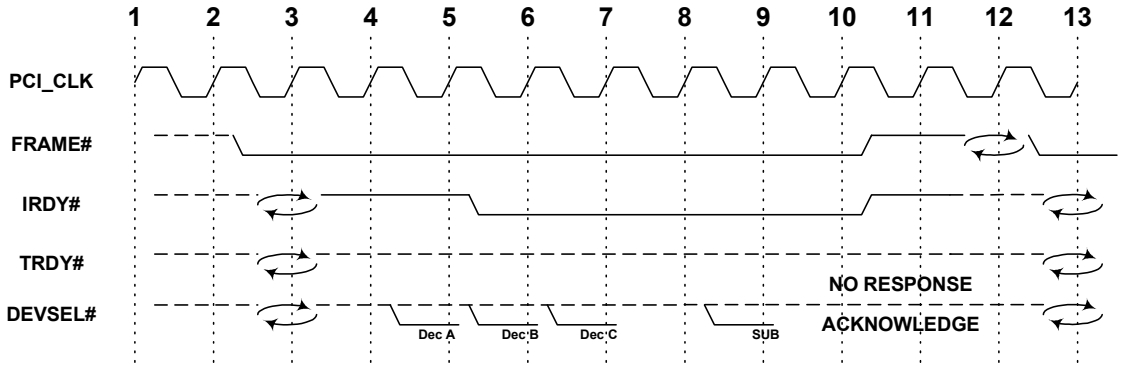

Figure 2-71: Master-Abort Termination

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 2-21 shows all of the alternatives for target data phase signaling.

Table 2-21: Target Data Phase Signaling

Target Data Phase Signaling	DEVSEL#	STOP#	TRDY#	Data Transfer	Transaction Terminates or Continues
Master-Abort (Note 1)	Deassert	Deassert	Deassert	na	na
Split Response	Deassert	Deassert	Assert	(Note 2)	Terminates
Target-Abort	Deassert	Assert	Deassert	No	Terminates
Single Data Phase Disconnect	Deassert	Assert	Assert	Yes	Terminates
Wait State	Assert	Deassert	Deassert	No	Continues
Data Transfer	Assert	Deassert	Assert	Yes	Continues
Retry	Assert	Assert	Deassert	No	Terminates
Disconnect at Next ADB	Assert	Assert	Assert	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 Sequence 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. For DWORD transactions and common-clock burst transactions on a 16-bit bus, the target signals data phase action in pairs of clocks if it signals Data Transfer, Single Data Phase Disconnect, or Split Response. (16-bit common-clock transactions that transfer data always have an even number of data phases. See Section 2.12.2.3.)

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.

All of the figures in this section illustrate transaction termination using PCI-X Mode 1. Transaction termination in PCI-X Mode 2 would be the same, except that in PCI-X Mode 2, the initiator and target drive and float the AD and C/BE# buses as described in Sections 2.16 and 4.1.2, and for DWORD and common-clock burst transactions on a 16-bit bus the target signals Data Transfer, Single Data Phase Disconnect, or Split Response in pairs of clocks (see Section 2.12.2.3).

2.11.2.1. Single Data Phase Disconnection

The target signals its intention to complete a single data phase (two data phases on a 16-bit common-clock transaction) 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 other than Device ID Message transaction (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 (two data phases on a 16-bit common-clock transaction), and the initiator deasserts FRAME# and IRDY# two clocks after the data phase (or first data phase on a 16-bit bus, see Section 2.12.2.3.3). In PCI-X Mode 2, if the transaction is a source-synchronous transaction, it contains a single data phase with all its subphases, but only the data of the subphase that includes the starting address is transferred. The other subphases are ignored, except for ECC checking (see Section 5.1.2).

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 byte 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.

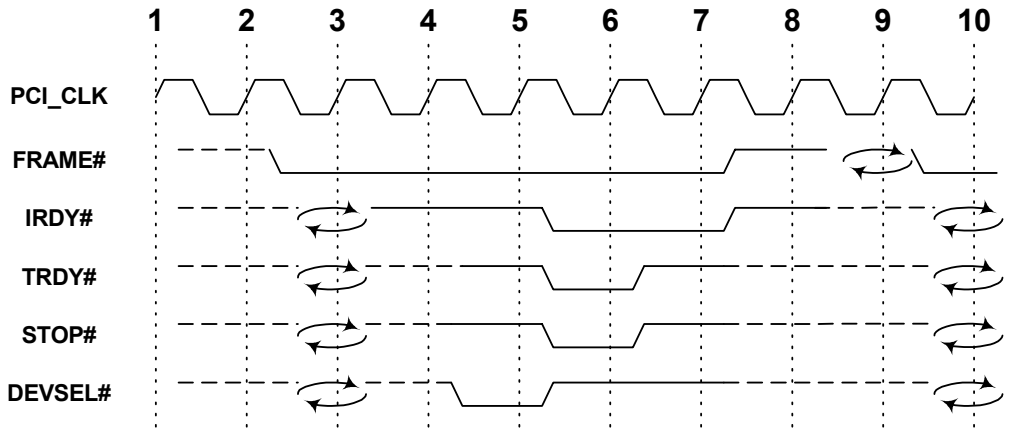

Figure 2-72: Single Data Phase Disconnection

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), it 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. In these illustrations, the number of clocks between ADBs is calculated for a common-clock 64-bit burst transfer. The number of clocks is different for other combinations of transfer widths and modes. (See Table 2-1.) Figure 2-73 illustrates Disconnect at Next ADB after the first data phase and four data phases from an ADB.

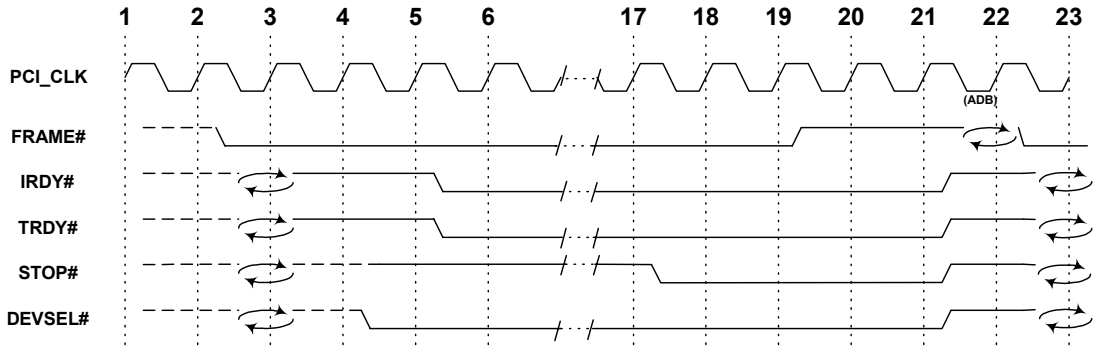

Figure 2-73: Disconnect at Next ADB Four Data Phases from an ADB

Figure 2-74 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 Figure 2-74), the transaction disconnects on the ADB.

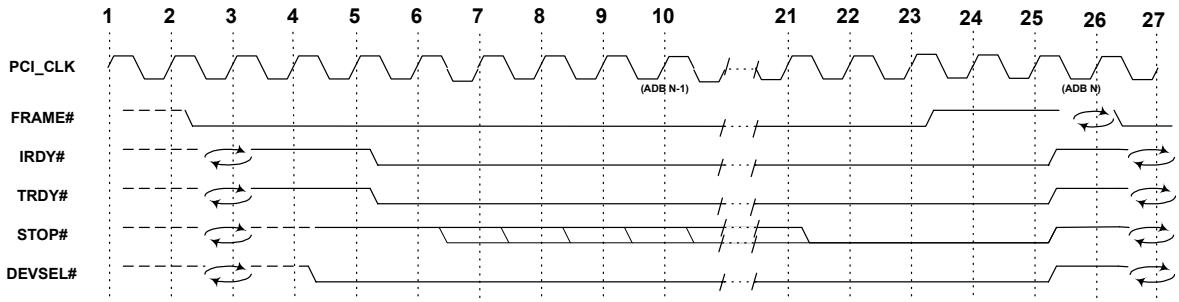

Figure 2-74: Disconnect at Next ADB on ADB N

Figure 2-75 through Figure 2-77 illustrate the target signaling Disconnect at Next ADB for transactions whose starting address is three, two, and one data phases from the ADB, respectively. (Note that the three- and one-data phase cases are not possible for 16-bit common-clock transactions because 16-bit common-clock transactions always transfer data in an even number of data phases.) 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).

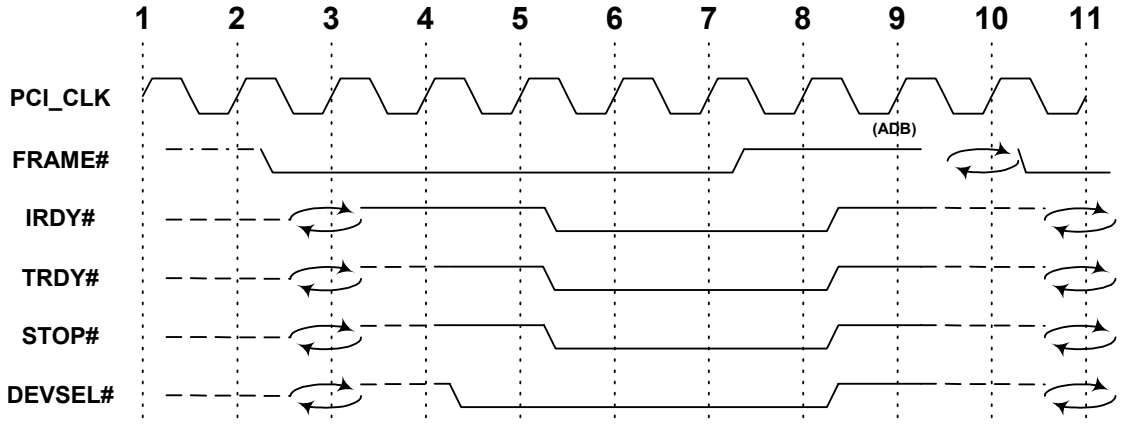

Figure 2-75: Disconnect at Next ADB with Starting Address Three Data Phases from an ADB

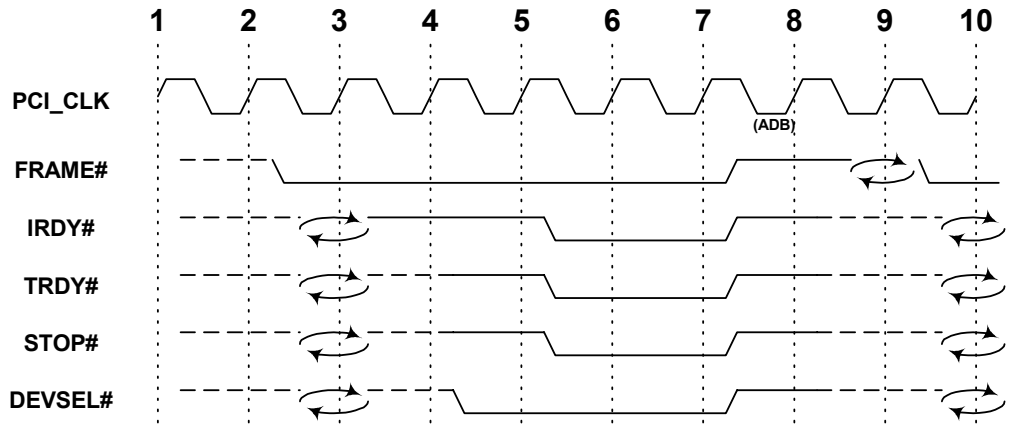

Figure 2-76: Disconnect at Next ADB with Starting Address Two Data Phases from an ADB

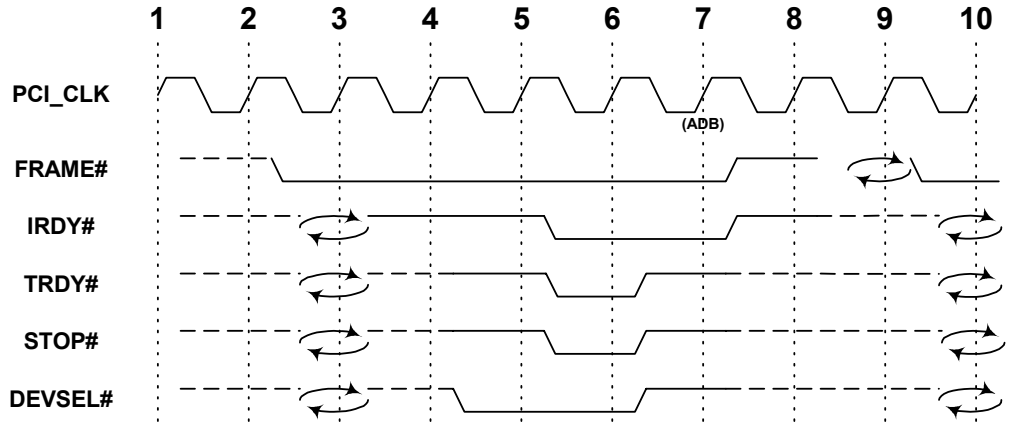

Figure 2-77: Disconnect at Next ADB with Starting Address One Data Phase from an ADB

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 2-7, "3.3V General Timing Parameters," in PCI-X EM 2.0) has not elapsed.
- The device normally transfers data within the target initial latency limit listed in Table 2-12, 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 2.3 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.)

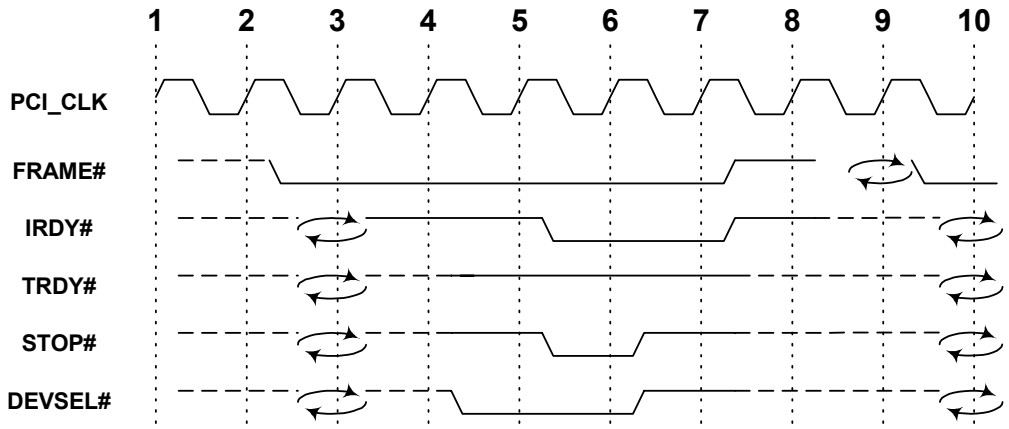

Figure 2-78: Retry Termination

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). (See Section 2.12.2.3.3 for the requirement for the target to signal Split Response for two clocks in 16-bit common-clock cases.) The target it permitted to signal Split Response on any DWORD transaction (except Special Cycle) and any read transaction. The target drives all bits of the AD bus high during the clock in which it signals Split Response for a read transaction.

Figure 2-79 shows Split Response for a read transaction (either burst or DWORD) in PCI-X Mode 1. Figure 2-80 shows Split Response for a DWORD write transaction in PCI-X Mode 1. Split Response in PCI-X Mode 2 would be identical, except the bus and control

signal driving and floating requirements are the same as for DWORD read and write transactions specified in Section 2.7.1.

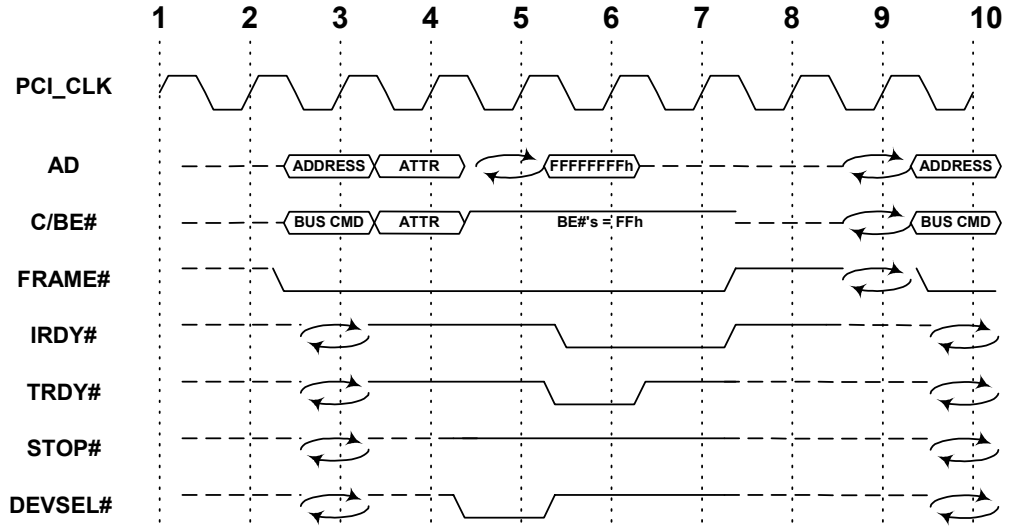

Figure 2-79: Split Response Termination for a Read Transaction, Mode 1

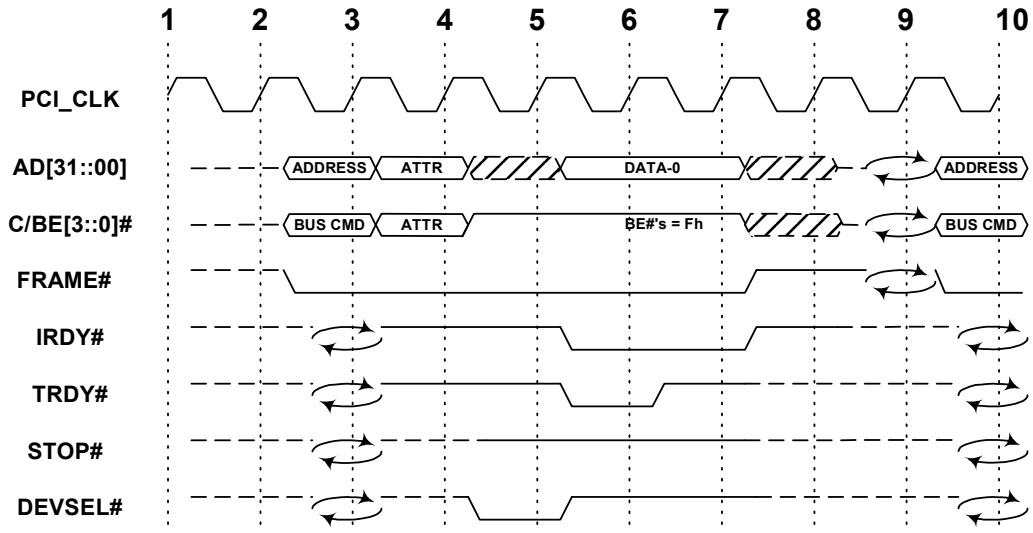

Figure 2-80: Split Response Termination for a DWORD Write Transaction, Mode 1

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 2-81 illustrates a Target-Abort in the first data phase of a transaction. Figure 2-82 illustrates a Target-Abort after the target has signaled Data Transfer for several data phases of a burst transaction.

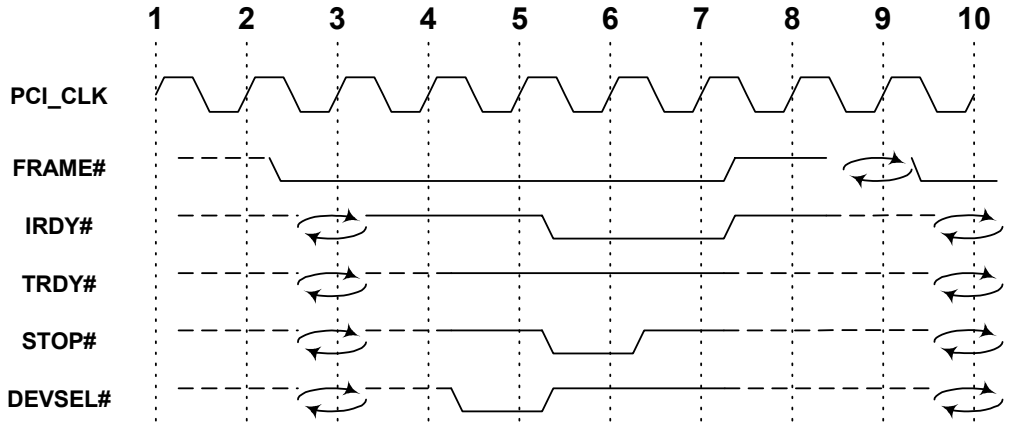

Figure 2-81: Target-Abort on First Data Phase

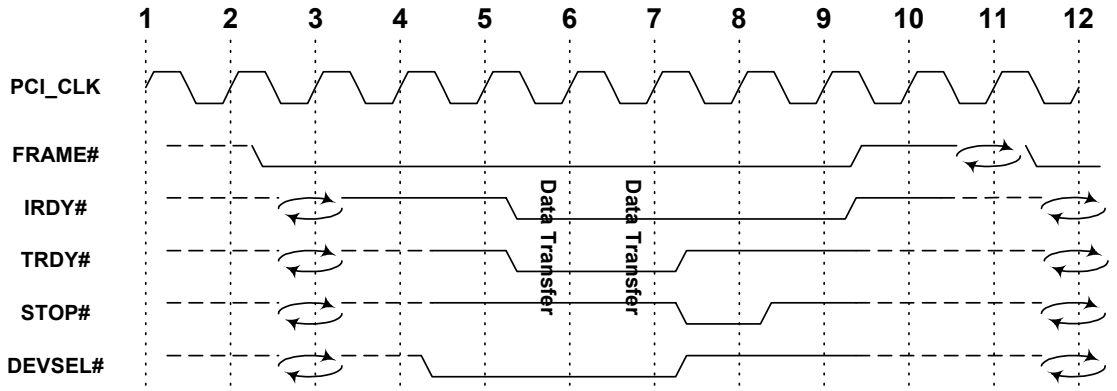

Figure 2-82: Target-Abort after Data Transfer

2.12. Bus Width

2.12.1. 64-Bit and 32-Bit Bus Width

PCI-X devices designed to connect directly to an add-in card connector must support a 32-bit interface. They optionally support a 64-bit interface.

As in conventional PCI, 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 2.12.1.1). (See Section 2.16.1 for dual address cycles on device ID message transactions.) 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 in a manner similar to conventional PCI (see Section 2.12.1.3).

In ECC mode, the upper bus (AD[63::32], C/BE[7::4]#, and ECC[7]) of 64-bit devices is used only for data phases. These bits are not used during the address and attribute phases, as described below. (This allows a seven-bit ECC to be used exclusively on the lower bus before the width of the transaction is negotiated. It also allows the target in PCI-X Mode 2 to leave its receivers for the upper bus disabled until the width of the transaction is negotiated. In PCI-X Mode 2, a device is not permitted to enable a receiver for a Category 1 signal, as defined in Table 2-1, "Driver and Receiver Categories," in PCI-X EM 2.0, except when it is required to be driven by another device.) Furthermore, REQ64# and ACK64# signal pins are shared with ECC bits, which makes the negotiation of transfer width slightly different from operation in parity mode (see Section 2.12.1.3).

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 2.3.

At various places throughout the discussion of bus widths in PCI-X Mode 1, a bus or a portion of a bus is described as "reserved." Unless otherwise noted, the state of a "reserved" bus is not specified (either drive or float) and is ignored by the device receiving

the bus. Discussion of buses in PCI-X Mode 2 describes specific bus requirements and do not use the term "reserved."

2.12.1.1. 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 Mode 1 in parity mode are the same as for conventional PCI. See Section 2.16.1 for dual address cycles on device ID message transactions.

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 1 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.

In ECC mode, the upper bus of 64-bit devices (AD[63::32], C/BE[7::4]#, and ECC[7]) is not used during the address phase or phases. The upper bus is optionally driven by the initiator, and the receivers in any PCI-X Mode 2 64-bit targets are disabled. (A Mode 2 device generally dissipates less power when its outputs are enabled than it does when its input terminators are enabled. See Section 2.1.3.7, "1.5V Environment Signal Electrical Termination," in PCI-X EM 2.0.) The requirements for the lower bus (AD[31::00], C/BE[3::0]#, and ECC[6::0]) are the same as described above for parity mode. Note that this means it is not possible for a 64-bit device to decode a 64-bit address any earlier than a 32-bit device when operating in ECC mode. See Section 2.16.1 for dual address cycles on device ID message transactions.

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 2.3. (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 not tolerate 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 PCI Bridge 1.1.

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

Figure 2-83 illustrates a 64-bit initiator in PCI-X Mode 1 in parity mode 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).

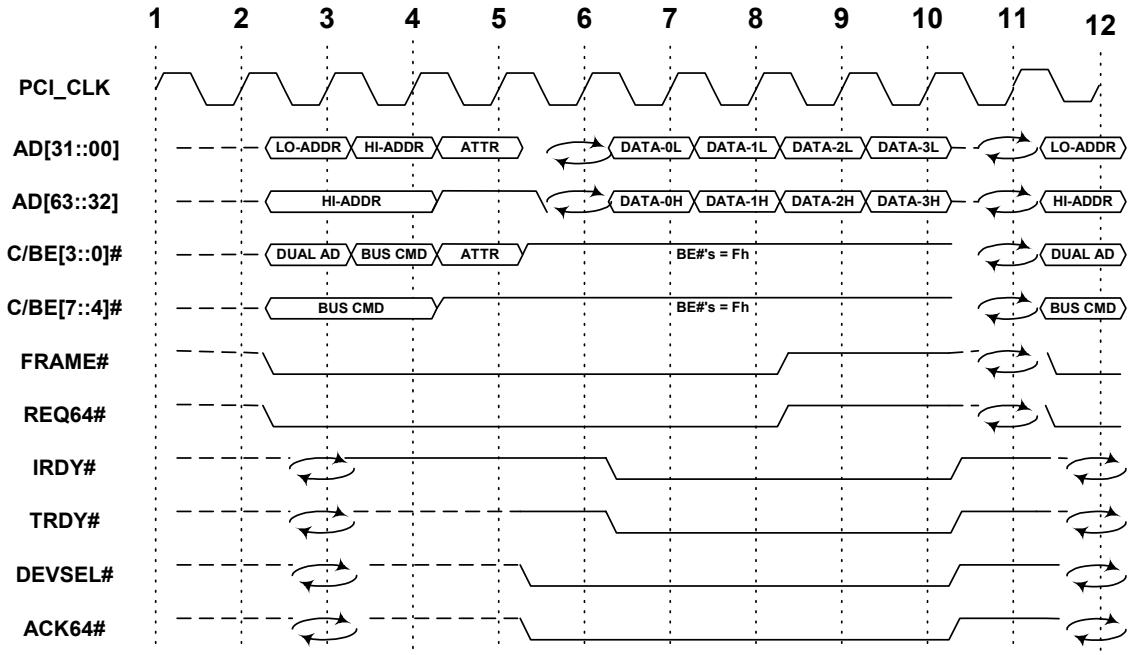

Figure 2-83: Dual Address Cycle 64-bit Memory Read Burst Transaction, Parity-Protected Mode 1

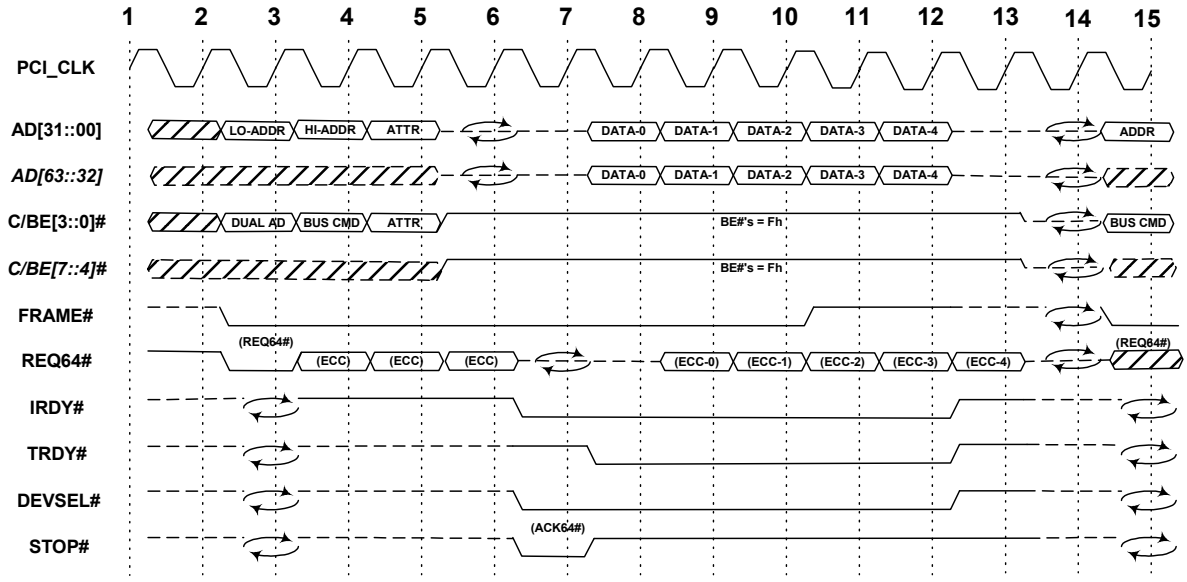

Figure 2-84: Dual Address Cycle 64-bit Memory Read Burst Transaction, Mode 2

Figure 2-84 illustrates a 64-bit initiator in PCI-X Mode 2 (with ECC support) 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 lower AD and C/BE# buses during the address and attribute phases the same as in Mode 1 (except ECC replaces parity). It optionally drives the upper buses, but all the receivers are disabled. The figure shows a 64-bit target responding with device select

timing B by asserting DEVSEL# in clock 7. (See Section 2.12.1.3 for the description of how the initiator and target indicate their widths in ECC mode.)

2.12.1.2. Attribute Width

Attributes are always driven in a single attribute phase both for 64-bit and 32-bit initiators. In parity mode, AD[63::32] and C/BE[7::4]# are driven high during the attribute phase of transactions from a 64-bit initiator. In ECC mode, AD[63::32] and C/BE[7::4]# are optionally driven during the attribute phase of transactions from a 64-bit initiator. (In PCI-X Mode 2, the device generally dissipates less power when its outputs are enabled than it does when its input terminators are enabled. See Section 2.1.3.7, "1.5V Environment Signal Electrical Termination," in PCI-X EM 2.0.)

2.12.1.3. Data Transfer Width

PCI-X support for varying the width of 64- and 32-bit 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. 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 parity-mode burst push transaction, the initiator is permitted to do the same with the AD[63::32] bus and PAR64. If the transaction is an ECC-mode burst push transaction, the target inserts a minimum of two wait states to allow the initiator to downshift to a 32-bit width. In this case, the upper buses (AD[63::32], C/BE[7::0]#, and ECC[7]) are not used in the transaction after the wait states, and are optionally driven by the initiator. (A device operating in Mode 2 generally dissipates less power when its outputs are enabled than it does when its input terminators are enabled. If the outputs are driven, they are recommended to be driven to a single value to minimize electrical noise and power dissipation caused by switching transients.)

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

1. A 64-bit initiator asserts REQ64# with the same timing as FRAME# to request a 64-bit data transfer. In parity mode, it deasserts REQ64# with the same timing as FRAME# at the end of the transaction (the same as for conventional PCI). In ECC mode, REQ64# carries ECC after the first (or only) address phase (see Section 5.1.2.4), so the target must latch the state of REQ64# during the first (or only) address phase.
2. All PCI-X devices support a status bit indicating whether they are a 64- or 32-bit device. See Section 7.2.4.

3. Only burst transactions use 64-bit transfers. DWORD transactions use 32-bit transfers.
4. 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 compared to a 64-bit transfer. (See Table 2-1.)
5. 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.)
6. The following rules apply to memory write and Split Completion transactions from a 64-bit initiator operating in parity mode. (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 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).

   The following rules apply to burst push transactions from a 64-bit initiator in ECC mode.

   d. Data before the starting address is not specified other than that it must be included in the ECC calculation and be driven with the ECC check bits, even if AD[2] of the starting byte address is 1. (Unlike parity mode, there is no requirement to copy down data from the upper bus.)
   e. Unlike parity mode, target width in ECC mode is encoded on STOP# and DEVSEL#, and the ACK64# pin is used to carry ECC. If the target asserts STOP# 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 data phase values (including all subphases) on AD[63::00]. Data prior to the starting address is unspecified other than that it must be included in the ECC calculation and be driven with the ECC check bits, each time the first data phase value is repeated. (This means the unspecified data is not required to repeat the same value.)
   f. If the target does not assert STOP# when it asserts DEVSEL# (indicating it is responding as a 32-bit target), the target inserts a minimum of two wait states. The first time the initiator repeats the first two data phase values, it drives the data on AD[31::0] as a 32-bit initiator would, starting from a 32-bit data phase boundary.

Note that the 32-bit data phase boundary is not the same as the 64-bit data phase boundary, if AD[3] of the starting address is a 1 in PCI-X 266 mode or AD[4] is a 1 in PCI-X 533 mode. If the target continues to insert wait states, the initiator toggles between the first and second data phase values as a 32-bit initiator would.

IMPLEMENTATION NOTE

Deassertion of ACK64# for Single Data Phase Disconnect in PCI-X Mode 1

As in conventional PCI, the width of a parity-mode 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 2-85 through Figure 2-92 illustrate the cases in which a device initiates a 64-bit transaction and a 32-bit target responds, in parity-protected PCI-X Mode 1. Other burst push transactions behave the same as the Memory Write transactions shown 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.

In ECC-protected PCI-X Mode 1, decode speed A is not allowed, and REQ64# and ACK64# are combined with ECC bits, as shown in the Mode 2 examples in Figure 2-93 through Figure 2-95.

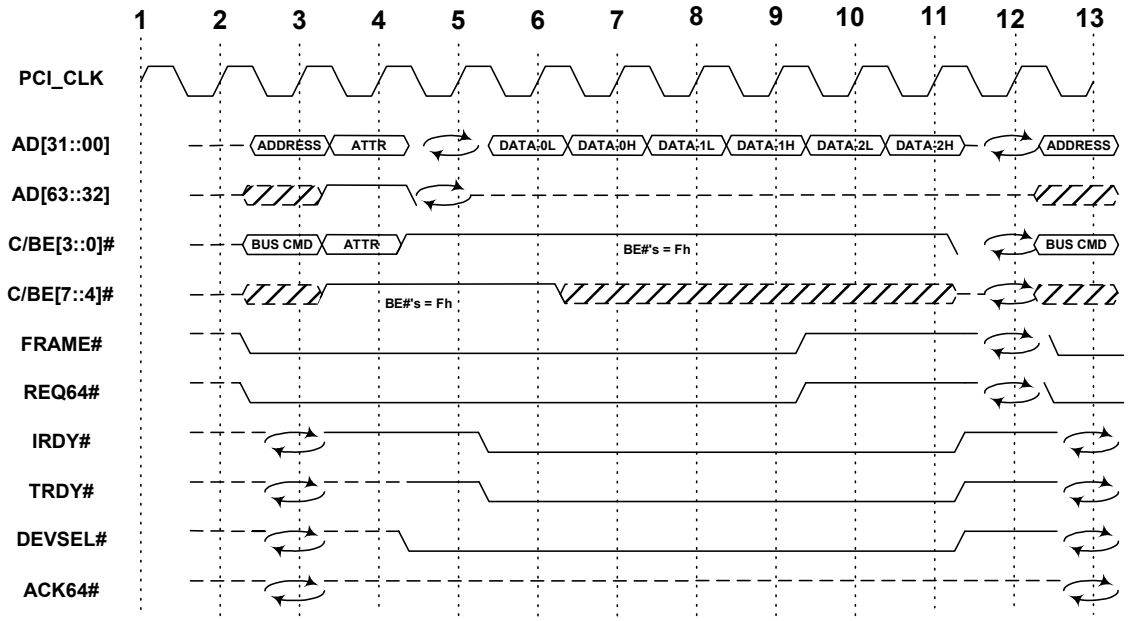

Figure 2-85: 64-bit Initiator Reading from 32-bit Target Starting on Even DWORD, Parity-Protected Mode 1

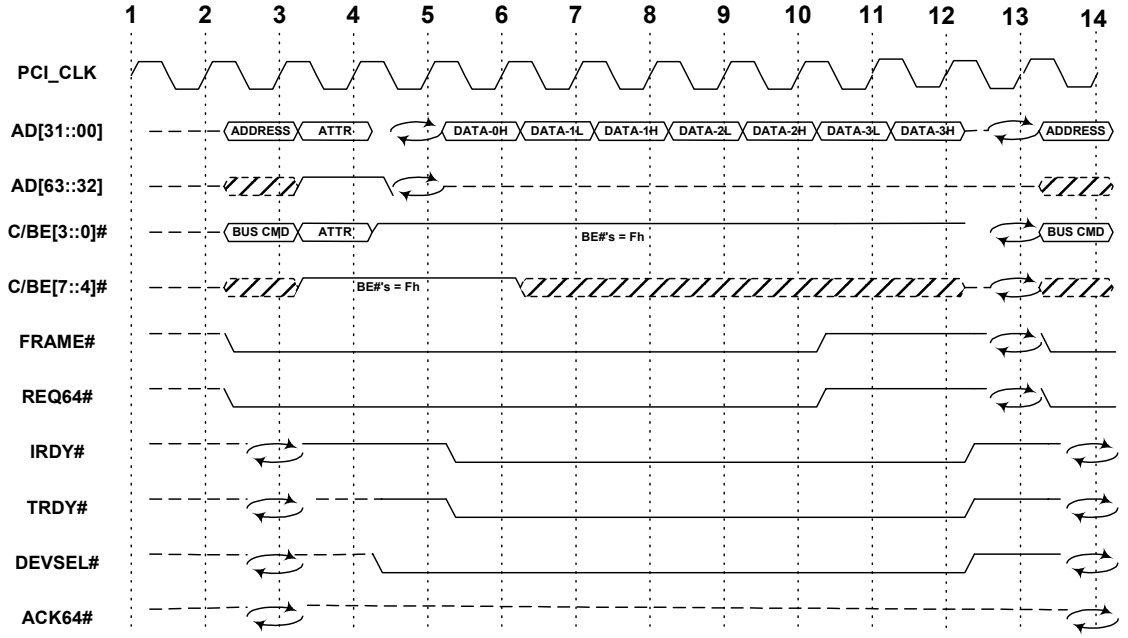

Figure 2-86: 64-bit Initiator Reading from 32-bit Target Starting on Odd DWORD, Parity-Protected Mode 1

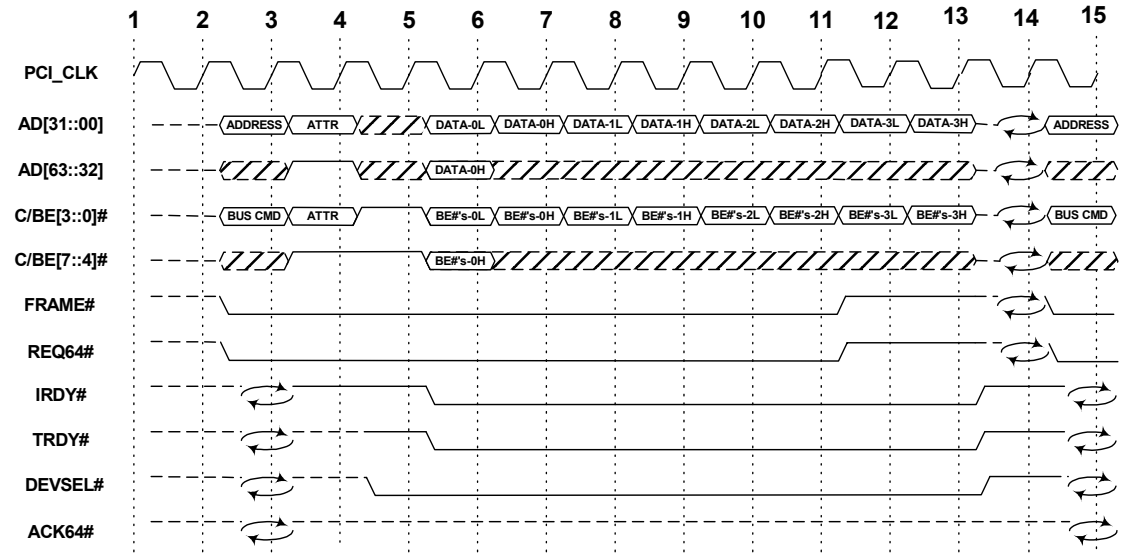

Figure 2-87: 64-bit Initiator Writing to 32-bit Target Starting on Even DWORD, Parity-Protected Mode 1

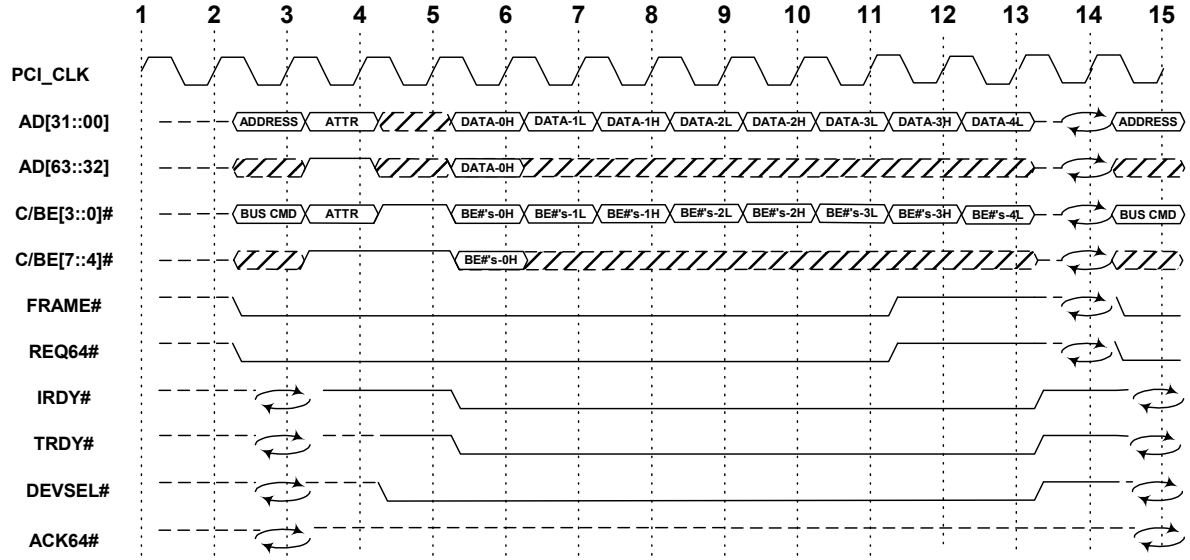

Figure 2-88: 64-bit Initiator Writing to 32-bit Target Starting on Odd DWORD, Parity-Protected Mode 1

Figure 2-88 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 2-88), 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 2-88 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.)

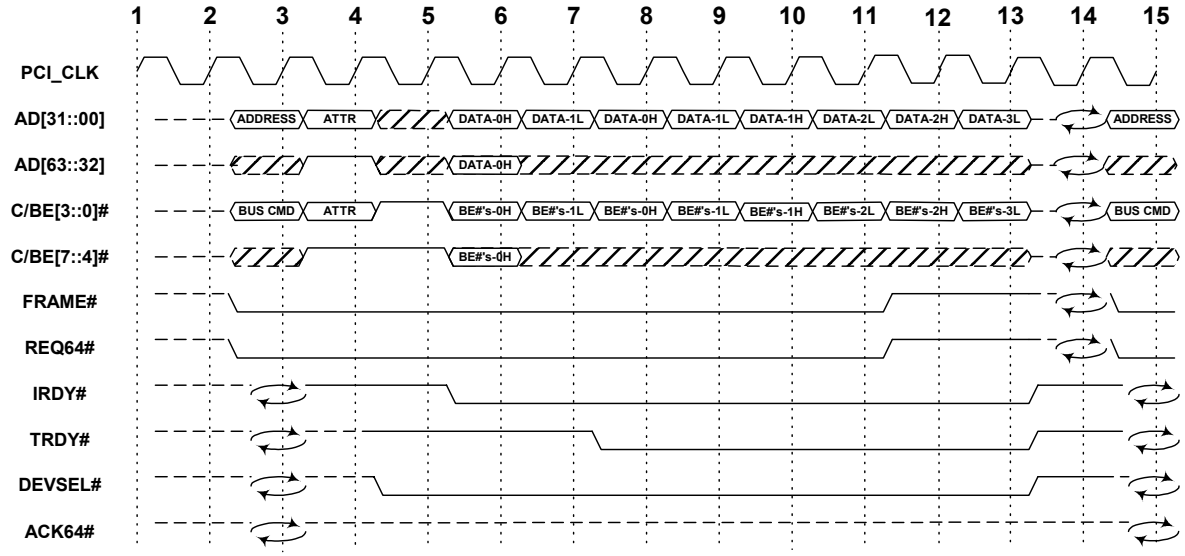

Figure 2-89: 64-bit Initiator Writing to 32-bit Target Starting on Odd DWORD, Decode A and Two Initial Wait States, Parity-Protected Mode 1

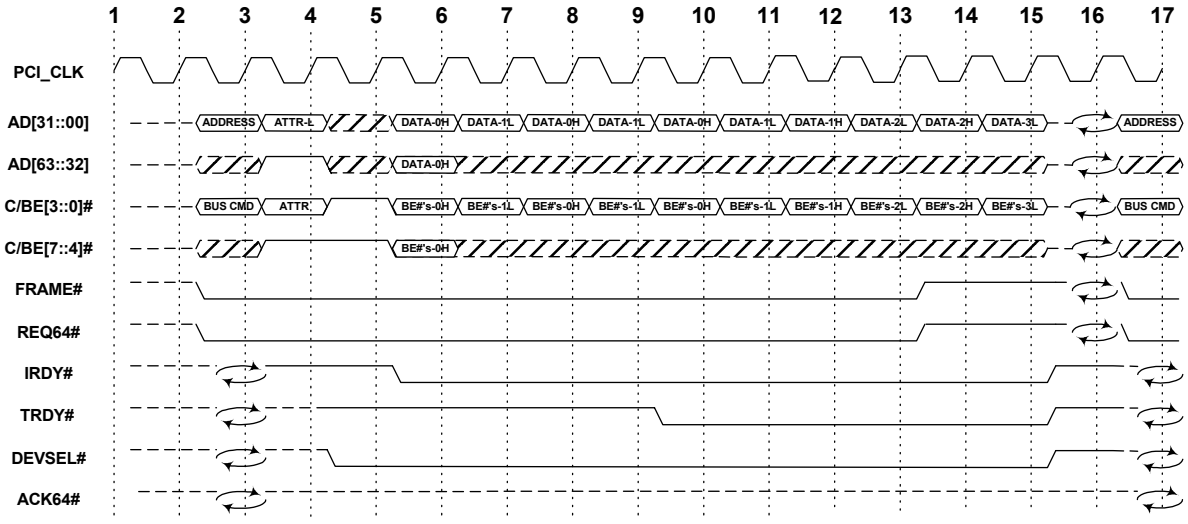

Figure 2-90: 64-bit Initiator Writing to 32-bit Target Starting on Odd DWORD, Decode A and Four Initial Wait States, Parity-Protected Mode 1

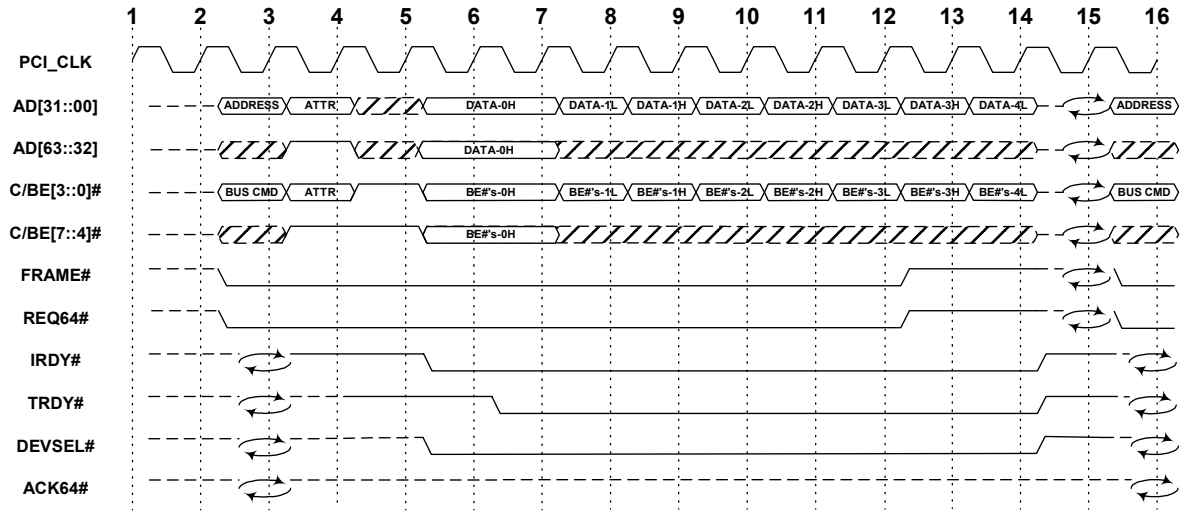

Figure 2-91: 64-bit Initiator Writing to 32-bit Target Starting on Odd DWORD, Decode B, Parity-Protected Mode 1

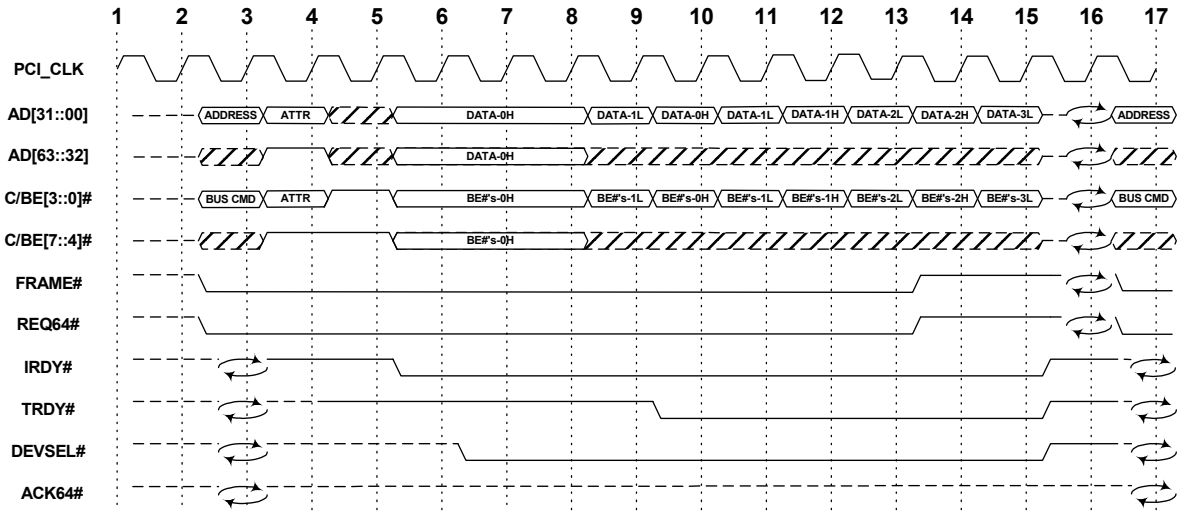

Figure 2-92: 64-bit Initiator Writing to 32-bit Target Starting on Odd DWORD, Decode C and Two Initial Wait States, Parity-Protected Mode 1

Figure 2-93 through Figure 2-95 illustrate the cases in which a device initiates a 64-bit transaction and a 32-bit target responds, in PCI-X Mode 2. All of these figures illustrate source-synchronous burst push transactions. Common-clock burst push transactions (Memory Write command) would have the same bus drive and float requirements and data phases would not include subphases. In each case in these figures, the initiator begins at the first 64-bit data phase boundary that is less than or equal to the starting address. As with all cases of decode speed B and C from a 64-bit initiator, the initiator repeats the first data phase value (with all its subphases) as a 64-bit initiator until the target asserts DEVSEL#. (The target does not declare its width until it asserts DEVSEL#.) When the target asserts DEVSEL#, the initiator drives the first two data phase values (including all the subphases) as

a 64-bit transfer. Since the target is required to insert a minimum of two wait states (see Section 2.9), no data is transferred in these two clocks. When the initiator repeats the first two data phase values, it uses only AD[31::00] and begins at the first 32-bit data phase boundary that is less than the starting address. The target is permitted to transfer the data or signal more wait states (in pairs) up to the limit specified in Section 2.9.1.

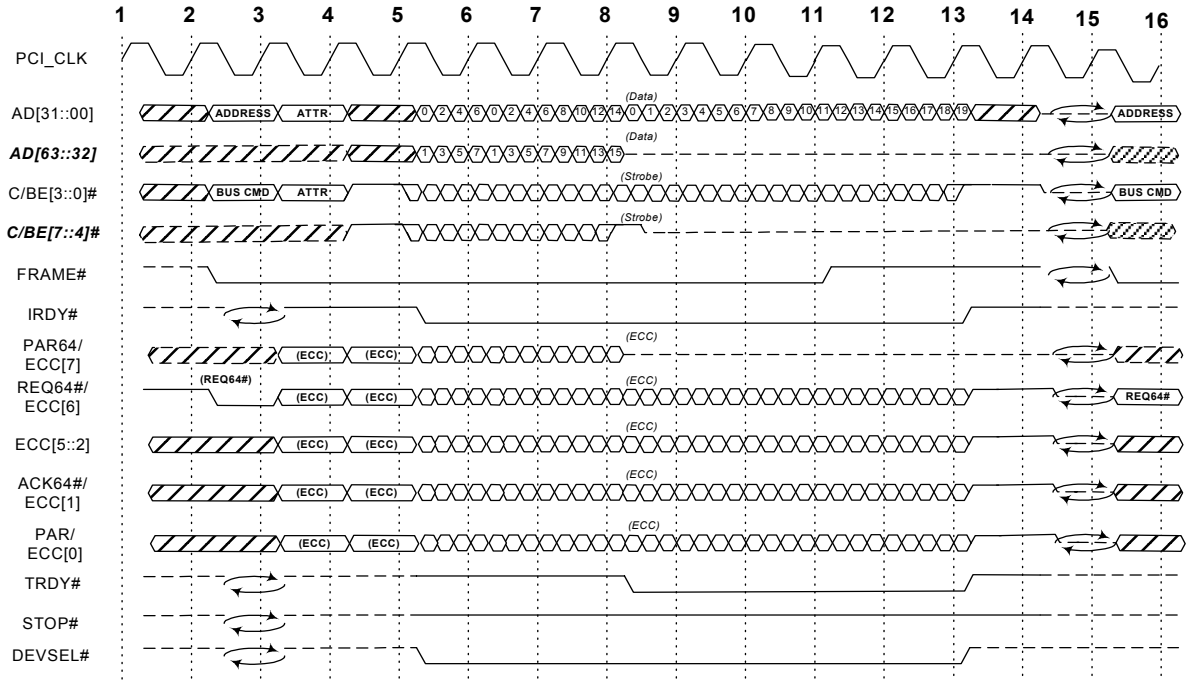

Figure 2-93: 64-bit Initiator Burst Push Addressing a 32-bit Target, Decode B and Two Initial Wait States, Mode 2

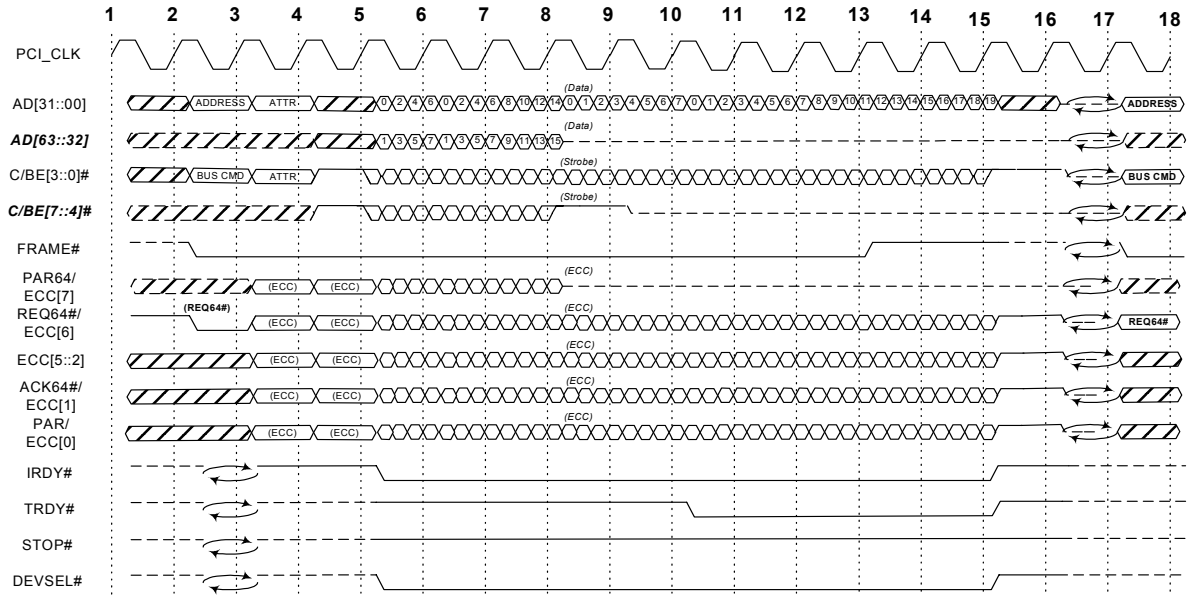

Figure 2-94: 64-bit Initiator Burst Push Addressing a 32-bit Target, Decode B and Four Initial Wait States, Mode 2

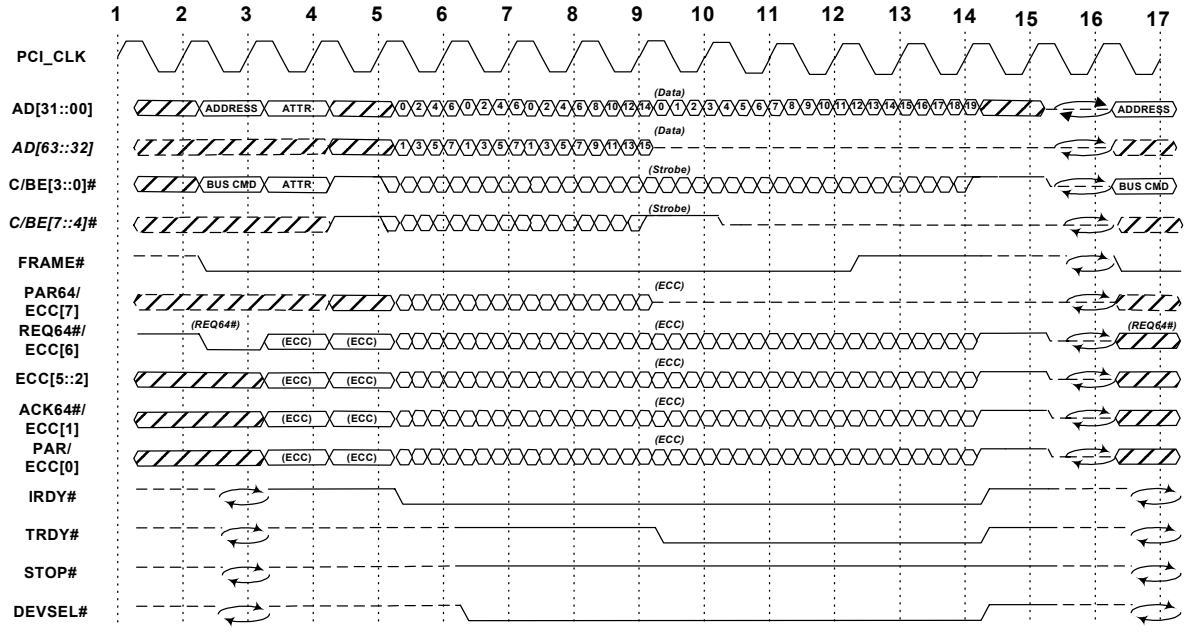

Figure 2-95: 64-bit Initiator Burst Push Addressing a 32-bit Target, Decode C and Two Initial Wait States, Mode 2

2.12.2. 16-Bit Bus Width

PCI-X Mode 2 devices (other than bridges, which are described below) are required to support a 16-bit version of the AD bus. A device designed exclusively for embedded applications (not to connect directly to an add-in card connector) is permitted to support only a 16-bit interface. (64- and 32-bit support is not required for devices designed exclusively for embedded applications.)

A 16-bit bus always operates in PCI-X Mode 2 (i.e., either in PCI-X 266 mode or PCI-X 533 mode). Devices designed exclusively for 16-bit embedded applications are exempt from all requirements that do not apply when operating in PCI-X Mode 2. For example, such a device has no need for parity protection or conventional PCI protocol support, and Category 1 signals (as defined in Table 2-1, "Driver and Receiver Categories," in PCI-X EM 2.0) use exclusively 1.5V signaling.

Table 2-22: 16-Bit Bus Pin Sharing

Device External Signal Name	Device Internal 32-Bit Signal Usage	
	16-Bit Interface First Phase	16-Bit Interface Second Phase
AD[16]	AD[00]	AD[16]
AD[17]	AD[01]	AD[17]
AD[18]	AD[02]	AD[18]
AD[19]	AD[03]	AD[19]
AD[20]	AD[04]	AD[20]
AD[21]	AD[05]	AD[21]
AD[22]	AD[06]	AD[22]
AD[23]	AD[07]	AD[23]
AD[24]	AD[08]	AD[24]
AD[25]	AD[09]	AD[25]
AD[26]	AD[10]	AD[26]
AD[27]	AD[11]	AD[27]
AD[28]	AD[12]	AD[28]
AD[29]	AD[13]	AD[29]
AD[30]	AD[14]	AD[30]
AD[31]	AD[15]	AD[31]
ECC[2]	ECC[0]	ECC[2]
ECC[3]	ECC[1]	ECC[3]
ECC[4]	ECC[6]	ECC[4]
ECC[5]	E16	ECC[5]
C/BE[2]#	C/BE[0]#	C/BE[2]#
C/BE[3]#	C/BE[1]#	C/BE[3]#

As with all Mode 2 electrical specifications, only two devices with a point-to-point connection are supported.

The 16-bit bus uses the same control signals as 64- and 32-bit buses (except for LOCK#) but only half of the AD, C/BE#, and ECC buses. For source-synchronous data phases, only the single pair of source-synchronous data strobes that share these two C/BE# pins is used.

As shown in Table 2-22, information is driven on the 16-bit interface by taking the information from the device's equivalent interval 32-bit interface and driving it on the external 16-bit interface in two phases or subphases. The table shows the external pins that would be shared for 16-bit devices that also support the 32-bit interface.

The 16-bit interface is optimized for embedded applications. No add-in card connector is specified and no status bit in configuration space defines this interface width. (The 64-bit Device bit in the PCI-X Status register is intended to aid hot-plug or system-management software make recommendations to the user as to which slots are best for a particular card. No such requirement exists for the 16-bit interface.)

All transactions on a 16-bit bus are 16 bits wide. There is no width negotiation protocol. The interface of all devices connected to a 16-bit bus are permanently set in the 16-bit mode. The means by which the device selects its 16-bit interface is not specified, but the electrical connection of dedicated mode pins is one alternative.

An implementation using a 16-bit bus is limited to a single subordinate level in the PCI Configuration Space hierarchy. That is, bridges are not permitted to connect a 16-bit PCI bus to a subordinate PCI bus of any width. PCI-X bridges do not implement 16-bit primary interfaces. A bridge with a 64- or 32-bit interface on its primary side optionally implements one or more 16-bit interfaces on its secondary side. If a PCI-X bridge is designed exclusively for embedded applications (i.e., not to connect its secondary interface to an add-in card slot), the bridge is permitted to implement only a 16-bit interface on its secondary side (i.e., such a bridge is not required to support a 64- or 32-bit secondary interface).

Host bridges optionally implement one or more 16-bit interfaces for the buses they create. If a host bridge is designed exclusively for embedded applications (i.e., not to connect to an add-in card slot), the host bridge is permitted to implement only a 16-bit interfaced on the bus it creates (i.e., such a host bridge is not required to support a 64- or 32-bit interface).

Exclusive access is not forwarded across a 16-bit bus. If a bridge forwards a locked transaction to a 16-bit bus, lock is automatically established on the secondary bus when the transaction executes on that bus. (See Section 8.5.)

A 32-bit address is multiplexed on the 16-bit AD bus. The address predrive of configuration transactions as described in Section 2.7.2.1 is not used on a 16-bit bus. The source bridge for a 16-bit bus must do one of the following for device selection for configuration transactions:

- Provide separately decoded IDSEL output pins for each 16-bit interface.
- Require that the IDSEL input of the device be pulled high externally when using the 16-bit interface and the bridge guarantees that only configuration transactions addressing a single bus number and device number are forwarded to that 16-bit interface.

Transactions on 16-bit buses use the same seven-bit ECC defined for 32-bit transfers, with one additional check bit for improved detection of uncorrectable errors. See Section 5.1.3 for ECC requirements for 16-bit transactions.

IMPLEMENTATION NOTE

Compliance Testing of the 16-Bit Interface

For convenience, compliance testing generally requires standard electrical and mechanical interfaces and form factors. Since the 16-bit interface is intended for embedded applications, no such standards exist.

However, since the 16-bit interface is a subset of the 32-bit interface (which does have such a standard add-in card electrical and mechanical specification), compliance testing of the 16-bit interface can conveniently leverage this infrastructure. Specially modified system boards that use only 16 bits of the 32-bit connector can be used to test specially modified add-in cards.

Designers of devices that support both the 16-bit and either the 64- or 32-bit interface are encouraged to provide an add-in card with a convenient method of strapping the card into 16-bit mode for compliance testing. Designers of devices that exclusively support the 16-bit interface are encouraged to implement a test card with a 32-bit add-in card connector specifically for compliance testing.

2.12.2.1. Address Width

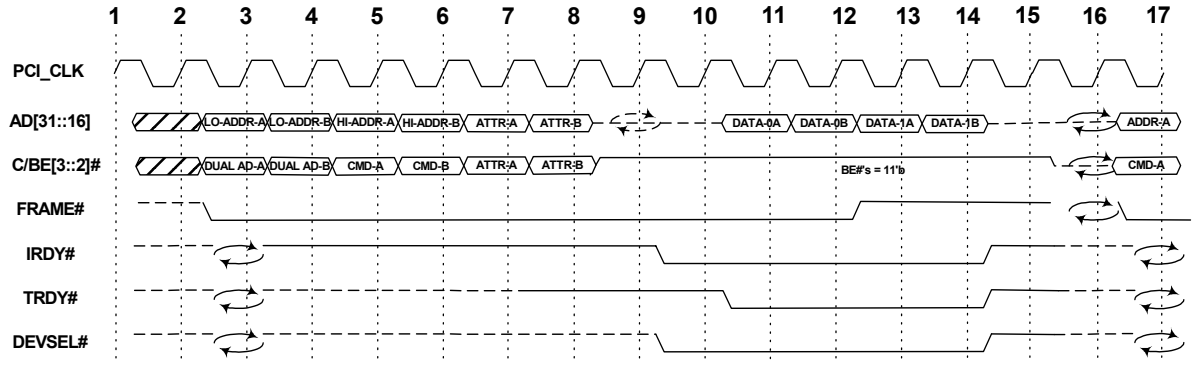

Figure 2-96: 64-Bit Address in a 16-Bit Transaction

In 16-bit transactions, a 32-bit address requires two address phases. The initiator drives the address and command in the two address phases as indicated in Table 2-22. (See Section 5.1.3 for the requirements for the ECC check bits.) A 64-bit address requires four address phases. The initiator drives the least significant 32 bits of the address and the first four bits of the command in the first two address phases (as shown in Table 2-22) and the most significant 32 bits of the address and the second four bits of the command in the next

two address phases (also as shown in Table 2-22). (The first four bits of the command are the Dual Address Cycle command and the second four bits are the actual command, the same as for 64- and 32-bit buses.)

2.12.2.2. Attribute Width

In 16-bit transactions, there are always two attribute phases. Attribute bits that appear on AD[31::0] and C/BE[3::0]# in 32-bit transactions are driven by the initiator on the 16-bit bus as shown in Table 2-22.

2.12.2.3. Data Transfer Width

In 16-bit transactions, a common-clock data phase includes two bytes of data, a PCI-X 266 source-synchronous data phase includes four bytes of data, and a PCI-X 533 source-synchronous data phase includes eight bytes of data.

As with 64- and 32-bit transfers, the minimum unit of data transfer (ignoring byte enables) in a 16-bit transfer is one DWORD. Both the initiator and target are required to transfer DWORD-aligned pairs of common-clock data phases or source-synchronous subphases. (ECC protection for this data is also based on DWORDs.)

For common-clock transactions, initiators are required to transfer DWORD-aligned pairs of data phases regardless of the starting address and byte count. DWORD transactions that successfully transfer data on 16-bit buses have two data phases. Common-clock burst transactions that successfully transfer data on 16-bit buses have an even number of data phases. Common-clock bursts always start and end on DWORD boundaries. If the starting address has AD[1::0]=10b (that is, byte addresses 00b and 01b are before the starting address), the transaction starts at address 00b anyway, and the data values for bytes 00b and 01b are ignored, except for ECC calculations. Similarly, if the ending address has AD[1::0]=01b (that is, byte addresses 10b and 11b are after the ending address), the transactions ends at address 11b anyway, and the data values for bytes 10b and 11b are ignored, except for ECC calculations.

2.12.2.3.1. Burst Transactions on a 16-Bit Bus

Figure 2-97 through Figure 2-101 illustrate burst transactions on a 16-bit bus. The figures show the view of the transactions not only on the bus, but also inside both the initiator and target, using the format described in Section 2.6.1.1.

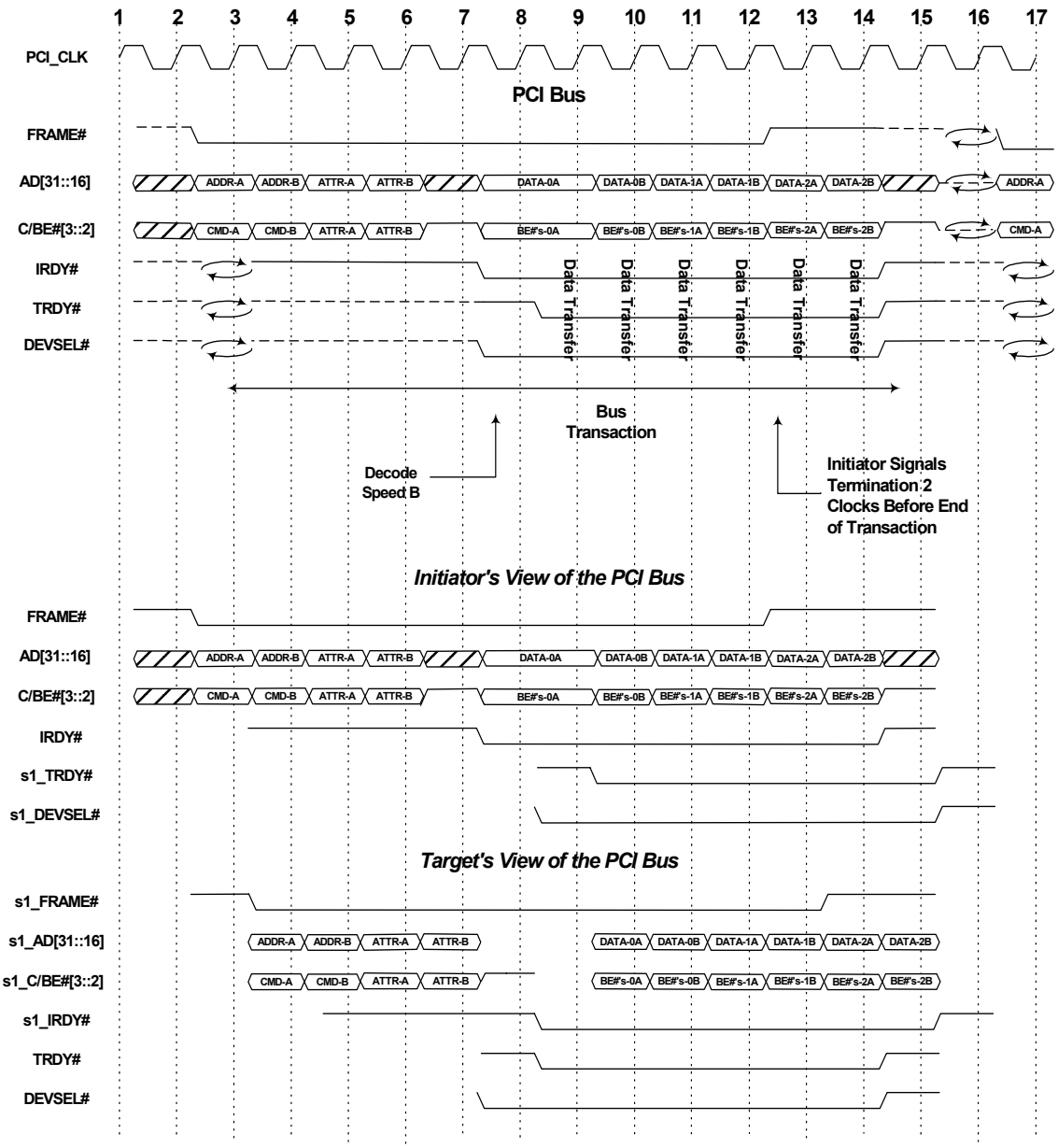

Figure 2-97: Common-Clock Burst Write with No Target Initial Wait States, 16-Bit Mode 2

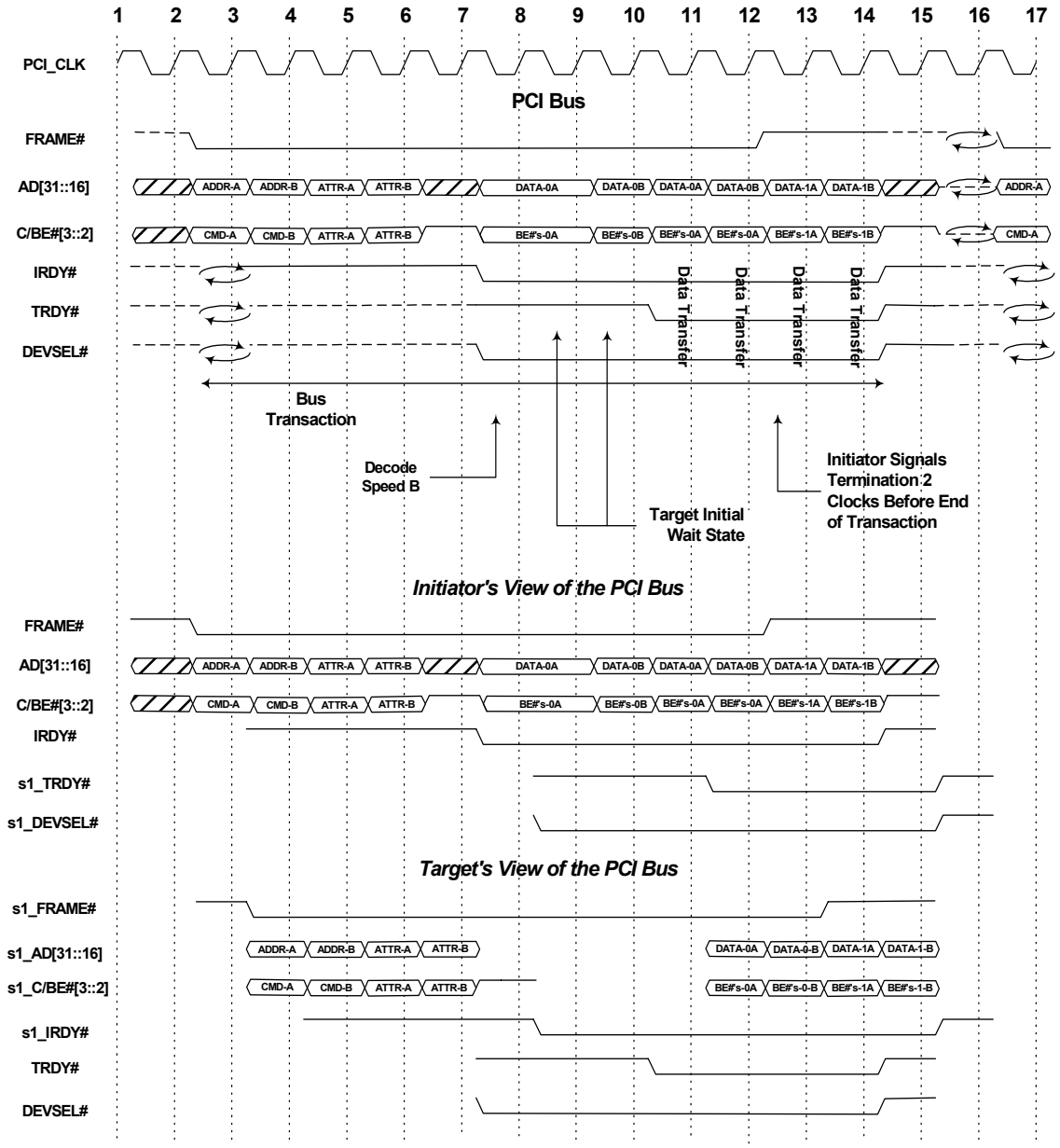

Figure 2-98: Common-Clock Burst Write with Two Target Initial Wait States, 16-Bit Mode 2

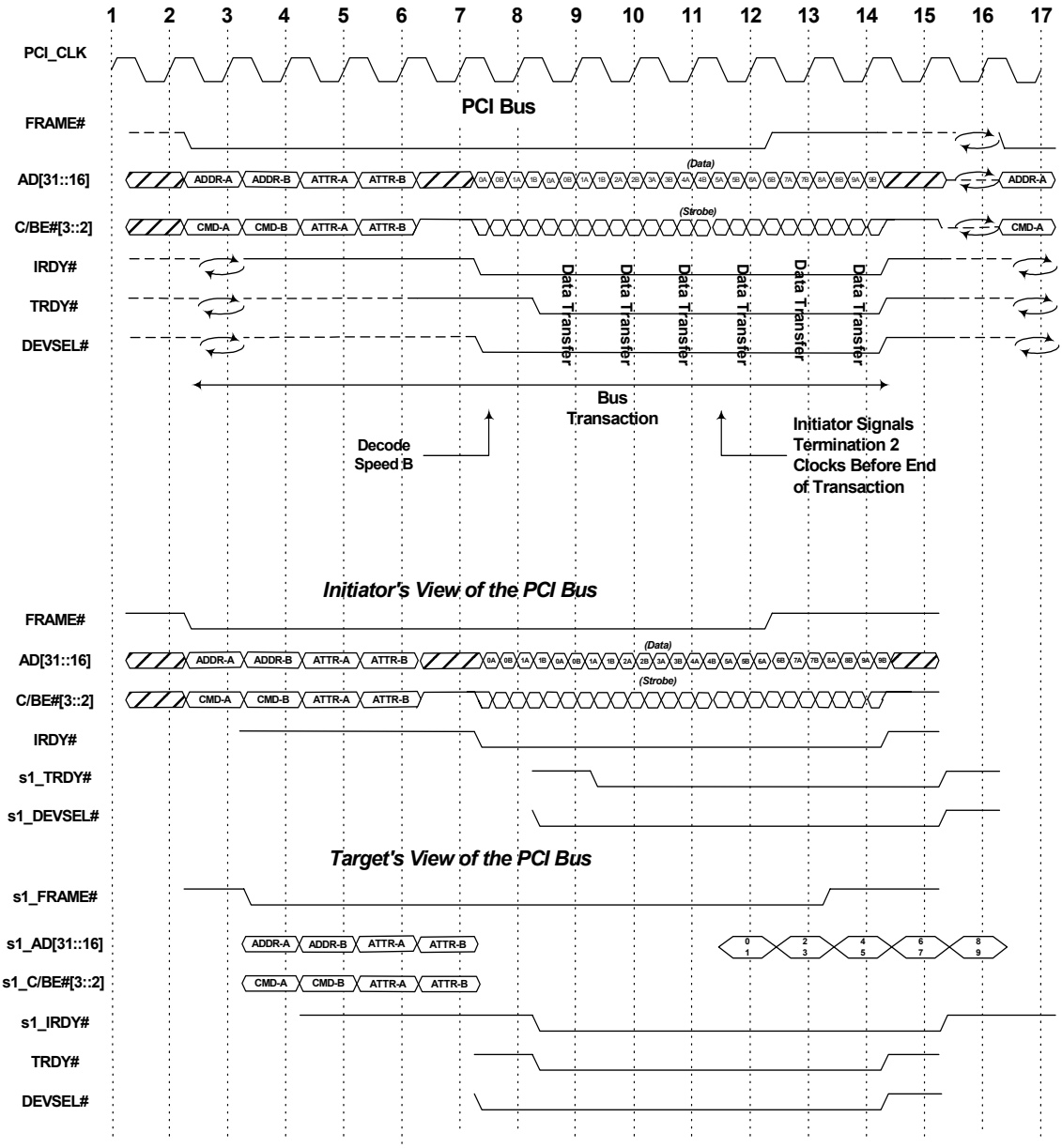

Figure 2-99: Source-Synchronous (PCI-X 533) Burst Write with No Target Initial Wait States, 16-Bit Mode 2

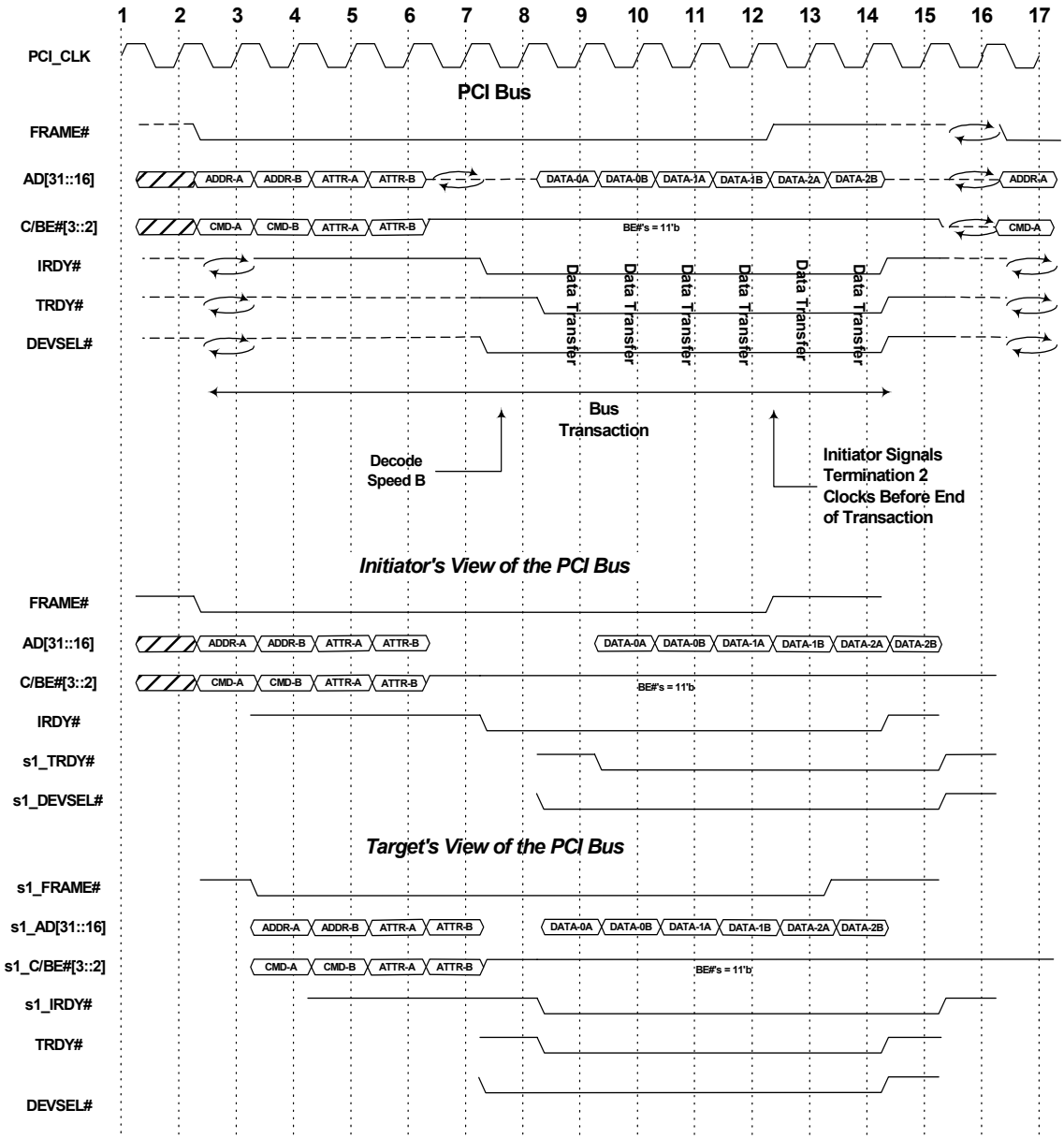

Figure 2-100: Common-Clock Burst Read with No Target Initial Wait States, 16-Bit Mode 2

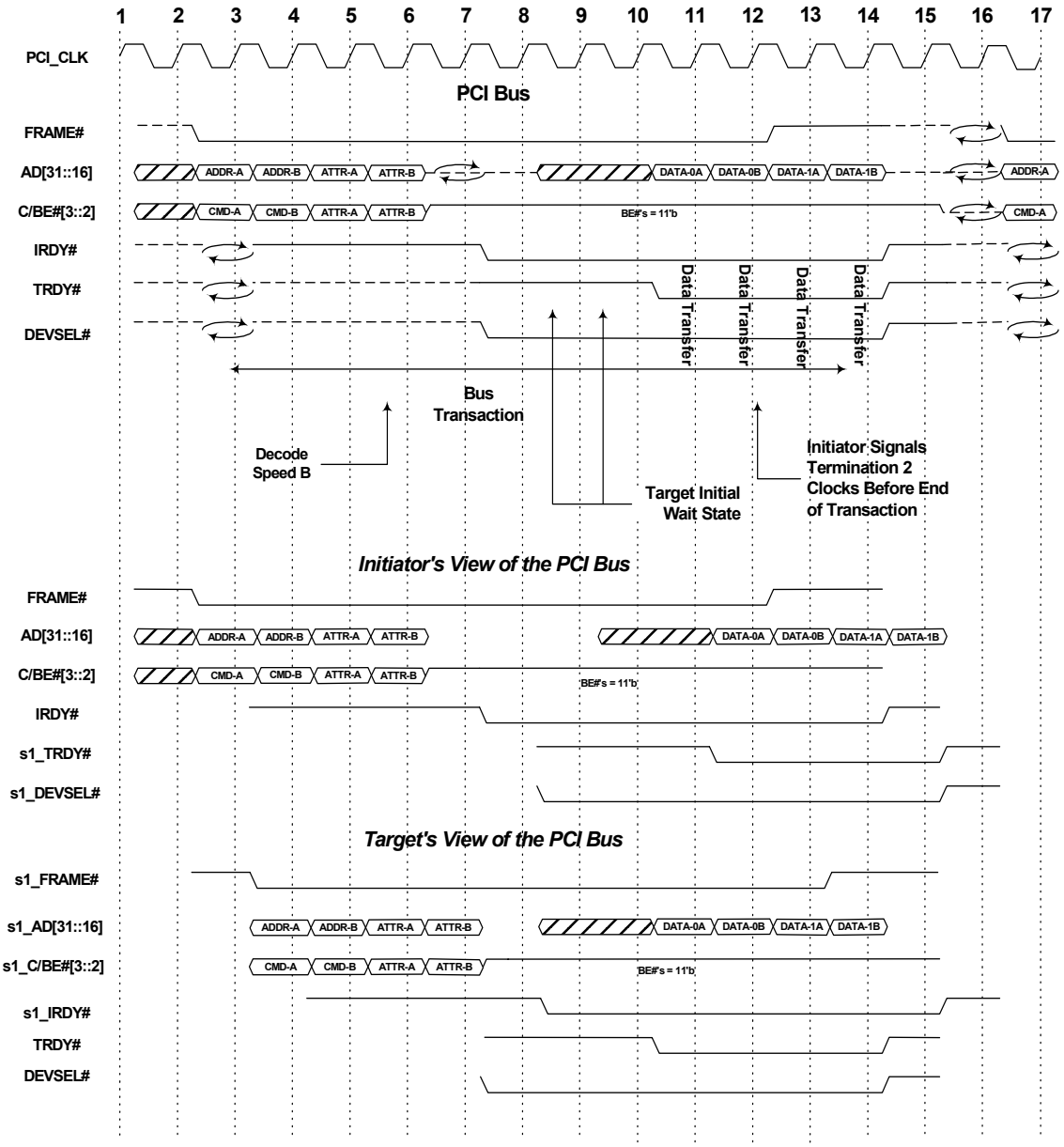

Figure 2-101: Common-Clock Burst Read with Two Target Initial Wait States, 16-Bit Mode 2

2.12.2.3.2. DWORD Transactions on a 16-Bit Bus

Figure 2-102 and Figure 2-103 illustrate that DWORD transactions that successfully transfer data on a 16-bit bus always have two data phases. Configuration transactions on a 16-bit bus do not include the address predrive defined in Section 2.7.2.1, and use the same timing as other DWORD transactions.

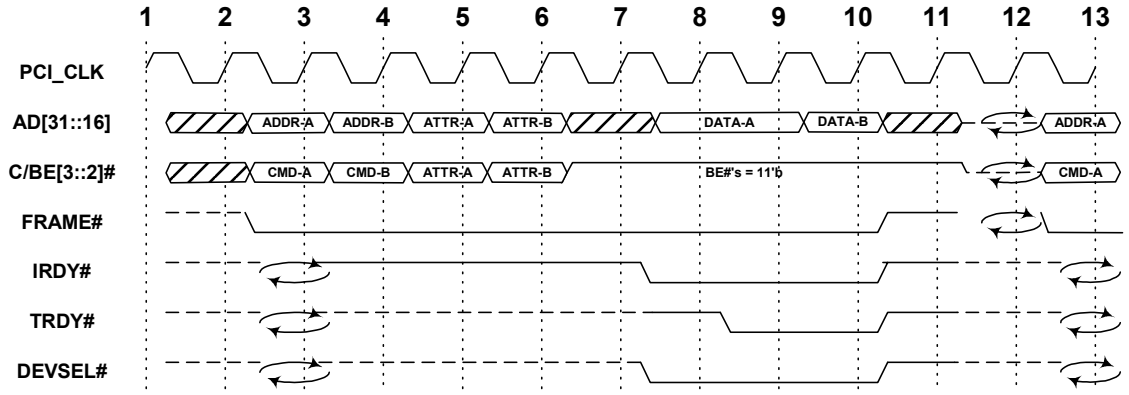

Figure 2-102: DWORD Write with No Wait States and Data Transfer, 16-Bit Mode 2

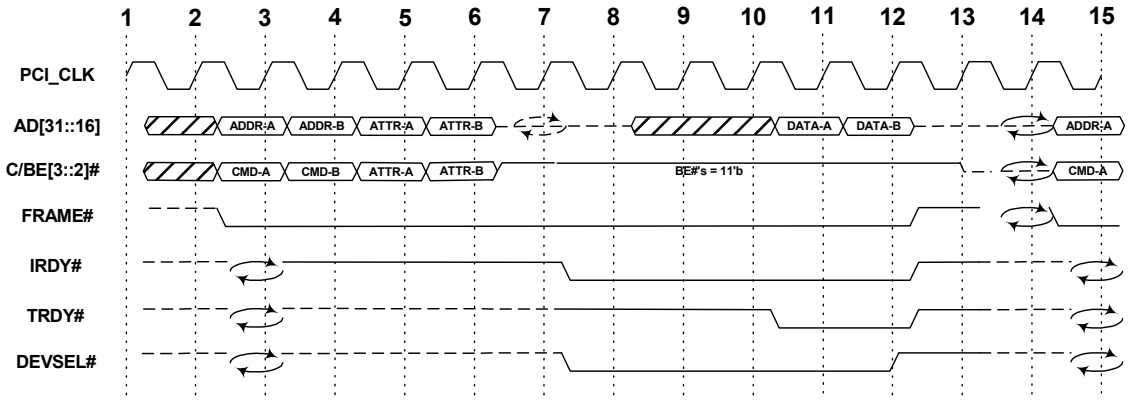

Figure 2-103: DWORD Read with No Wait States and Data Transfer, 16-Bit Mode 2

2.12.2.3.3. Target Data Phase Signaling on a 16-Bit Bus

For common-clock transactions, targets are required to signal data phase action in pairs of clocks for those transactions that transfer data. That is, if the target signals Data Transfer, Single Data Phase Disconnect, or Split Response, the target must signal them for two consecutive clocks. The target is permitted to signal Disconnect at Next ADB on any clock as shown in Section 2.11.2.2 (except that common-clock transactions always have an even number of data phase).

If the target signals Target-Abort on either data phase of a data-phase pair on a 16-bit common-clock transaction, both data phases are discarded and the transaction ends.

In all other cases, including source-synchronous transactions and common-clock transactions in which the target signals Retry or Target-Abort, the target signals its data phase action the same as for 64- and 32-bit buses.

Figure 2-104, Figure 2-105, and Figure 2-106 show the target signaling Single Data Phase Disconnect and Split Response for two data phases on common-clock transfers.

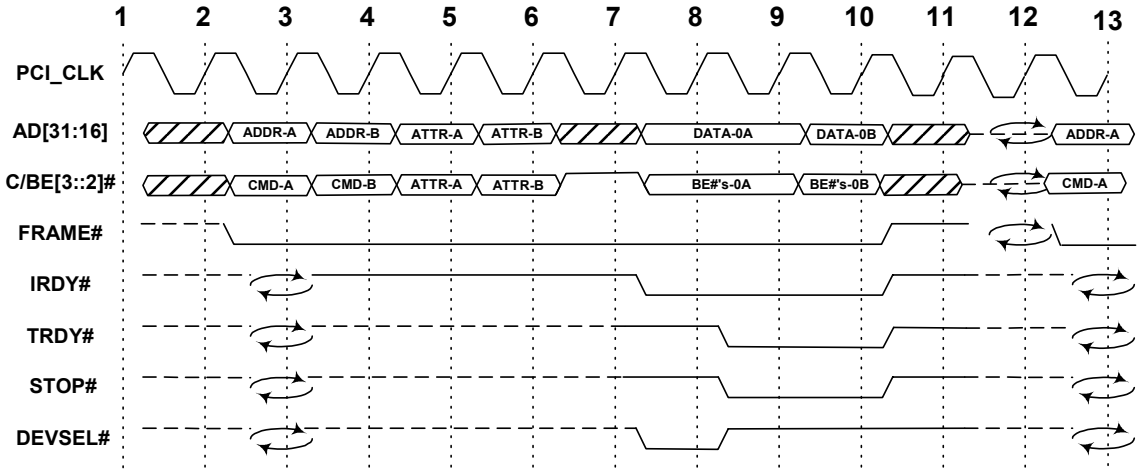

Figure 2-104: Single Data Phase Disconnect on Common-Clock 16-Bit Transaction

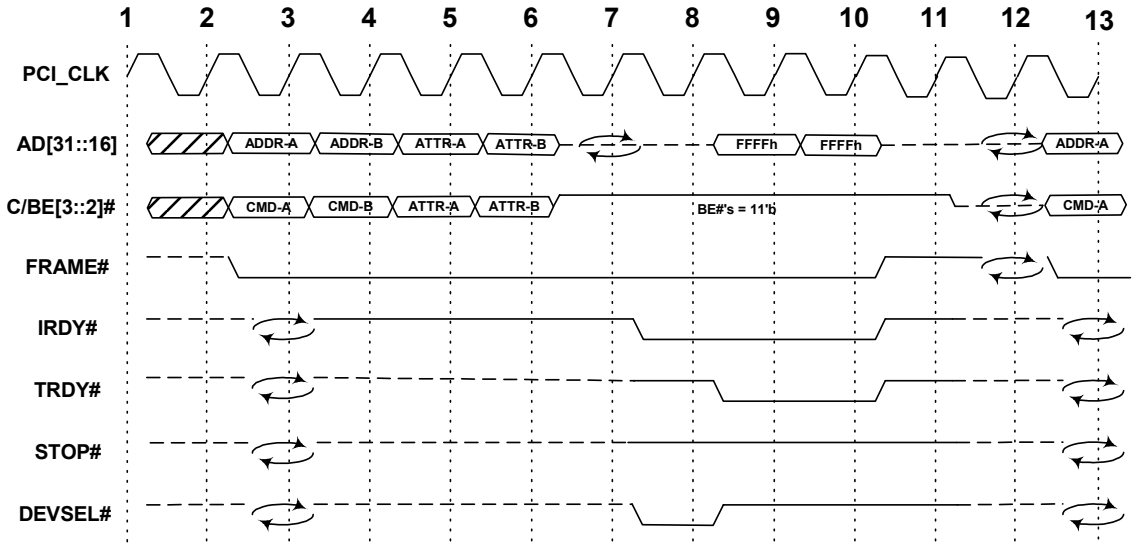

Figure 2-105: Split Response to Common-Clock 16-Bit Read Transaction

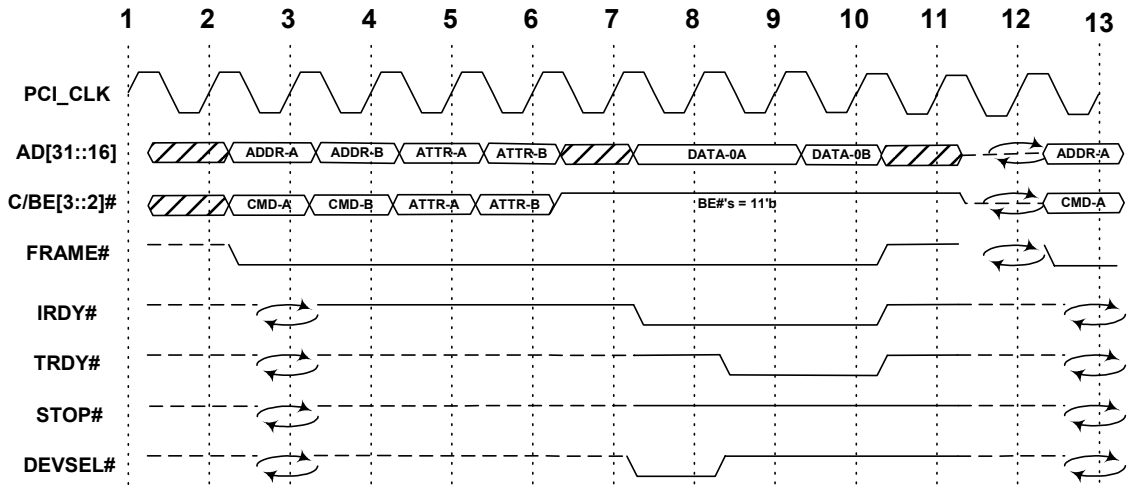

Figure 2-106: Split Response to Common-Clock 16-Bit Write Transaction

2.13. Required Acceptance and Completion Rules for Simple Devices

Using the terminology of PCI 2.3, 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 2.3 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.)

Table 2-23: PCI-X Write Completion Limit

Clock Frequency Range	Write Completion Limit	Units
50-66 MHz	133	clocks
66-100 MHz	200	clocks
100-133 MHz	267	clocks

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 the number of clocks shown in Table 2-23, based on the frequency range selected in 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 2.3 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 2.3 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. PCI HP 1.1 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 $\boldsymbol{D3_{hot}}$. (See Section 3.3.)
- Software-initiated reset of the add-in 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.

2.16. Device ID Messaging

A device ID message is a Sequence using the Device ID Message command. Device ID message transactions are burst transactions that either explicitly address the completer by using the Completer ID (the completer bus number, completer device number, and completer function number) or else implicitly address the host bridge. Device ID message transactions that explicitly address the completer (Route Type = 0b) are called "explicit device ID messages" and are said to use "explicit routing." Device ID messages that

implicitly address the host bridge (Route Type = 1b) are called "implicit device ID messages" and are said to use "implicit routing." Bridges forward explicit device ID messages based on the completer bus number (the same as Type 1 configuration transactions and Split Completions). The completer decodes the Completer ID and asserts DEVSEL# if it matches the ID of a function within the device. Bridges forward implicit device ID messages only upstream (towards the host bridge).

Device ID messages are permitted to be of any length from one byte to 4096 bytes. They are subject to the same bus width rules as other burst transactions.

PCI-X Mode 1 and Mode 2 non-bridge devices (Type 00h Configuration Space header) and host bridges optionally support device ID messaging. Some host bridges and bridges to interfaces other than PCI use Type 01h configuration headers. Support of device ID messages is optional for these devices, also. If a device supports device ID messages, it decodes and accepts transactions using the Device ID Message command with explicit routing (Route Type = 0b) and the appropriate Completer ID. If a host bridge supports device ID messages, it also decodes and accepts transactions using the Device ID Message command with implicit routing (Route Type = 1b).

All PCI-X Mode 2 bridges are required to support forwarding of device ID messages. Forwarding support is optional in PCI-X Mode 1 bridges. Bridges indicate support for device ID messaging via the Device ID Messaging Capable bit in the PCI-X Bridge Status register. Bridges optionally support device ID messages as the requester or completer. That is, they optionally create device ID messages, and optionally accept device ID messages addressed to their completer ID.

If a PCI-X bridge (Type 01h Configuration Space header) supports forwarding of device ID messages, that bridge decodes and forwards explicit device ID message transactions with a completer bus number on the other side of the bridge, independent of whether the interfaces are operating in PCI-X Mode 1 or PCI-X Mode 2. (Device ID messages are not supported in conventional mode. See Section 8.4.3.2.) A bridge that supports forwarding of device ID messages decodes and forwards implicit device ID message transactions from its secondary interface to its primary interface, if the primary interface is operating in PCI-X mode. A device that integrates more than one PCI-X bridge that supports forwarding of device ID messages (i.e., it is the source bridge for more than one secondary bus) must forward explicit device ID messages from a requester on any of its interfaces to a completer on any other of its interfaces operating in PCI-X mode. Such a bridge must also forward implicit device ID messages from a requester on any of its secondary interfaces to its primary interface if the primary interface is operating in PCI-X mode.

Error handling for device ID messages is affected by the state of the Silent Drop bit in the Device ID Message Address (see Section 2.16.1). If the Silent Drop bit is 0, device ID messages are handled from an error handling and reporting standpoint as described elsewhere in this specification for memory write transactions. If the Silent Drop bit is 1, some errors are ignored as described below. The Silent Drop bit does not affect the handling of parity or ECC errors.

If a device that supports device ID messages receives a message as a completer using an unsupported message or a reserved message class and the Silent Drop bit is 1, it accepts all data phases of the message, discards them, and takes no further action. If the Silent Drop bit is 0, the device terminates the transaction with Target-Abort. If a device does not

support device ID messages, it ignores transactions using the Device ID Message command regardless of the state of the Silent Drop bit. That is, it does not assert DEVSEL# and allows the transaction to end in a Master-Abort.

If a bridge that supports forwarding of device ID messages receives a message, the destination bus is operating in conventional PCI mode, and the Silent Drop bit is 1, the bridge accepts all data phases of the message, discards them, and takes no further action. If the Silent Drop bit is 0, the bridge terminates the transaction with Target-Abort.

If an initiator (requester or bridge) initiates a transaction using the Device ID Message command with the Silent Drop bit 1 and no device responds (i.e., Master-Abort), the initiator takes no error action. That is, the initiator does not set the Received Master-Abort bit in the Status register or assert SERR#. If the target (completer or bridge) signals Target-Abort, the initiator's actions are the same as described elsewhere in this specification for memory write transactions, independent of the state of the Silent Drop bit.

See Section 2.6.1 for the prohibition on the use of Single Data Phase Disconnect and Split Response on Device ID Message transactions.

For purposes of target initial latency, buffering and required acceptance rules, and transaction ordering, targets (completers and bridges) treat the device ID message transaction as if it were a memory write transaction.

IMPLEMENTATION NOTE

Forwarding Device ID Messages Across a Host Bridge

Device ID messages provide the greatest benefit to the user if all PCI-X devices can send messages to all other devices, regardless of whether those other devices are subordinate to the same host bridge or a different one. Forwarding messages across a host bridge is critical as the number of slots supported by each host bridge decreases because of electrical loading limitations at the higher bus speeds.

PCI 2.3 permits chip sets not to forward conventional PCI transactions from a device subordinate to one host bridge to a device subordinate to another. PCI-X allows this same limitation. However, forwarding of device ID messages alone is much simpler than forwarding all transactions. Device ID messages are burst push transactions, so like memory write transactions, they are posted, have simple ordering implications, and have no subsequent completions.

Host bridges are strongly encouraged to support the forwarding of device ID messages to the fullest extent possible. In some system architectures, host bridges for different slots are connected only indirectly through a complex path to the memory controller, and forwarding device ID messages across this path would be quite complex. Forwarding device ID messages across these most complex host bridges is not required. Other host bridges are more closely coupled. These host bridges should support the forwarding of device ID messages to provide the greatest flexibility to the user.

2.16.1. Device ID Message Address

The Device ID Message Address (DIM Address) is driven on the AD bus during the address phase of Device ID Message transactions. It is permitted to use either a single address cycle or a dual address cycle. Figure 2-107 shows the format for the DIM Address for a single address cycle and Figure 2-108 shows the format for a dual address cycle. Table 2-24 describes each field.

Device ID messages do not use any of the address spaces defined for other commands (Memory Space, I/O Space, or Configuration Space), however, for disconnection purposes, the data phases are treated as if the transaction addressed Memory Space and began on an ADB. If the transaction uses a dual address cycle, the address bits are used as shown in Figure 2-108 and the bus drive and float requirements are as defined in Section 2.12.1.1 for 64- and 32-bit buses and Section 2.12.2.1 for 16-bit buses.

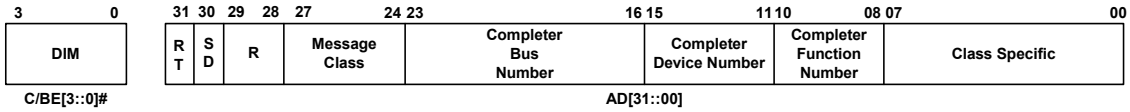

Figure 2-107: DIM Address Format, Single Address Cycle

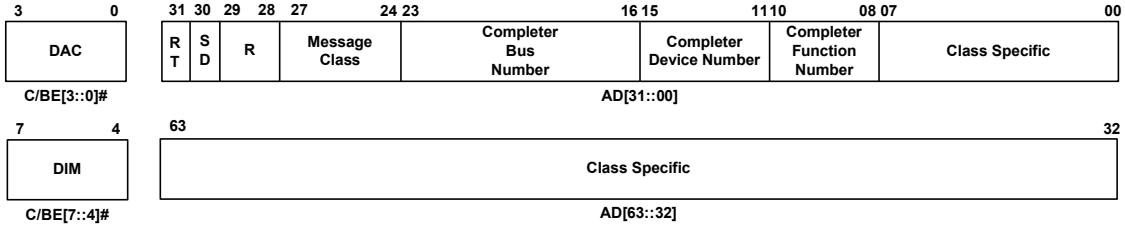

Figure 2-108: DIM Address Format, Dual Address Cycle

Table 2-24: DIM Address Field Definitions

Field	Function
Route Type (RT)	If this bit is 0, the transaction explicitly addresses the completer using the Completer ID. If this bit is 1, the transaction implicitly addresses the host bridge.
Silent Drop (SD)	If this bit is 0, the error conditions for device ID message transactions are treated the same as error conditions for memory write transactions. If this bit is 1, some errors are ignored as described in Section 2.16.

Field	Function
Reserved (R)	Must be set to 0 by the requester and ignored by the completer (except for parity or ECC calculations). PCI-X bridges that forward the transaction forward these bits unmodified.
Message Class	Identifies one of 16 classes of message encoding. Class — Usage 0 — Vendor Defined. DIM Address bits 7-0 plus bits 63-32 (if a dual address cycle) are available for vendor definition. 1-15 — reserved
Completer Bus Number	The completer uses this information to identify the appropriate explicitly routed Device ID Messages. This field is reserved when implicit routing is used. Reserved fields are set to 0 by the requester and ignored by the completer (except for parity or ECC calculations), and forwarded unmodified by PCI-X bridges. A PCI-X bridge uses this field to identify transactions to forward. If this field of an explicitly routed Device ID Message 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 an explicitly routed Device ID Message on the primary bus is between the bridge's secondary bus number and subordinate bus number, inclusive, and the secondary interface is operating in PCI-X mode, the bridge forwards the transaction downstream. If the bridge forwards the Device ID Message to another bus operating in PCI-X mode, it leaves this field unmodified.
Completer Device Number	The completer uses this information to identify the appropriate explicitly routed Device ID Messages. This field is reserved when implicit routing is used. Reserved fields are set to 0 by the requester and ignored by the completer (except for parity or ECC calculations), and forwarded unmodified by PCI-X bridges. If a PCI-X bridge forwards the Device ID Message to another bus operating in PCI-X mode, it leaves this field unmodified.
Completer Function Number	The completer uses this information to identify the appropriate explicitly routed Device ID Messages. This field is reserved when implicit routing is used. Reserved fields are set to 0 by the requester and ignored by the completer (except for parity or ECC calculations), and forwarded unmodified by PCI-X bridges. If a PCI-X bridge forwards the Device ID Message to another bus operating in PCI-X mode, it leaves this field unmodified.
Class Specific	Contents are defined according to the Message Class. Messages optionally also use a Dual Address Cycle to provide an additional 32 class-specific bits.

2.16.2. Device ID Message Attributes

The attribute phase of a Device ID Message transaction contains the Device ID Message Attributes (DIM Attributes). Figure 2-109 shows the format for the DIM Attribute and Table 2-25 describes each field. The Requester Bus Number, Requester Device Number, Requester Function Number, Upper Byte Count, and Lower Byte Count are the same as defined in Section 2.5 for Requester Attributes for burst and DWORD transactions.

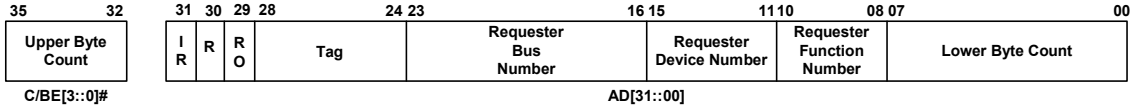

Figure 2-109: DIM Attribute Format

Table 2-25: DIM Attribute Field Definitions

Field	Function
Initial Request (IR)	This bit is set if this transaction is the first transaction of the Sequence. The bit is cleared if this transaction is a continuation of the Sequence, i.e., the Sequence had been previously disconnected either by the completer, requester, or intervening bridge.
Reserved (R)	Must be set to 0 by the requester and ignored by the completer (except for parity or ECC calculations). PCI-X bridges that forward the transaction forward this bit unmodified.
Relaxed Ordering (RO)	A requester is permitted to set this bit on a device ID message Sequence if its usage model does not require this device ID message Sequence to stay in order with respect to other device ID message Sequences and memory write Sequences moving in the same direction. (Device ID message data for the same Sequence always stays in its original order.) If a transaction is disconnected, the requester must not change the value of this attribute on any subsequent transaction in the same Sequence. Use of this bit is optional for targets. If the bit is set, the completer is permitted to allow this device ID message transaction to pass previously posted device ID message and memory write transactions moving in the same direction. PCI-X bridges ignore this bit for device ID message transactions and forward them in the order in which they were received. A PCI-X bridge forwarding the transaction to another bus operating in PCI-X mode forwards this bit unmodified with the transaction. See Section 8.4.4 for the details of transaction ordering rules.

Field	Function
Tag	This 5-bit field uniquely identifies up to 32 device ID message Sequences from a single requester. The requester 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. 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. (See Table 2-9.)
Requester Device Number	This 5-bit field contains the device number assigned to the requester. (See Table 2-9.)
Requester Function Number	This 3-bit field contains the function number of the requester within the device. (See Table 2-9.)
Upper Byte Count, Lower Byte Count	This 12-bit field contains the byte count of the transaction. (See Table 2-9.)

2.16.3. Device ID Message Format

The device ID message is driven during the data phase or phases of the Device ID Message transaction. Device ID Message transactions follow the same rules for data phases as other burst push transactions.

2.16.3.1. Vendor-Defined Message Class

Device ID Message Class 0 is used for vendor-defined messages. All of the bits in the Class Specific field in the DIM Address for Message Class 0 are available for vendor definition. In addition, the entire message payload (i.e., the data in each of the data phases of the transaction) is available for vendor definition.

2.16.3.2. Reserved Message Classes

The format of all other device ID message classes is reserved for future use by the PCI-SIG. Devices must not initiate transactions using reserved message classes and must ignore any device ID message address to them (not assert DEVSEL#) if the transaction uses a reserved message class.

2.17. PCI-X Mode 2 Bus Drive and Turn-Around

The electrical drive and termination requirements of PCI-X Mode 2 specify that all input receivers for Category 1 signals (as defined in Table 2-1, "Driver and Receiver Categories," in PCI-X EM 2.0) be disabled when the bus is not being driven. This leads to protocol differences between Mode 1 and Mode 2 in the following areas:

- Lower bus at the beginning and end of transactions
- Upper bus at the beginning and end of transactions
- AD and ECC buses during read transactions

The lower 32-bit portion of the buses (AD[31::00], C/BE[3::0]#, ECC[6::0]) must be driven at all times, except during precisely defined bus turn-around clocks. See Section 4.1.2.1 for details.

The initiator does not drive and all input receivers are disabled for the upper 32-bit portion of the buses (AD[63::32], C/BE[7::4]#, ECC[7]) except during 64-bit data phases. See Section 2.12.1.3 for details.

Turn-around of the AD and ECC buses for data phases of immediate read transactions is also precisely controlled. The clock after the attribute phase of a read transaction is reserved for turning around the AD bus. The initiator stops driving the AD bus and the target disables its receivers the clock after the attribute phase. The target begins driving the AD bus and the initiator enables its receivers one clock after the target asserts DEVSEL# (the same time the target starts driving the AD bus in Mode 1). At the end of the transaction, the target stops driving the AD bus and the initiator disables its receivers the clock after the last data phase, both for burst and DWORD read transactions. The ECC bus lags the AD bus by one clock (like PAR and PAR64 in parity mode). (All read-transactions are common-clock.)

3

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 2.1.2.4.2, "3.3V Environment Timing Parameters," and Section 2.1.3.4.1, "1.5V Environment Common-Clock Timing Parameters," in PCI-X EM 2.0 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 Mode 1 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#. Similar logic would be required in PCI-X Mode 2.

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.

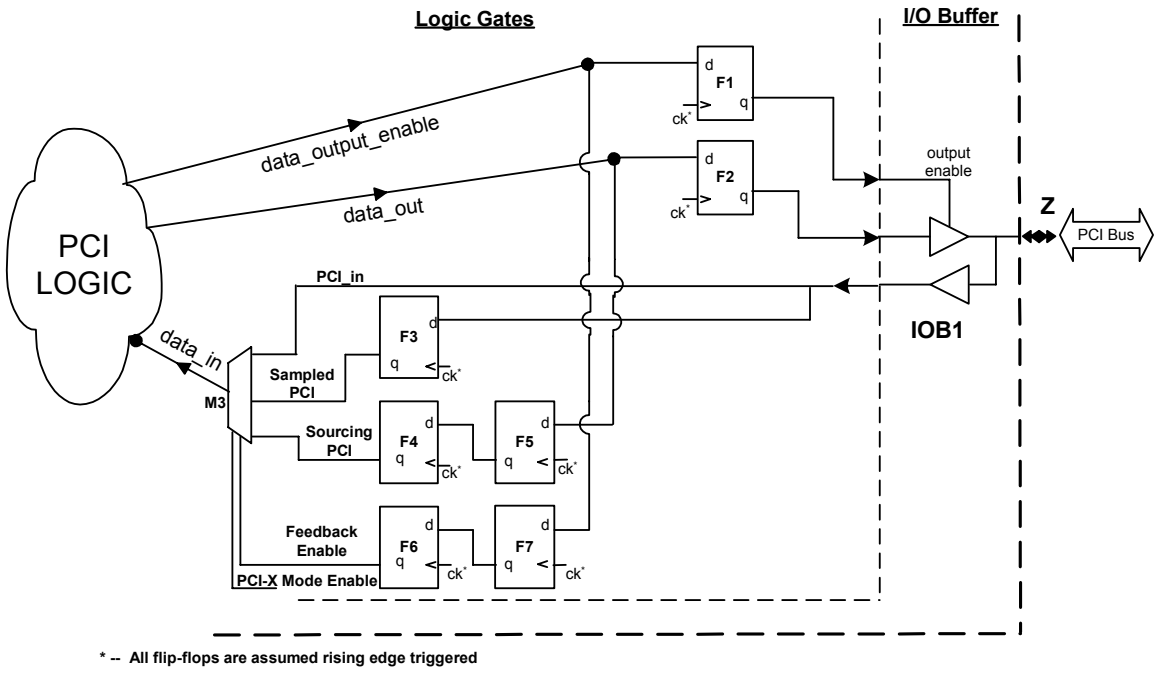

Figure 3-1: A Logic Block Diagram for Bypassing Source Sampling

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 2.3. 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 clear the No Snoop and Relaxed Ordering bits in the Requester Attributes.

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 Appendix C 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 2.1.2.4.1, "Clock Specification," in PCI-X EM 2.0, 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

4. Arbitration

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

4.1. Arbitration Parking and Bus Turn-Around

As in conventional PCI, the bus is said to be parked when the arbiter asserts GNT# to a device that is not asserting its REQ# (except for certain cases of the bus turn-around alert, e.g., when exiting the interface low-power state in PCI-X Mode 2 as described in Section 4.1.2.2). If a 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 arbiter must park the bus on a device that is capable of being an initiator. Target-only Mode 1 devices that do not use Split Transactions are not required to implement the REQ# and GNT# pins or to be able to drive all the bus signals. Target-only Mode 2 devices that do not use Split Transactions are not required to implement the REQ# pin or to be able to drive all the bus signals, but must implement a GNT# pin for the bus turn-around alert. The arbiter is permitted to assume that the device is capable of being an initiator if the device ever asserts its REQ# pin.

4.1.1. PCI-X Mode 1 Arbitration Parking

As in conventional PCI, if no initiators request the bus, the arbiter is permitted to park the bus in PCI-X Mode 1 at any initiator that is capable of being an 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 or ECC[6::0] 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.

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 4-1.

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

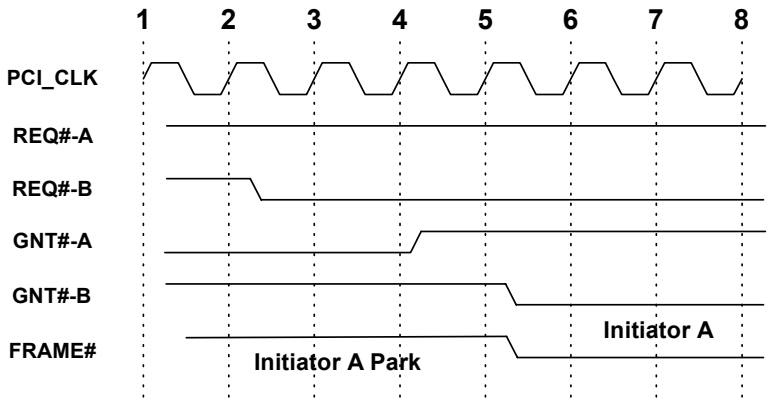

Figure 4-1: Initiating a Transaction While the Bus Is Parked, Mode 1

4.1.2. PCI-X Mode 2 Bus Drive and Turn-Around

The electrical requirements of PCI-X Mode 2 specified in Section 2.1.3, "1.5V Signaling Environment," in PCI-X EM 2.0 require that input receivers for all Category 1 signals be disabled when the bus is not being driven. The lower bus interface (AD[31::00], C/BE[3::0]#, and ECC[6::0]) must be driven at all times, except during precisely defined bus turn-around clocks. The input receivers for the upper buses (AD[63::32], C/BE[7::4]#, and ECC[7]) are disabled except during 64-bit data phases, as described in Section 2.12.1.3. (Turn-around of the AD bus for data phases of immediate read transactions is also described in Section 2.)

In Mode 2, the arbiter always guarantees that there is exactly one owner of the bus, or that the bus is in the interface low-power state. A device whose GNT# signal is asserted after a turn-around alert and has gone through the turn-around cycle described below is the present bus owner. The present bus owner is required to drive the lower bus interface (AD[31::00], C/BE[3::0]#, and ECC[6::0]) whether the device is asserting its REQ# signal and has a transaction to initiate or not. The device stops driving these signals when its GNT# signal is deasserted after a turn-around alert and the turn-around cycle begins, as described below.

The signaling between the arbiter and the device operating in Mode 2 is defined to support an arbitrary number of devices on the bus, even though the electrical requirements of PCI-X Mode 2 only support point-to-point connections. During a bus turn-around alert, GNT# asserts for one clock (during the bus turn-around alert) and then deasserts for a device that is neither the old bus owner nor the new bus owner.

4.1.2.1. Bus Turn-Around Alert and Bus Turn-Around

To turn the bus around in PCI-X Mode 2, the arbiter deasserts GNT# to the present bus owner and asserts GNT# to all other devices simultaneously. The first clock GNT# is deasserted for the present bus owner and the first clock GNT# is asserted for a device that is not the present bus owner is the bus turn-around alert. (This corresponds to the clock in Mode 1 in which no GNT# is asserted.) The device whose GNT# is asserted the clock after the turn-around alert is the new bus owner. (After the turn-around alert, the meaning of GNT# is identical between Mode 1 and Mode 2.) The new bus owner begins driving the bus as soon as the turn-around cycle is complete, as described below.

There is no requirement for the arbiter to change bus owners after a bus turn-around alert. The arbiter is permitted to return the bus to the same device that owned the bus prior to the bus turn-around alert. (This case could occur, for example, if the bus was parked at Device A, lower-priority Device B asserted its REQ#, and then one clock later higher-priority Device A asserted its REQ#.)

If the bus is idle in the clock after the turn-around alert, the bus turns around on the second clock after the turn-around alert. The previous bus owner stops driving the bus during the bus turn-around, and the new bus owner starts driving the bus the clock after the bus turn-around. All devices disable their receivers for Category 1 signals during the bus turn-around.

If the bus is not idle in the clock after the turn-around alert, the bus turns around at the end of the transaction. The second clock after the end of every transaction (the clock following the first clock in which both FRAME# and IRDY# are deasserted) is reserved for a bus turn-around, whether there is an actual change of bus ownership or not. Devices treat the turn-around at the end of a transaction the same as a turn-around on an idle bus. That is, the previous bus owner stops driving the bus during the turn-around and the new bus owner (or the same if there was no actual change of bus ownership) begins driving the bus the clock after the turn-around. All devices disable their input receivers during the turn-around.

Multiple bus turn-around alerts are permitted during a single transaction. In this case, there is still only a single bus turn-around cycle, and it occurs in the same clock at the end of the transaction. If the second alert occurs on clock N that is the first idle clock at the end of the transaction, the bus turn-around for the first change of ownership corresponds to clock N+1 of the second bus turn-around alert. In this case, the bus turns around twice in a row (clock N+1 and N+2, see Figure 4-5).

The interface low-power state described in Section 4.1.2.2 introduces some restrictions on how frequently the arbiter is permitted to change bus owners. In PCI-X Mode 2, if a bus turn-around alert occurs in clock N, the next bus turn-around alert must occur no earlier than clock N+3, unless the arbiter is placing the devices in the interface low-power state. Therefore, GNT# is always asserted for a new bus owner (that is, the case in which GNT# remains asserted after the turn-around alert) for a minimum of three clocks (including the turn-around alert), and GNT# is always deasserted for a minimum of two clocks, except when entering the interface low-power state.

Figure 4-2 illustrates a bus turn-around alert and turn-around cycle on an idle bus. In this example, two devices are sharing the bus. The states of the I/O buffers of each device are shown under the shared signals.

D = Drive: Output on, receiver disabled, terminator off.

R = Receive: Output off, receiver enabled, terminator on.

B = Blind: Output off, receiver disabled, terminator on.

L = Low power: Output off, receiver disabled, terminator off.

When the figure begins, no device is requesting the bus and the bus is parked at Initiator A. The arbiter decides to change the parked owner to Initiator B, and signals a bus turn-around alert in clock 3. Since the bus is idle in clock 3, the turn-around occurs in clock 5. Initiator A stops driving the bus in clock 5, and Initiator B begins driving the bus in clock 6. All devices disable their input receivers during the turn-around in clock 5.

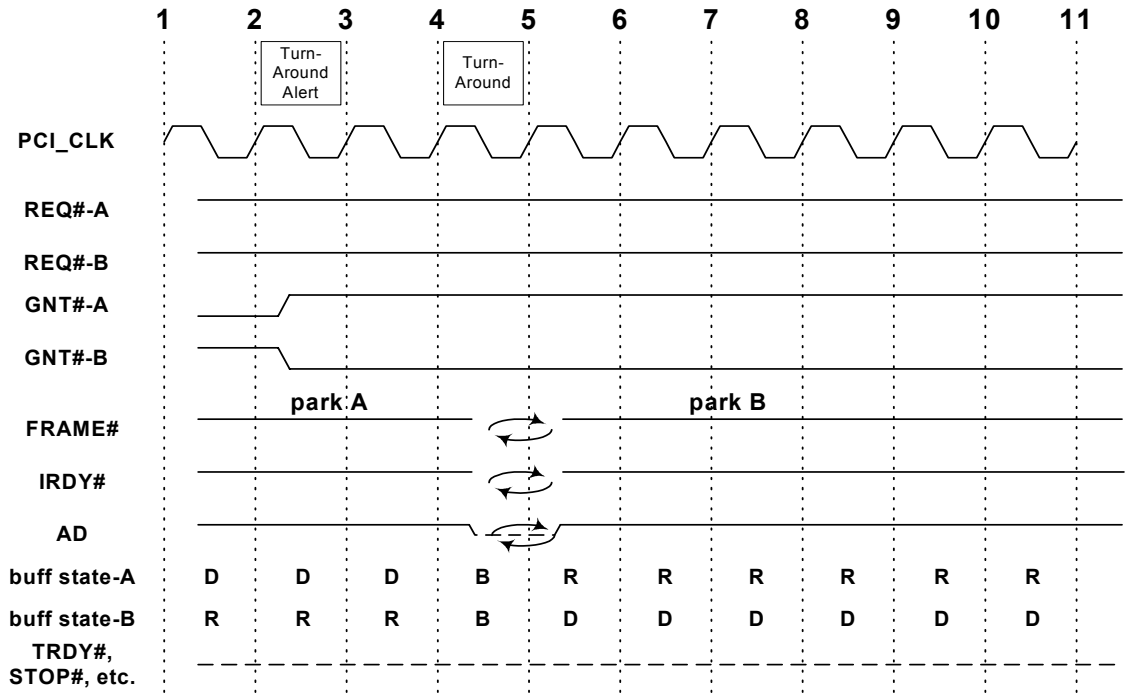

Figure 4-2: Bus Turn-Around While Idle (Mode 2)

Figure 4-3 through Figure 4-6 illustrate a series of cases in which two devices share a bus. Each case begins with Device A being the parked owner (REQ#-A deasserted) and driving a transaction when Device B asserts REQ#-B to request the bus. Then Device A (the higher priority device) asserts REQ#-A to request the bus. In each case, the turn-around alert to give the bus to Device B occurs in clock 4, while the bus is busy with the transaction from Device A. In the first figure Device A's request for the bus also occurs in clock 4. As the series of figures progress, Device A's request occurs one clock later in each figure.

In Figure 4-3, Device A requests the bus in clock 4. Generally the arbiter would be able to signal a turn-around two clocks after REQ# was asserted (clock 6). However, in this case, that would cause the pattern on GNT# to put the interfaces in the low-power state, as described in Section 4.1.2.2, so the second turn-around alert must be delayed by the arbiter. In this example, the second turn-around alert occurs on clock 7. Since Device A owns the

bus when it goes idle at the end of the transaction (clock 8), it drives the bus in clock 10 after the turn-around.

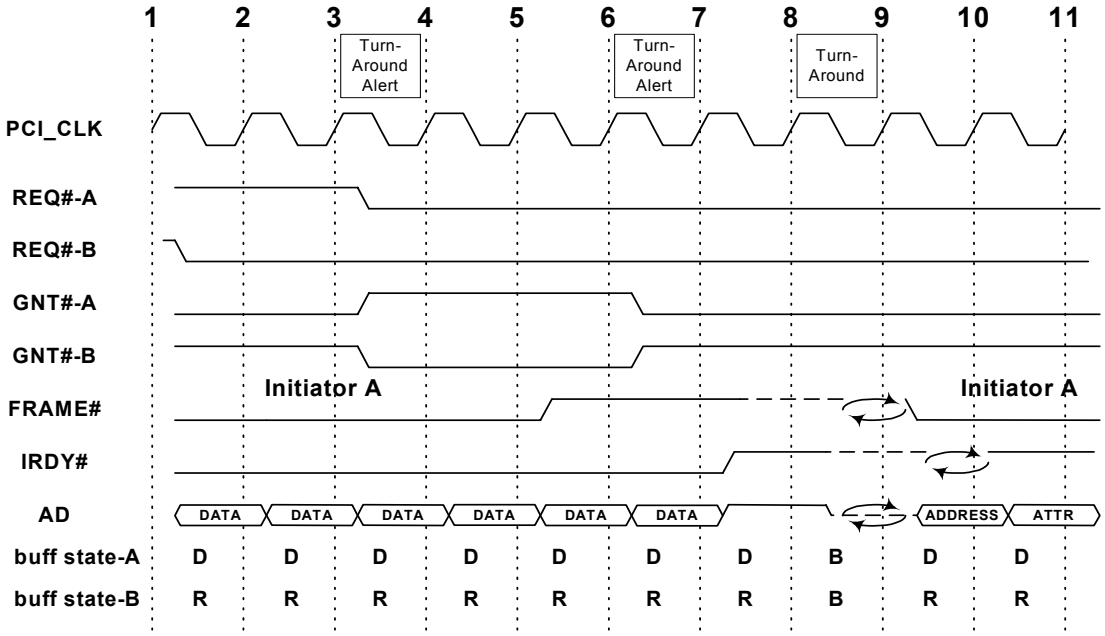

Figure 4-3: Bus Turn-Around on Busy Bus, Case 1 (Mode 2)

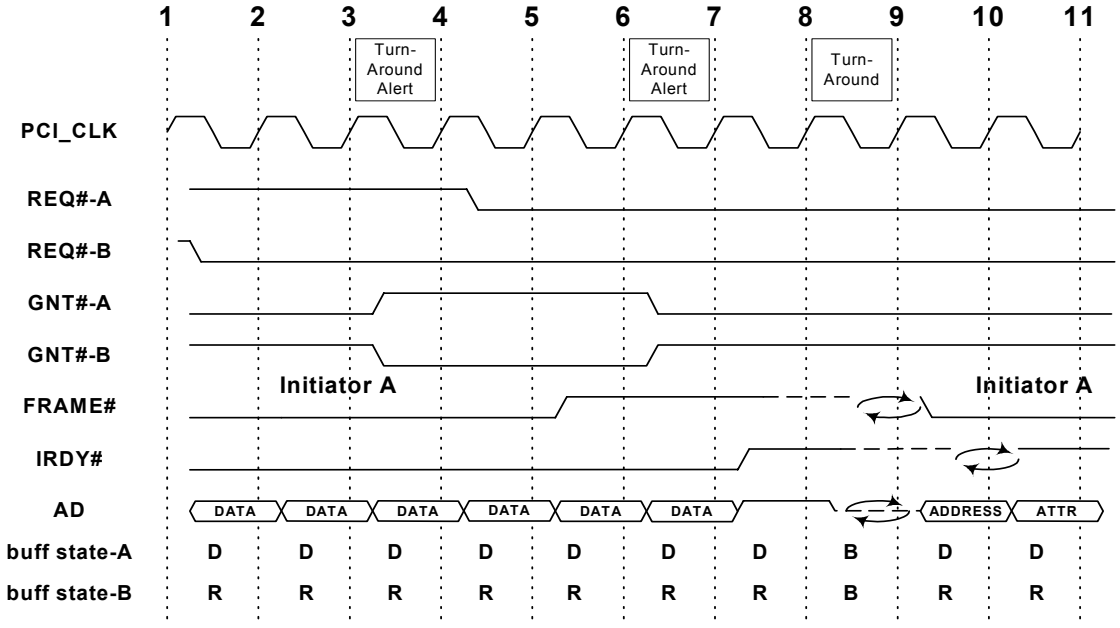

Figure 4-4: Bus Turn-Around on Busy Bus, Case 2 (Mode 2)

In Figure 4-4, the request from Device A is delayed by one clock (clock 5). The arbiter signals a bus turn-around alert as soon as possible (clock 7), and the bus turns around as in the previous case.

In Figure 4-5, the request from Device A is delayed on more clock (clock 6), and again the arbiter signals a bus turn-around alert as soon as possible (clock 8). However, in this case, the transaction from Device A (while it was the parked owner) has already ended when the turn-around alert is signaled, so the bus turns around twice, once (clock 9) two clock after the end of the transaction and once (clock 10) two clocks after the turn-around alert while the bus was idle. Since Device A owns the bus for the second turn-around, it drives the bus at the end of the turn-around.

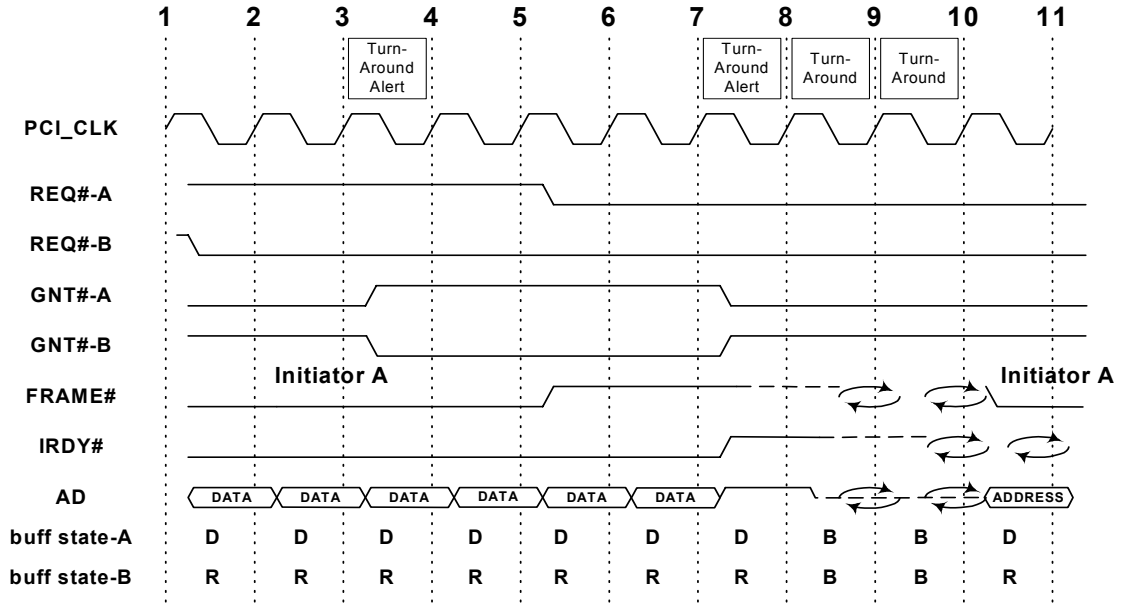

Figure 4-5: Bus Turn-Around on Busy Bus, Case 3 (Mode 2)

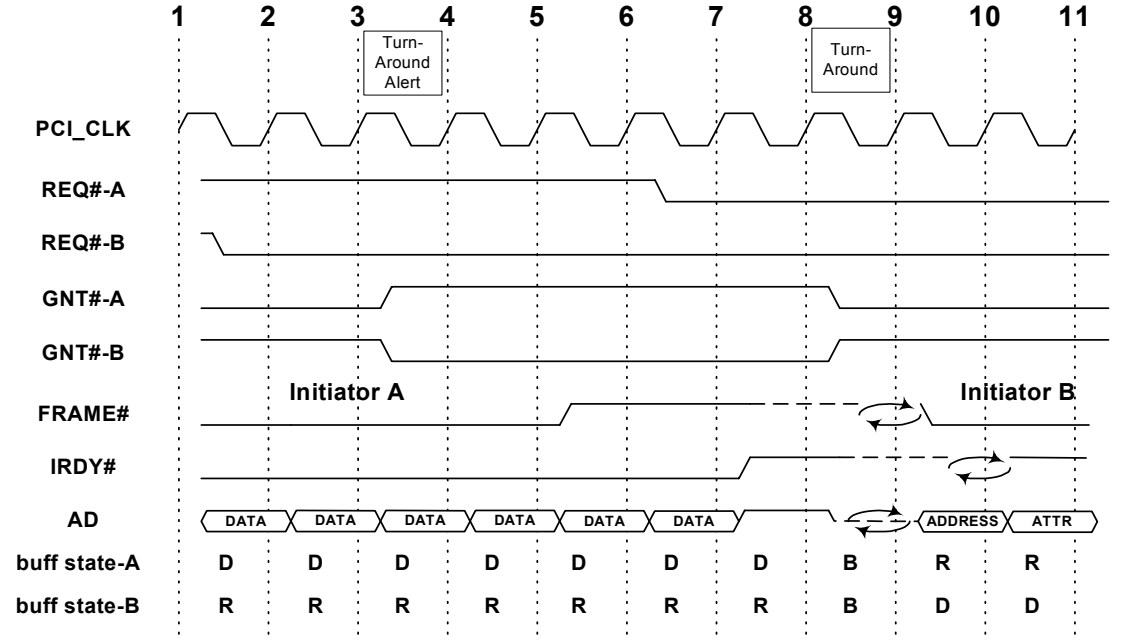

Figure 4-6: Bus Turn-Around on Busy Bus, Case 4 (Mode 2)

In Figure 4-6, the request from Device A is delayed one more clock (clock 7), and the arbiter signals bus turn-around alert as soon as possible (clock 9). In this case, the bus is already turning around for Device B in clock 9, so Device B begins driving the bus before the next transaction from Device A.

4.1.2.2. Interface Low-Power State

In the interface low-power state, all devices on the bus disable the output drivers and electrical terminators for all their Category 1 signals (see Section 2.1.1, "Driver and Receiver Categories," in PCI-X EM 2.0).

The arbiter is permitted to place all device interfaces in the interface low-power state at any time that no device is requesting the bus and the bus is idle. The arbiter is permitted to awaken the devices from the interface low-power state at a minimum of three clocks after placing the devices in interface low-power state as described below.

If no device is requesting the bus and the bus is idle in clock N-2, the arbiter places all devices in the interface low-power state by signaling a bus turn-around alert on clock N and then signaling another bus turn-around alert on clock N+2. After the second alert, the arbiter deasserts all GNT# signals. If the bus is idle on clock N+1, all devices go to the interface low-power state on clock N+4. Note that the present bus owner recognizes the bus turn-around alert as a change in GNT# from asserted to deasserted and all other devices recognize the bus turn-around alert as a change in GNT# from deasserted to asserted.

If the parked bus owner begins a transaction (asserts FRAME#) after clock N-2, the bus is not idle in clock N+1. In this case, the devices enter the interface low-power state two clocks after FRAME# and IRDY# are deasserted at the end of the transaction, unless the arbiter signals bus turn-around alert again to grant the bus to a device.

To awaken the devices from the interface low-power state, the arbiter signals a bus turn-around alert and leaves GNT# asserted to the new bus owner. If the bus turn-around alert to place the devices in interface low-power state occurred on clocks N and N+2, the arbiter is permitted to signal a bus turn-around alert to awaken the devices no sooner than clock N+5.

Figure 4-7 illustrates entering the interface low-power state while the bus is idle. The arbiter signals the first bus turn-around alert in clock 3 and the second turn-around alert in clock 5. Since the bus is idle in clock 4, all devices enter the interface low-power state as a result of the second turn-around alert. The arbiter signals one more turn-around alert in clock 8 to grant the bus to Device B. Note that clock 8 is the earliest that the arbiter is permitted to signal to the devices to leave the interface low-power state after entering it in clock 5.

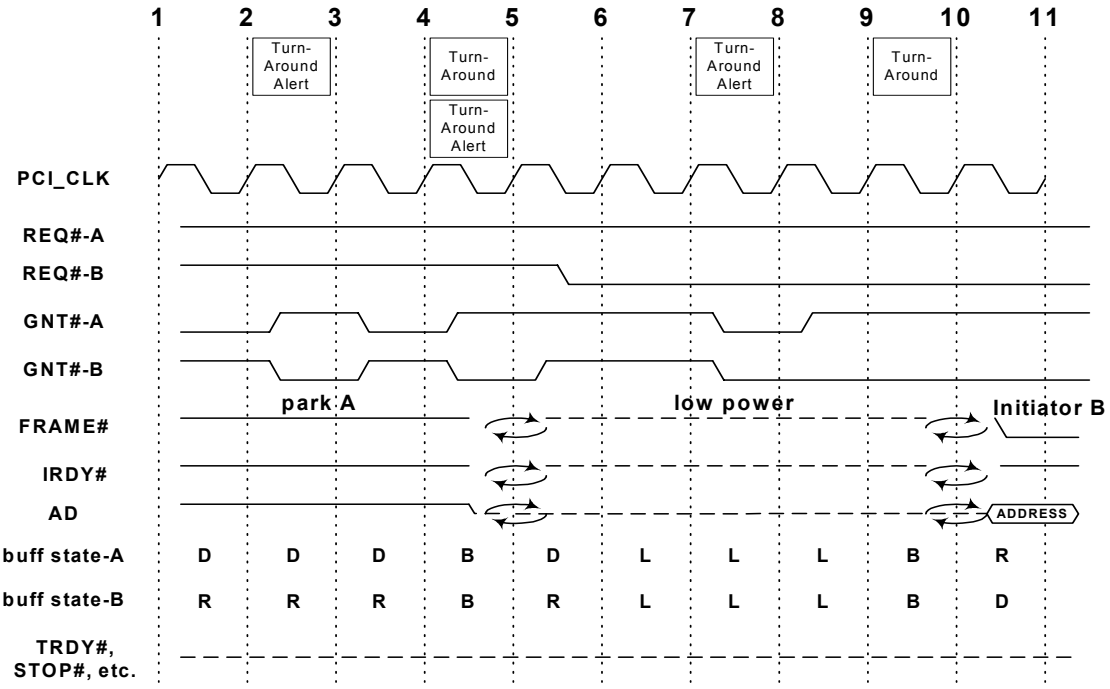

Figure 4-7: Interface Low-Power State (Mode 2)

Figure 4-8 illustrates the case in which the parked bus owner starts a transaction as the arbiter is putting the devices in the interface low-power state. In this example, Device A asserts FRAME# one clock after the first bus turn-around alert, which is the latest possible time Device A can assert FRAME# relative to a bus turn-around alert. Since the bus is not idle in clock 4, the device enters the interface low-power state after the turn-around at the end of the transaction.

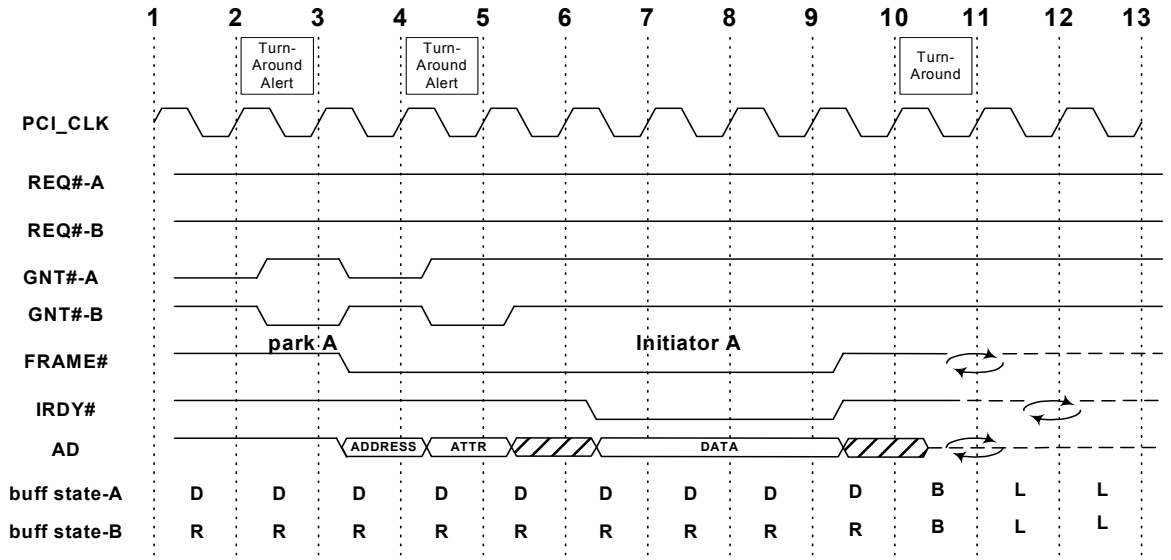

Figure 4-8: Entering Low-Power State Blocked by New Transaction, Case 1 (Mode 2)

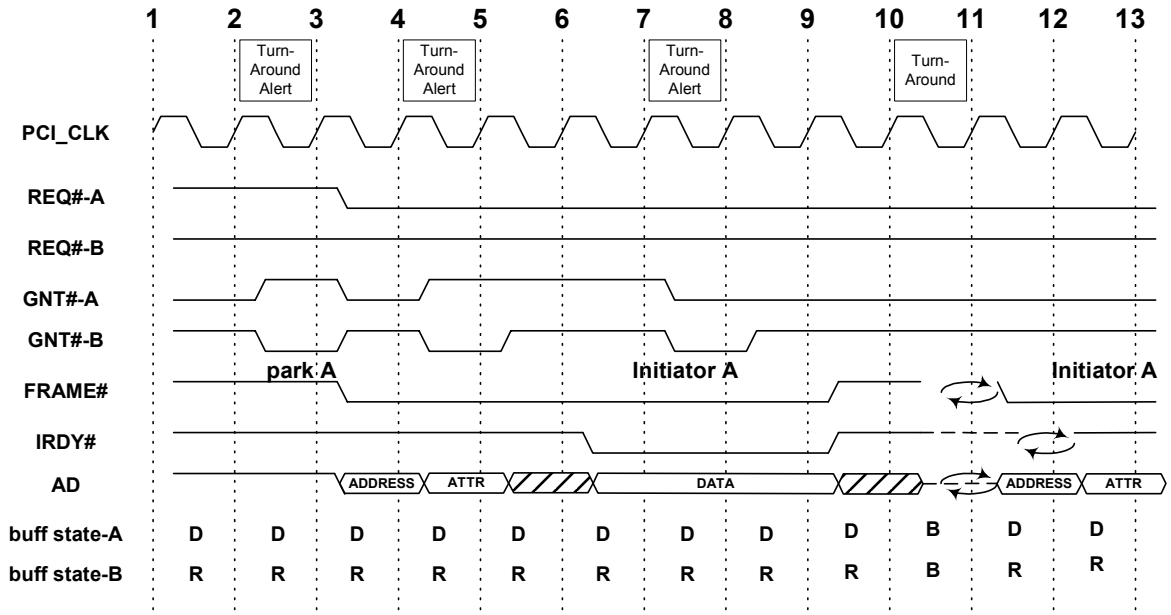

Figure 4-9: Entering Low-Power State Blocked by New Transaction, Case 2 (Mode 2)

Figure 4-9 illustrates another example in which the parked bus owner starts a transaction as the arbiter is putting the devices in the interface low-power state. As in the previous example, Device A asserts FRAME# one clock after the first bus turn-around alert, which is the latest possible time Device A can assert FRAME# relative to a bus turn-around alert. However, in this example, Device A asserts its REQ#, indicating it has more transactions to execute. Since the second bus turn-around alert occurred on clock 5, the arbiter is not

permitted to signal turn-around alert again until clock 8. When the bus turns around at the end of the transaction, GNT#-A is asserted, so the devices do not enter the interface low-power state and Device A continues driving the bus.

Figure 4-10 illustrates another example in which the parked bus owner starts a transaction as the arbiter is putting the devices in the interface low-power state. In this example, Device A asserts FRAME# one clock before the first bus turn-around alert, which is the earliest possible time the device could assert FRAME# before the arbiter attempts to place the devices in the interface low-power state. As in the previous examples, the devices must delay entering interface low-power state until the transaction completes. However, in this example, Device B asserts its REQ# shortly before the transaction ends. Since the second bus turn-around alert occurred on clock 5, the arbiter is not permitted to signal turn-around alert again until clock 8, which leads to two bus turn-around cycle in a row, one on clock 9 at the end of the transaction and one on clock 10 that grants the bus to Device B (and prevents it from entering the interface low-power state. As can be seen from this example, if Device B delayed its request for the bus, it is possible for the devices to stay in the interface low-power state for any number of clocks from 1 to infinity.

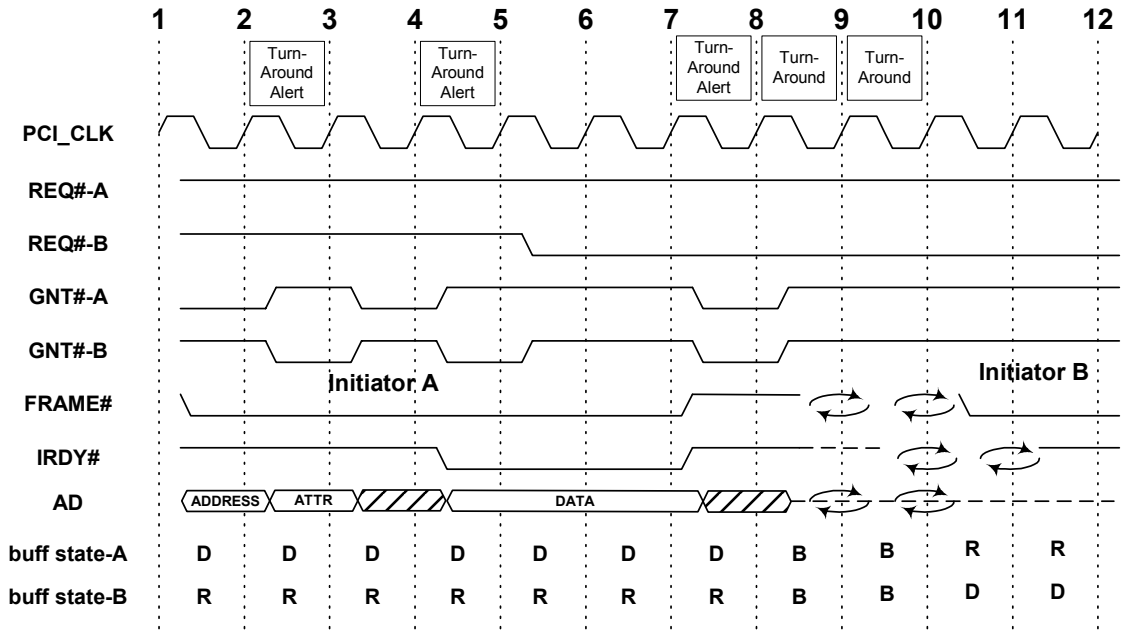

Figure 4-10: Entering Low-Power State Blocked by New Transaction, Case 3 (Mode 2)

IMPLEMENTATION NOTE

Control of Interface Low-Power State in the Arbiter

Although implementation of the interface low-power state is optional, its use can cause a significant reduction in the average power dissipation of a Mode 2 system. The electrical

drive and termination requirements of Category 1 signals dissipate a significant amount of power even when the signals are not switching.

Some Mode 2 systems will also find it useful to disable use of the interface low-power state.

- Systems where maximum performance is required and power consumption is not an issue.
- For system debug purposes.

Such systems would implement an option in the bus arbiter to prevent it from ever putting the devices on a bus in interface low-power state. The rest of the Mode 2 arbiter behavior would remain unchanged, including the signaling of bus turn-around alert to take the devices out of interface low-power state after the rising edge of RST# and to change bus owners. As with most other arbiter features, the method by which the arbiter is programmed to disable interface low-power state is not specified, however, pin strapping and control bits in device-specific Configuration Space in the source bridge would be two alternatives.

4.1.2.3. Bus State Initialization

When RST# is asserted, all devices on the bus enter the interface low-power state. That is, all Category 1 output drivers and input terminators are disabled, and no GNT# signals are asserted. To assert GNT# to any device (either to run a transaction or to park), the arbiter signals bus turn-around alert to awaken the devices as described in Section 4.1.2.2. The arbiter is not permitted to signal the first bus turn-around alert until T_{rhfa} (as specified in Table 2-7, "3.3V General Timing Parameters," in PCI-X EM 2.0) after the rising edge or RST#, unless the arbiter knows that all devices on the bus are ready before then (e.g., the path to the boot ROM).

4.2. 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 2.3.
- 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 2.3 are not permitted in PCI-X mode. In PCI-X Mode 1, the minimum number of idle clocks between any two transactions is either one or two, depending on the way the earlier transaction ends, as described below. In PCI-X Mode 2, there is always a minimum of two idle clocks between any two transactions.
- A special handshake is defined in PCI-X Mode 2 to allow the arbiter to put the device interfaces in a low power state (see Section 4.1.2.2).

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.

4.2.1. 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 on a 64- or 32-bit bus, 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.

In PCI-X Mode 1, 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.

In PCI-X Mode 2, GNT# asserts one clock earlier than in Mode 1 as a bus turn-around alert, described in Section 4.1.2.1. An initiator acquiring the bus is permitted to start a transaction in any clock N if the initiator's GNT# was asserted on clock N-3 (for the alert) and clock N-2 and the bus was idle (FRAME# and IRDY# are both deasserted) on clock N-2. There is always a minimum of two idle clocks between transactions in Mode 2 (from the same or different initiators.)

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, both in Mode 1 and Mode 2.

In PCI-X Mode 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. In PCI-X Mode 2, if an initiator has more than one transaction to execute, it is permitted to start the second transaction if GNT# is asserted and the bus is idle. There is always a minimum of two idle clocks between transactions in Mode 2 (from the same or different initiators.)

All fast back-to-back transactions as defined in PCI 2.3 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.

In PCI-X Mode 2, if the arbiter signals a bus turn-around alert and two clocks later signals another one, the device places its interface in the low-power state, as described in Section 4.1.2.2.

4.2.2. Arbiter Requirements

In PCI-X Mode 1, if no GNT# signals are asserted, the arbiter is permitted to assert any single GNT# (to any device capable of being an initiator) on any clock. In PCI-X Mode 2, the arbiter always guarantees that there is exactly one owner of the bus or that the bus is in the interface low-power state, and change of bus ownership requires a bus turn-around alert as described in Section 4.1.2.1.

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 (in Mode 1) or six clocks (in Mode 2) while the bus is idle for the device to initiate a configuration transaction on a 64- or 32-bit bus. GNT# is permitted to be asserted for fewer 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 (for Mode 1) or N-1 (for Mode 2).
- GNT# remains asserted through clock N+5.
- The device keeps FRAME# deasserted and REQ# asserted through clock N+6.

In PCI-X Mode 1, 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.)

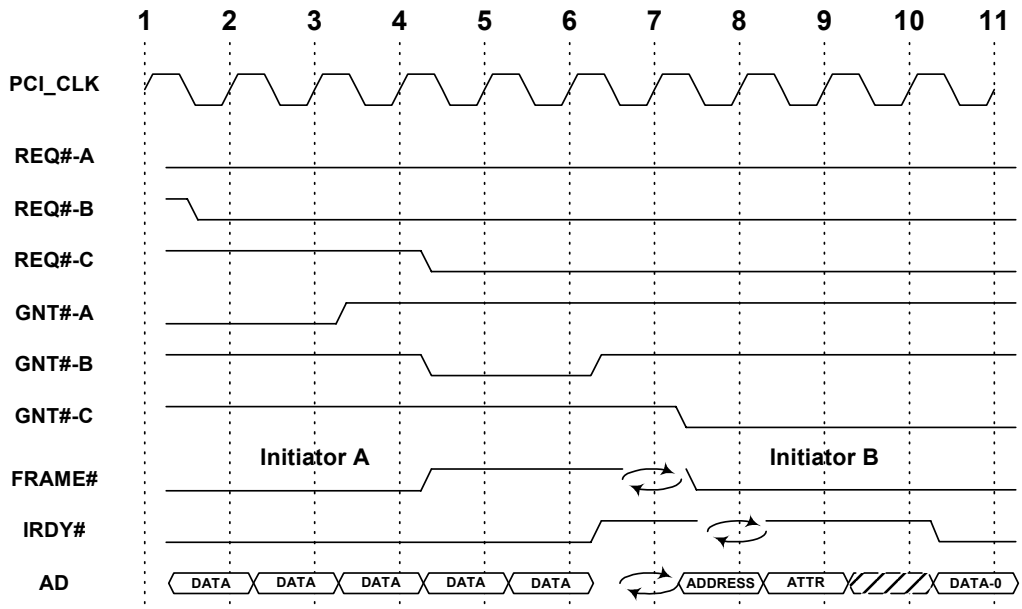

Figure 4-11: Arbitration Example, Mode 1

Figure 4-11 illustrates an arbitration example switching between several initiators in PCI-X Mode 1. 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 4-11 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.

In PCI-X Mode 2, when the arbiter deasserts GNT# to a device, it must signal bus turn-around alert, as described in Section 4.1.2.1.

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.

4.3. Arbiter Coordination with the PCI Hot-Plug Controller

PCI HP 1.1 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

5. Error Functions

5.1. Bus Error Protection

The PCI-X definition provides for parity protection of all transaction phases in one mode of operation and ECC protection of all transaction phases in another mode of operation. When the bus is operating with parity protection, the bus is said to be operating in parity mode. When the bus is operating with ECC protection, the bus is said to be operating in ECC mode.

Both Mode 1 and Mode 2 PCI-X devices are required to support parity protection. Mode 1 devices optionally support ECC protection. Mode 2 devices are required to support ECC protection when operating in Mode 2, and optionally support ECC when operating in Mode 1. See Section 8.7 for the requirement for PCI-X bridges that support ECC in Mode 1 to do so on both the primary and the secondary interfaces.

Devices that support ECC must implement either the version 1 or the version 2 PCI-X Capabilities List item described in Section 7.2.3 for non-bridge devices and Section 8.6.2.3 for bridge devices. Devices that support ECC in Mode 2 but not in Mode 1 use version 1. Devices that support ECC both in Mode 2 and Mode 1 use version 2.

In PCI-X Mode 1, the error protection mode is initialized by the PCI-X initialization pattern (see Table 6-2). (The system initializes a bus in Mode 1 and ECC mode only if all devices on the bus support ECC. See Section 6.2.2.) In some system (e.g., embedded applications), the system initializes in Mode 1 and ECC mode. In other systems, configuration software selects between parity and ECC modes when operating in PCI-X Mode 1.

In parity mode, a parity error is an uncorrectable error. In ECC mode, all single-bit errors are corrected (if error correction is enabled) and the transaction completes. In ECC mode, multiple-bit errors (if they are detected) are uncorrectable errors. Reporting methods are defined for uncorrectable errors to enable recovery at the software level.

The following PCI-X uncorrectable error detection and response functions are the same as conventional PCI:

- Uncorrectable address errors cause SERR# to be asserted (if enabled).
- Initiators and targets set bits in the Status register when an uncorrectable error occurs.
- SERR# is an open-drain signal. Whenever it is asserted, it is actively driven low synchronously with CLK for one clock period. It is pulled up by a resistor supplied by the system board, 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 PCI-X uncorrectable error detection and response functions are different from conventional PCI:

- Uncorrectable attribute errors cause SERR# to be asserted (if enabled).
- No parity or ECC is generated or checked for the target response phase.
- The target of a burst push transaction must not check data-phase parity or ECC while it is inserting target initial wait states. It must check data-phase parity or ECC only on the data-transfer clock (single common-clock or (in Mode 2) multiple source-synchronous data subphases).
- PERR# is asserted as described in Section 5.2.1 after an uncorrectable error in a common-clock data phase two clocks after the clock that latches the parity or ECC check bits, one clock later than conventional PCI. (See Section 5.1.2.4 for requirements for errors in source-synchronous data phases.)
- The requester sets the Master Data Parity Error bit to record an uncorrectable data error. For Split Completions, this is the target rather than the initiator. See Sections 5.2.1 and 5.2.6.
- The Uncorrectable Data Error Recover Enable bit in the PCI-X Command register enables the system to recover from some uncorrectable data errors. See Sections 5.2.1.1 and 5.2.1.2.

5.1.1. Parity Mode

The following PCI-X parity-mode error detection and response functions are the same as conventional PCI (see Section 5.1.2 for the analogous ECC-mode features). (Some 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 all burst push 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.

The following item is different for PCI-X parity mode than for conventional PCI (see Section 5.1.2 for the analogous ECC-mode feature):

- 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 following sections provide additional details about parity generation and checking in parity mode.

5.1.1.1. 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.

5.1.1.2. 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 2.3 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.

5.1.1.3. 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 burst Memory Write transaction. Other burst push transactions would be the same, except the C/BE# bus would be reserved and driven high during the data phases. Figure 5-2 illustrates parity generation and checking on a read transaction. For the Memory Write transaction in Figure 5-1, the initiator drives PAR64 (for a 64-bit device) and PAR on clock 4 for the address phase and on clock 5 for the attribute phase. 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 Memory Write 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 Memory Write 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.)

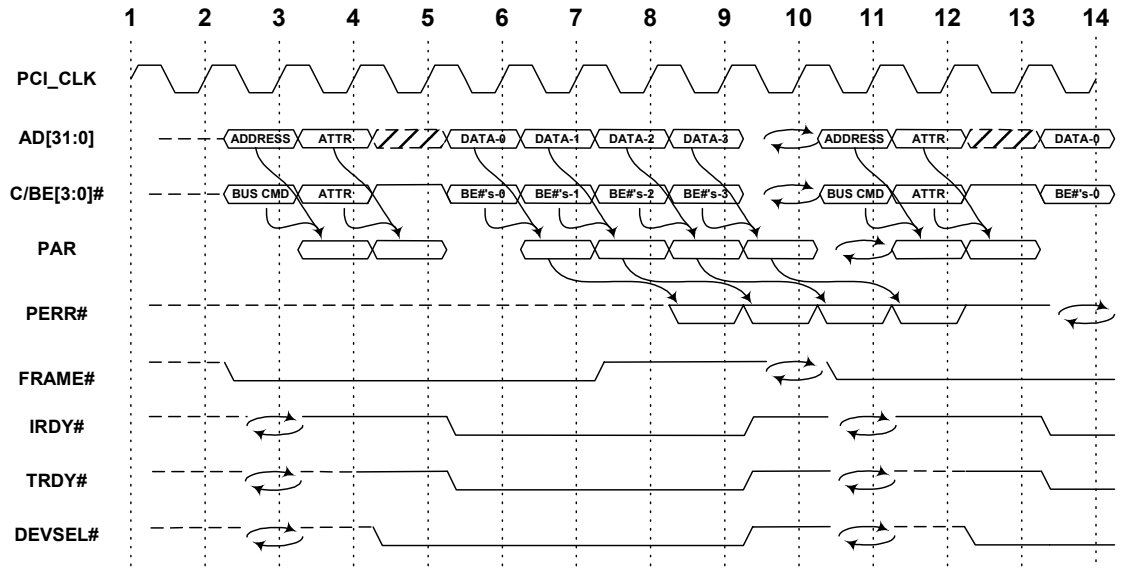

Figure 5-1: Burst Memory Write Transaction Parity Operation

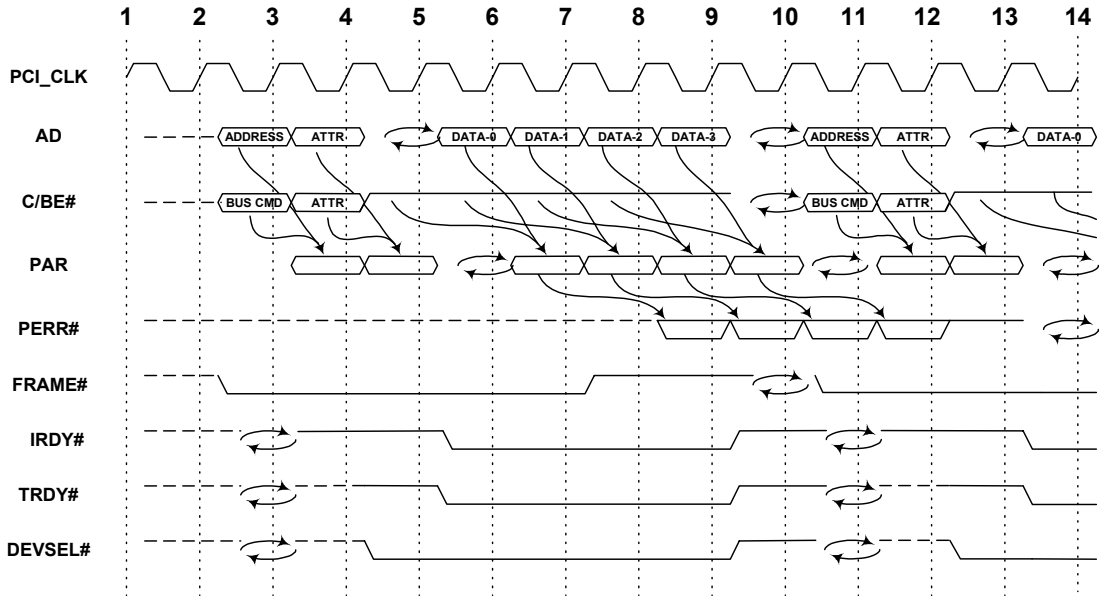

Figure 5-2: Burst Read Transaction Parity Operation

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 Memory Write 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 5-3 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.

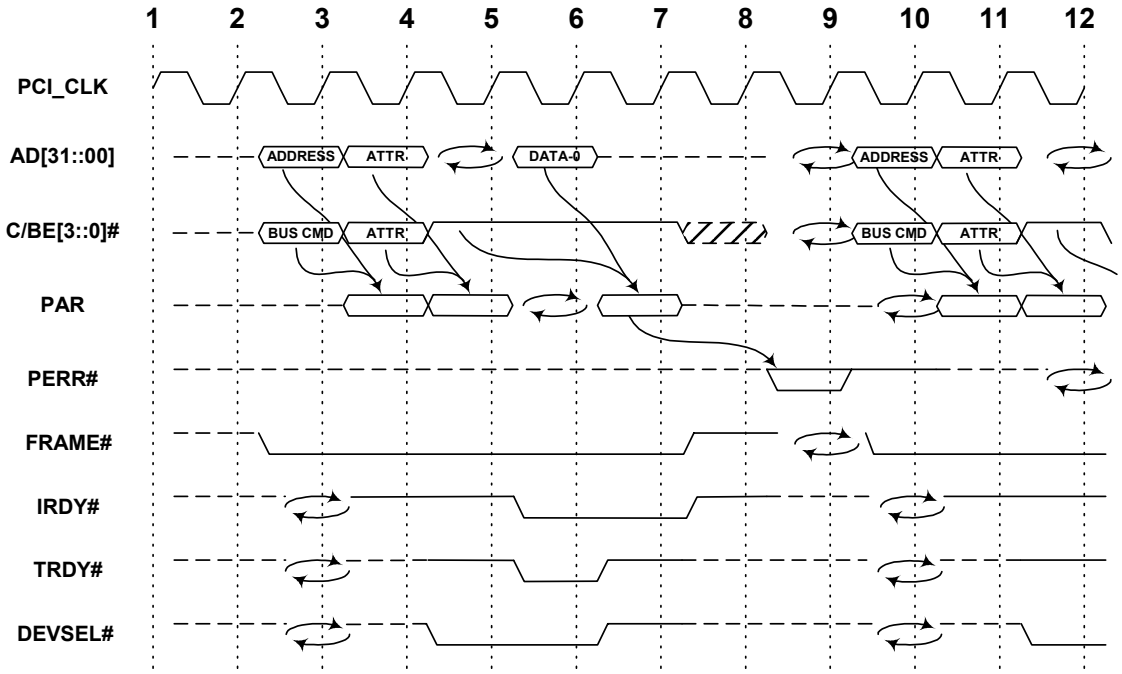

Figure 5-3: DWORD Read Parity Operation, Decode A and No Initial Wait States

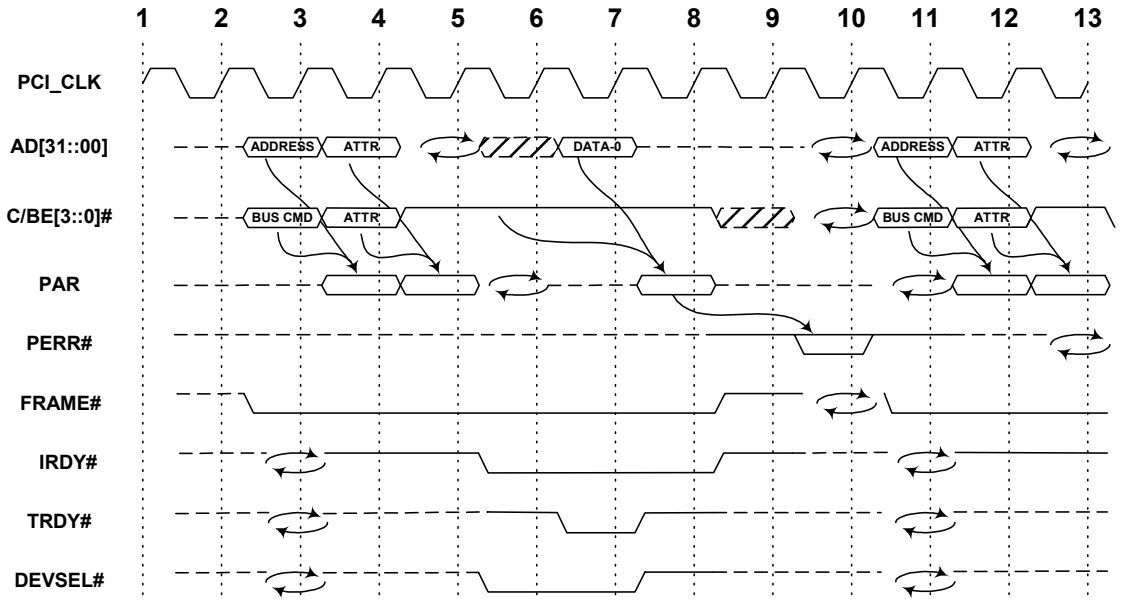

Figure 5-4: DWORD Read Parity Operation, Decode B and No Initial Wait States

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 push transaction, 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.

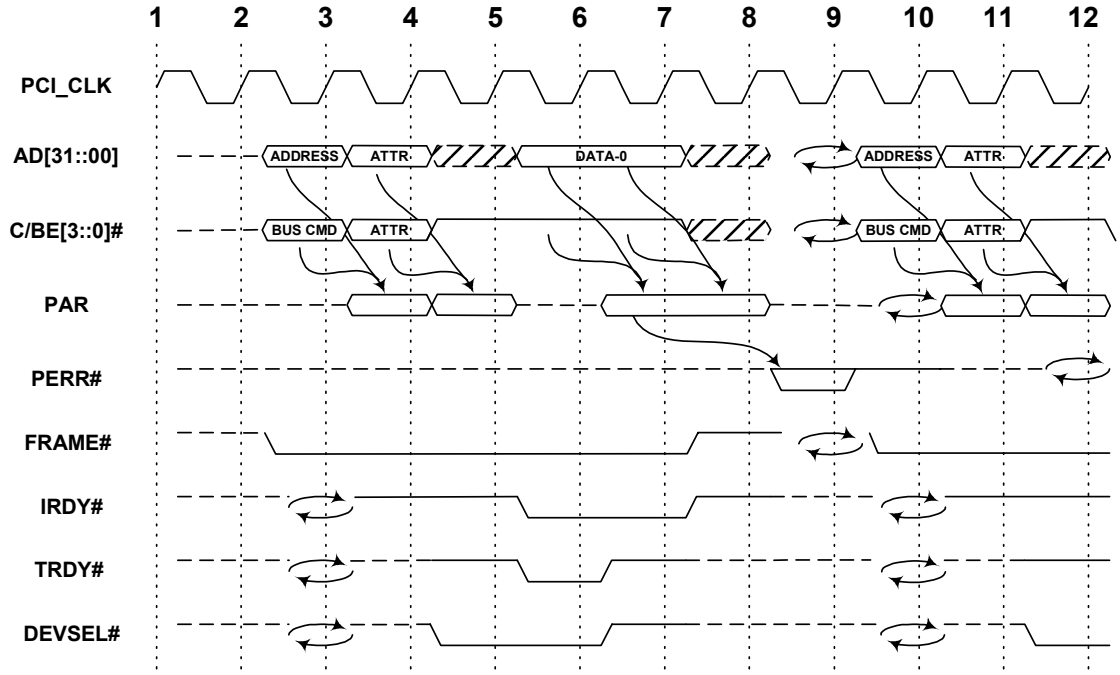

Figure 5-5: DWORD Write Parity Operation, Decode A and No Initial Wait States

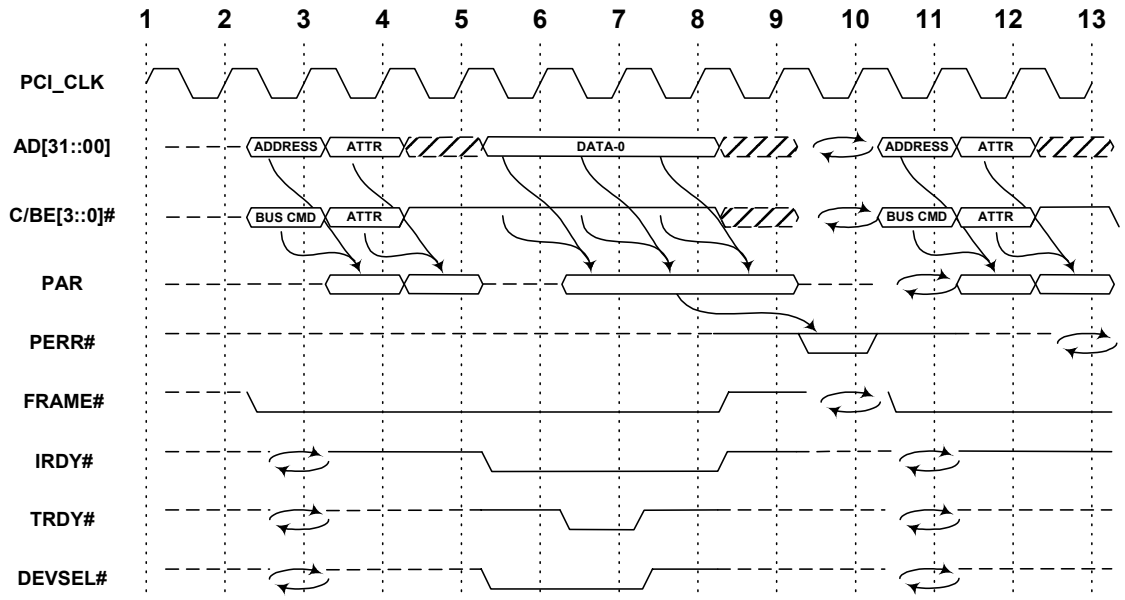

Figure 5-6: DWORD Write Parity Operation, Decode B and No Initial Wait States

5.1.2. ECC Mode

In ECC mode, the ECC check bits are driven on the ECC bus (which in some cases shares pins with other signals). This discussion generally refers to the ECC check bits as if they used dedicated pins on the PCI interface. Refer to Section 2.4, "Mode 2 Pin Sharing," in PCI-X EM 2.0 for a complete description of pin sharing.

The following items are different for ECC mode than for PCI-X parity mode or conventional PCI.

- All devices are required to generate ECC signatures for information they drive, including address, attribute, common-clock data phases, and source-synchronous data subphases. For source-synchronous data subphases, the initiator is required to generate ECC signatures for all subphases in all data phases, even if the starting and ending addresses are not aligned to a data phase boundary.
- All devices are required to verify the ECC signature bits for the following information they receive:
  - Address phases
  - Attribute phases
  - Common-clock data phases
  - Source-synchronous data subphases between the starting and ending addresses, inclusive.

  The target of a source-synchronous transaction optionally verifies ECC signature bits for data subphases before the starting address and after the ending address.

- A single-bit-correcting, double-bit-detecting ECC is used, as described in Section 5.1.2.2.
- The number of check bits and the amount of information they protect varies according to the type and width of transaction.
  - The C/BE# bus is included in ECC calculations during address and attribute phases. The C/BE# bus during data phases is included in ECC calculations only for Memory Write transactions. In data phases for all other transactions, ECC signatures are generated as if the C/BE# bits were all zero, and the actual values on the C/BE# bus are ignored during checking.
  - A seven-bit ECC signature is used for address and attribute phases and for 32-bit common-clock and source-synchronous data phases, and protects AD[31::00] and (for Memory Write transactions) C/BE[3::0]#.
  - For 16-bit transactions, the seven-bit ECC signature is extended by an extra guard bit, resulting in an eight-bit wide extended ECC signature. (The "7+1" extended signature is not the same as the eight-bit ECC signature used with 64-bit data phases.) The extra guard bit provides additional protection against mis-correction cases, but it does not enhance the correction capability provided by the ECC.
  - An eight-bit ECC signature is used for 64-bit common-clock and source-synchronous data phases, and protects AD[63::00] and (for Memory Write transactions) C/BE[7::0]#. The eight-bit ECC is an extension of the seven-bit ECC and the same ECC signatures are used for AD[31::00] and C/BE[3::0]#.
- The device driving the AD bus also drives the ECC bus as follows:
  - ECC[6::0] for all address, attribute, and 32-bit common-clock and source-synchronous data phases.
  - Four ECC signals described in Section 2.12.2 for all 16-bit address, attribute, and common-clock and source-synchronous data phases.
  - ECC[7::0] for all 64-bit common-clock and source-synchronous data phases.
- Common-clock ECC signature bits lag the information they protect by one clock (like PAR and PAR64).
- Source-synchronous ECC signature bits are coincident with the information they protect. If the target detects an uncorrectable error for source-synchronous data subphases that the initiator drove between clocks N and N+1, the target asserts PERR# (if enabled) on clock N+5.

5.1.2.1. ECC Basics and Definitions

An ECC "signature" is a (specific) value associated with or carried on the ECC check bits or a value used in the construction of a value carried on the ECC check bits. An ECC signature is often constructed by combining (using XOR, symbolized by "⊕") a set of signatures. The usage of the general term varies slightly by context, as follows:

- The ECC signature of a single bit or bit position is the value of the check bits associated with that bit position, as given in the tables below. ECC signatures are also specified for other information associated with a data value, e.g., the phase within the transaction or a forwarded error. These ECC signatures are treated very much as if they were bit-signatures for "hidden" data bits.
- The ECC signature of a data value is the value of the check bits obtained by combining (using XOR) the individual bit-position ECC signatures for each bit that has a value of 1 within the data value.
- The ECC signature of a group or set of bits (bit positions) is the combination (using XOR) of the individual ECC bit-position signatures of the bits in the group. It describes the effect (using XOR) of those bits, as a group, on the ECC check bits or signature, i.e., change in the signature due to an inversion of those bits or the contribution of all logic 1's in those bits to the calculation of a data value signature.
- The ECC signature for a phase (address, attribute, or data) is obtained by combining the ECC signatures for the actual data value (by bit position) with the appropriate signatures specified for any associated or "hidden" information such as the phase.

ECC signatures are always "combined" using XOR. There are no other applicable operations.

The concatenation of the data value and the ECC signature for that data value is called in this discussion a "codeword" or "codeword value." Codewords are treated as a unit and the protection provided by an ECC covers both the data and check bits equally.

A "valid codeword" is the combination of a data value with the calculated or "correct" ECC signature for that data value. An "invalid codeword" is a combination of a data value with an ECC signature other than the correct signature for that data value and indicates an error that, in some cases, is correctable.

An "error" (in the context of an ECC) is the erroneous inversion of a (specific) set of codeword bits (which in various instances includes data bits, check bits, or both) in a valid codeword. Due to the use of XOR, the effects of an error are essentially independent of the underlying codeword value. The "error signature" of a group of bits (for example, the "signature of a triple-bit error") is the ECC signature of the specific group of bits in error, i.e., the combination (XOR) of the bit-position signatures of those bits.

When a codeword is validated, the expected ECC signature is calculated using the data part of the codeword and possibly other associated or "hidden" information (such as the phase) and the expected ECC signature is combined (using XOR) with the ECC signature in the actual codeword to form the "(ECC) syndrome". If the syndrome is zero, the codeword is considered to be valid. A non-zero syndrome indicates that the codeword is invalid and provides the information used to determine if the error is correctable and to correct the error if it is correctable.

If an error occurs, such as the erroneous inversion of a specific set of bits, the syndrome contains the error signature of that set of bits.

An error is "detectable" if the error signature for that specific set of bits is non-zero. If an error occurs but the error signature for that specific set of bits is zero, that error is called

"undetectable." Undetectable errors always involve an even number of bits. The failure to detect such an undetectable error is called a "mis-detection."

An error is considered to be "correctable" (or "uncorrectable") if the error signature for that set of bits matches (or does not match) the bit-position signature of some bit in the codeword. If correction is enabled, the error is corrected by re-inverting that bit. If the error consists only of an inversion of that bit-position, the correction is valid. Some multiple bit errors (always involving an odd number of bits) have error signatures that match the bit-position signature of an individual bit. If the error is corrected by inverting the indicated bit, a "mis-correction" has occurred, which has the effect of converting the original error into an undetectable error (involving one additional or one less bit).

An error involving an even number of bits is either uncorrectable or mis-detected, and is never mis-corrected. An error involving an odd number of bits is always detected and in various instances is (validly) correctable, uncorrectable, or results in a mis-correction (if corrected).

An ECC "protects" against a specific error if it neither mis-detects nor mis-corrects that error. For example, an ECC is said to protect against an error in bit-positions A, B, and Q if the error syndrome for A, B, and Q does not result in a mis-correction, or, for an even number of bits, in a mis-detection.

An ECC fully "protects" a group of bits (e.g., a nibble) if it protects against all errors involving only subsets of that group of bits.

In the following discussion, the eight individual ECC check bits are referred to using the notation E7-E0.

5.1.2.2. ECC Signature Generation

All devices are required to generate ECC signatures for all phases in which they drive the AD bus. The number of check bits used for the ECC signature and the amount of information they protect is described above.

The characteristics of the seven-bit ECC are described in Section 5.1.2.2.1. Table 5-1 provides the specific ECC signatures for the seven-bit ECC, including the additional guard bit for the extended seven-bit ECC used for 16-bit transactions. Section 5.1.2.2.2 describes the eight-bit ECC (which is actually a simple extension of the seven-bit ECC), along with certain implementation hints. Table 5-2 provides the specific signatures for the additional bits. Section 5.1.2.2.3, Table 5-3, and Table 5-4 describe the phase protection method, which applies to both the seven-bit and the eight-bit ECC protection mechanisms.

In the following discussion, when bits from the AD and C/BE# bus are included in ECC equations, this discussion always refers to the state of the AD and C/BE# signals as they appear on the bus. A low logic level is considered a logic 0 and a high logic level is considered a logic 1. The associated ECC signature is driven on the bus the same way: A logic 0 is represented with a low logic level, and a logic 1 is represented with a high logic level. In particular, for data phases for Memory Write transactions, C/BE[7::0]# use low logic levels for enabled bytes and are treated as logic 0 in the ECC calculations. Note that because a 32- or 64-bit data value that includes all four or eight bytes, respectively, has all byte enables asserted (logic 0), the ECC signature for the byte enables is 0, and therefore, the

ECC signature for the 32- or 64-bit data value combined with the four or eight byte enables, respectively, is identical to the ECC signature for the 32- or 64-bit data value alone. For other transactions, the byte enables are not used during data phases and the C/BE[7::0]# bus is ignored and not included in the ECC signature.

5.1.2.2.1. Seven-Bit ECC Description

The seven-bit ECC is designed to provide the following:

- Correction of all single bit errors in either the actual data (on AD[31::00]), the check bits (on ECC[6::0]), or in the additional protected signals (C/BE[3::0]# for Memory Write transactions).
- Detection of all double bit errors in the protected data and signals.
- Detection (as uncorrectable) of all triple or quadruple bit errors within a single nibble of the codeword (called "nibble protect"). Nibbles are assigned as follows: ECC[6::4], ECC[3::0], AD[31::28], AD[27::24], AD[23::20], AD[19::16], AD[15::12], AD[11::8], AD[7::4], AD[3::0], and C/BE[3::0]#.
- Each valid codeword has odd parity.
- As with any single bit correct, double bit detect ECC, if correction is disabled (see Sections 7.2.3, 8.6.2.3, and 8.6.2.4), the ECC acts as an Error Detection Code that detects all single, double, and triple bit errors. The syndrome for a single-bit error indicates the specific bit or pin failure and can be used to diagnose "stuck" electrical lines, driver/receiver failures, etc.
- Valid codewords always have at least three value-1 bits and at least two value-0 bits, which protects against failures that result in all-1 or all-0 codewords.
- Detection of most phase sequence errors. The ECC detects data transfers that are misplaced up to four positions in the phase sequence.

The ECC is specifically designed to make it suitable as a robust memory ECC. See Section E.2 for additional details.

The ECC check bits in Table 5-1 are numbered 6 through 0, corresponding with ECC[6::0]. Each row of the table is associated with either the designated data bit, numbered AD[31] through AD[00], or with C/BE[3]# through C/BE[0]#. The column marked "code" indicates the ECC signature for that bit. The ECC Check Bit columns (numbered 6 through 0) indicate the individual (XOR tree) bit taps for that bit of the ECC (corresponding to ECC[6::0]). The E16 (Chk) column provides the bit taps for the extra guard bit used on the 16-bit bus and is used only for that bus width.

Note that the values in the check bit columns correspond to the values in the code column, where each "X" or "C" in the check bit columns corresponds to a binary weighted one in the value indicated in the code column, and a blank in the check bit columns corresponds to a binary weighted zero in the value indicated in the code column. For example, the signature associated with bit AD[31] is 2Ch, or 0101100b in the check bit columns.

Table 5-1: Seven-Bit ECC Generation Table

		ECC Check Bits (E6-E0)							E16			ECC Check Bits (E6-E0)							E16
	code	6	5	4	3	2	1	0	Chk		code	6	5	4	3	2	1	0	Chk
AD[31]	2Ch		X		X	X			C	AD[15]	45h	X				X		X	
AD[30]	4Ah	X			X		X		C	AD[14]	51h	X		X				X	
AD[29]	1Ah			X	X		X		C	AD[13]	43h	X					X	X	
AD[28]	29h		X		X			X		AD[12]	61h	X	X					X	C
AD[27]	5Eh	X		X	X	X	X			AD[11]	25h		X			X		X	C
AD[26]	6Bh	X	X		X		X	X		AD[10]	31h		X	X				X	C
AD[25]	2Ah		X		X		X			AD[09]	13h			X			X	X	C
AD[24]	3Bh		X	X	X		X	X		AD[08]	5Bh	X		X	X		X	X	C
AD[23]	64h	X	X			X			C	AD[07]	46h	X				X	X		
AD[22]	26h		X			X	X		C	AD[06]	32h		X	X			X		
AD[21]	3Eh		X	X	X	X	X		C	AD[05]	23h		X				X	X	
AD[20]	15h			X		X		X	C	AD[04]	68h	X	X		X				
AD[19]	34h		X	X		X				AD[03]	4Ch	X			X	X			C
AD[18]	54h	X		X		X				AD[02]	52h	X		X			X		C
AD[17]	37h		X	X		X	X	X		AD[01]	62h	X	X				X		C
AD[16]	6Eh	X	X		X	X	X			AD[00]	49h	X			X			X	C
C/BE[3]#	3Dh		X	X	X	X		X	C	C/BE[1]#	58h	X		X	X				
C/BE[2]#	19h			X	X			X		C/BE[0]#	5Dh	X		X	X	X		X	C

For 16-bit transactions, a DWORD is sent in two 16-bit phases (as shown in Table 2-22) protected by a single ECC, and in some cases an fault affecting a single pin on the bus affects two bits. The ECC check bits are also split over the two phases with four bits of the seven-bit ECC in one phase and three bits in the other (as shown in Table 2-22). In the second phase, the fourth pin is used to transmit an extra guard bit (the calculation is by the XOR tree with the bit taps described in the E16 column above) that extends the ECC to an "extended ECC." The extra guard bit does not enhance the correctability provided by the extended ECC, it simply protects against some mis-corrections. When an error occurs, the seven-bit part of the extended ECC determines the actual correction (if any) and the extra guard bit is only used to validate the correction, turning some mis-correction cases into cases where an uncorrectable error is recognized. The additional protection provided by this guard bit is estimated to offset the higher likelihood of multi-bit errors due to the possibility of two data bits being corrupted by a problem on a single pin.

The basic ECC signature for a given data pattern in AD[31::00] and C/BE[3::0]# is calculated by accumulating (via XOR) the signature for each active (1) bit in the data. This value is commonly calculated using a bit-wise calculation of the individual ECC signature bits. Each bit of ECC[6::0] (and E16 on the 16-bit bus) is the output of an independent (bit-wise) XOR tree with input taps indicated by the "X"s (or "C"s) in the corresponding ECC check bit column in Table 5-1. For example, the XOR tree for check bit 5 taps AD[31], AD[28], AD[26], AD[25], etc. The maximum width of any of these XOR trees (with C/BE[3::0]# included) is 19 inputs. After reduction of common sub-terms, these seven XOR trees (as a group) are implementable in less than 80 two-input XOR gates. An eighth XOR tree is needed for the extra guard bit used on the 16-bit bus. Term sharing with the other XOR trees allows this extra XOR tree to be implemented with less than 10 two-input XOR gates.

It should be noted that the XOR of the seven check bits of the basic signature provides the parity for AD[31::00] and C/BE[3::0]#, which is identical to the parity-mode PAR signal.

For all transactions other than Memory Write, the values of C/BE[3::0]# are not included in the check bit calculation for data phases. In those cases, only the ECC signature (and E16 check bit) for AD[31::00] is used.

The basic ECC signature for the data is modified by the inclusion (again, via XOR) of the phase protection signature described in Section 5.1.2.2.3 to arrive at the final ECC signature for the codeword on the bus. Note that this implies that the ECC signature is thus dependent on the type of phase (address, attribute, and data), and, for data phases, on the address and width of the data phase or subphase. A codeword that is valid in one phase or subphase is, therefore, not valid in a different phase or subphase.

5.1.2.2.2. Eight-Bit ECC Description

The eight-bit ECC is an extension of the seven-bit ECC. The ECC signatures for AD[31::00] and (where applicable) C/BE[3::0]# are the ones provided in Table 5-1 (extended by adding a new blank column under ECC check bit 7) and the signatures for AD[63::32] and (where applicable) C/BE[7:4]# are provided in Table 5-2.

Table 5-2: Eight-Bit ECC Generation Table Extension

		ECC Check Bits (E7-E0)										ECC Check Bits (E7-E0)							
	code	7	6	5	4	3	2	1	0		code	7	6	5	4	3	2	1	0
AD[63]	D3h	X	X		X			X	X	AD[47]	BAh	X		X	X	X		X	
AD[62]	B5h	X		X	X		X		X	AD[46]	AEh	X		X		X	X	X	
AD[61]	E5h	X	X	X			X		X	AD[45]	BCh	X		X	X	X	X		
AD[60]	D6h	X	X		X		X	X		AD[44]	9Eh	X			X	X	X	X	
AD[59]	A1h	X		X					X	AD[43]	DAh	X	X		X	X		X	
AD[58]	94h	X			X		X			AD[42]	CEh	X	X			X	X	X	
AD[57]	D5h	X	X		X		X		X	AD[41]	ECh	X	X	X		X	X		
AD[56]	C4h	X	X				X			AD[40]	A4h	X		X			X		
AD[55]	9Bh	X			X	X		X	X	AD[39]	B9h	X		X	X	X			X
AD[54]	D9h	X	X		X	X			X	AD[38]	CDh	X	X			X	X		X
AD[53]	C1h	X	X						X	AD[37]	DCh	X	X		X	X	X		
AD[52]	EAh	X	X	X		X		X		AD[36]	97h	X			X		X	X	X
AD[51]	CBh	X	X			X		X	X	AD[35]	B3h	X		X	X			X	X
AD[50]	ABh	X		X		X		X	X	AD[34]	ADh	X		X		X	X		X
AD[49]	C8h	X	X			X				AD[33]	9Dh	X			X	X	X		X
AD[48]	91h	X			X				X	AD[32]	B6h	X		X	X		X	X	
C/BE[7]#	C2h	X	X					X		C/BE[5]#	A7h	X		X			X	X	X
C/BE[6]#	E6h	X	X	X			X	X		C/BE[4]#	A2h	X		X				X	

The ECC signatures in the table above (AD[63::32] and C/BE[7::4]#), are derived by an inversion of the signatures in Table 5-1. The E7 bit has exactly the same definition as the parity-mode PAR64 signal, i.e., it is the XOR of AD[63::32] and C/BE[7::4]#.

The eight-bit ECC signature is calculated using one of the following (different but equivalent) methods (an implementation is free to use either method or any method producing the same value):

1. Use a standard set of eight XOR-tree functions (with each XOR tree having 36 inputs, identical to the width of the XOR trees for PAR and PAR64). With common sub-term reduction, this can be done using a fairly small number of gates.

2. Calculate the eight-bit ECC signature from the seven-bit ECC signatures for the two halves (32 bits plus possibly four C/BE#s). This approach allows the XOR trees for the seven-bit ECC to be reused with only a few additional gates need to combine the seven-bit signatures. The method to combine the (basic) seven-bit signatures is as follows:

   a. Calculate the seven-bit ECC signature for AD[31::00], and possibly C/BE[3::0]#, with Table 5-1.

   b. Calculate a seven-bit ECC signature for AD[63::32] and C/BE[7::4]# using the same Table 5-1 as if AD[63::32] were AD[31::00] and C/BE[7::4]# were C/BE[3::0]# using the standard normal mapping.

   c. Calculate the parity (PAR64) of AD[63::32] and C/BE[7::4]#. This value is calculated directly or alternatively it is calculated as the XOR of the seven (upper-data) ECC signature bits calculated in Step b.

   d. Combine these values to form the eight-bit ECC by XORing together as follows:

      - E7 = PAR64
        (or, alternatively:
        E7 = upper-E6 ⊕ upper-E5 ⊕ upper-E4 ⊕ … ⊕ upper-E0)
      - E6 = PAR64 ⊕ lower-E6 ⊕ upper-E6
      - E5 = PAR64 ⊕ lower-E5 ⊕ upper-E5
      - E4 = PAR64 ⊕ lower-E4 ⊕ upper-E4
      - E3 = PAR64 ⊕ lower-E3 ⊕ upper-E3
      - E2 = PAR64 ⊕ lower-E2 ⊕ upper-E2
      - E1 = PAR64 ⊕ lower-E1 ⊕ upper-E1
      - E0 = PAR64 ⊕ lower-E0 ⊕ upper-E0

   This feature of the ECC makes it possible to easily combine two four-byte data items with seven-bit basic ECCs (i.e., without phase or error signatures) into a single eight-byte data item with a basic eight-bit ECC. (Step d above indicates the precise formula for combining two seven-bit basic ECC signatures into a single eight bit ECC signature.) See Section E.3.1 for special considerations for bridges combining two seven-bit ECC signatures that contain phase and/or error signatures.

For all transactions other than Memory Write, the values of C/BE[7::0]# are not included in check bit calculation for data phases. In those cases, only the ECC signature for AD[63::00] is used.

The basic ECC signature for the data is modified by the inclusion (again, via XOR) of the (data) phase protection signature described in Section 5.1.2.2.3 to arrive at the final ECC signature for the codeword on the bus. The ECC signature is thus dependent on the type of phase (data), the width (64-bit), and the address of the data phase or subphase. A codeword valid in one phase or subphase is, therefore, not valid in a different phase or subphase.

A codeword using the eight-bit ECC provides the same protections described for the seven-bit ECC (in Section 5.1.2.2.1), with the following modifications and extensions:

- A valid codeword using the eight-bit ECC always has at least three value-1 bits and at least three value-0 bits.
- The parity of the overall codeword is odd. The parity of the lower bus (AD[31::00], C/BE[3::0]#, and ECC[6::0]) is always odd and the parity of the upper bus (AD[63::32], C/BE[7::4]#, and ECC[7]) is always even.
- The "nibble protect" feature extends to include ECC[7::4] (rather than ECC[6::4]), AD[63::60], AD[59::56], AD[55::52], AD[51::48], AD[47::44], AD[43::40], AD[39::36], AD[35::32], and C/BE[7::4]# as nibbles.

The eight-bit ECC is also usable as a robust memory ECC. See Section E.2 for additional details.

5.1.2.2.3. Phase Protection Description

The phase protection signature applies to all address, attribute, and data phases. The type of phase and the position of data phases in the overall transaction are used to determine the phase protection signature, as listed in Table 5-4. The phase protection signature is combined (using XOR) with the data value ECC signature (covering AD[31::00] and C/BE[3::0]# or AD[63::00] and C/BE[7::0]# as appropriate) to form the ECC signature driven on the ECC check bits.

For address and attribute phases, the phase protection signature is selected according to the type of phase, as follows:

- For single address phases, row AX is selected. For dual address cycles, row AX is used for the first address phase and row AX modified by the AL signature (i.e., AX ⊕ AL) is used for the second address phase.
- For attribute phases, row AY is selected.

The phase protection signature for data phases or subphases applies to both the seven-bit ECC and the eight-bit ECC. This signature identifies the phase as a data phase, identifies the width (32- or 64-bit), and protects the position of the phase or subphase with respect to adjacent data phases or subphases in the transaction. The phase protection signature is developed by maintaining a "phase address" for each data phase or subphase and selecting a phase protection signature from Table 5-4 as follows:

1. The value of AD[06::02] that corresponds to the data phase or subphase is the phase address, even if this address is before the starting address or after the ending address. Note that a Split Completion provides address bits AD[06::00] in the address phase and these address bits are used as the initial phase address, even if the phase address would be different if a Split Completion were not used.
2. The upper four bits of the phase address (corresponding to AD[06::03]) are used to select one of the data phase protection signatures, as shown in Table 5-3. The first QWORD starting at an ADB uses data phase signature A0 and the remaining QWORDs use A1 through A5 in a cycle. If a transfer does not start on an ADB (address bits AD[06::03] are non-zero), the first data phase signature for the transfer will use a data phase signature other than A0.

3. For 32-bit data transfers using the seven-bit ECC, the phase protection signature is further modified by including (using XOR) a width signature (marker), which also acts to protect the lower bit of the phase address. These width signatures contain an even number of bits and do not affect the overall parity. The width marker is selected according to the lowest bit of the phase address (corresponding to AD[2])as follows:
   - If the lower bit (AD[2]) is 0, use the AL marker from Table 5-4.
   - If the lower phase address bit is 1, use the AH marker.
4. The selected signature value is combined (XORed) with the basic ECC signature as specified in Sections 5.1.2.2.1 and 5.1.2.2.2.

Table 5-3: Data Phase Address Signature Selection

Data Phase Address AD[6::0]	Data Phase Address AD[6::3]	Data Phase Address Signature
00h	00h	signature A0
08h	01h	signature A1
10h	02h	signature A2
18h	03h	signature A3
20h	04h	signature A4
28h	05h	signature A5
30h	06h	signature A1
38h	07h	signature A2
40h	08h	signature A3
48h	09h	signature A4
50h	0Ah	signature A5
58h	0Bh	signature A1
60h	0Ch	signature A2
68h	0Dh	signature A3
70h	0Eh	signature A4
78h	0Fh	signature A5

The usage of the PEL, PEH, and PED error signature lines is described in Section 8.7.1.1. The usage of the ADJ and PADJ lines is described in Section E.3.1. The RSV, RS2, RS3, and RS4 signatures are reserved and not used on the bus. They are permitted to be used on other interfaces of the device, for example, as part of an extension of the basic ECC to provide extended address protection in a memory ECC (see Section E.2).

Table 5-4: Phase Protection ECC Signatures

			ECC Check Bits (E7-E0)							
Usage or Case	**Name**	**Code**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
data phase signature #5	A5	7Ch		X	X	X	X	X		
data phase signature #4	A4	7Ah		X	X	X	X		X	
data phase signature #3	A3	76h		X	X	X		X	X	
data phase signature #2	A2	75h		X	X	X		X		X
data phase signature #1	A1	73h		X	X	X			X	X
data phase signature #0 (only on an ADB boundary)	A0	79h		X	X	X	X			X
Marker for 32-bit data, AD[2]=0 Marker for second DAC phase	AL	24h			X			X		
Marker for 32-bit data, AD[2]=1	AH	18h				X	X			
Address phase signature	AX	2Fh			X		X	X	X	X
Attribute phase signature	AY	4Fh		X			X	X	X	X
32-bit data “Poison” signature Lower data “Poison” signature	PEL	70h		X	X	X				
Upper data “Poison” signature	PEH	8Fh	X				X	X	X	X
64-bit data “Poison” signature	PED	F4h	X	X	X	X		X		
Adjustment to remove phase signature after combining (see Section E.3.1)	ADJ	C3h	X	X					X	X
Adjustment when combining two signatures with PEL (see Section E.3.1)	PADJ	C8h	X	X			X			
Reserved (see Section E.2)	RSV	16h				X		X	X	
Reserved (see Section E.2)	RS2	1Fh				X	X	X	X	X
Reserved (see Section E.2)	RS3	67h		X	X			X	X	X
Reserved (see Section E.2)	RS4	57h		X		X		X	X	X

Inclusion of the phase protection signature in the ECC ensures that each valid codeword has odd parity. For 64-bit data, the parity on the lower bus (AD[31::00], C/BE[3::0]#, and ECC[6::0]) is odd and the parity on the upper bus (AD[63::32], C/BE[7::4]#, and ECC[7]) is even. The address and phase protection signature also ensures all of the following:

- At least three bits in every valid codeword are 1.
- For 32-bit phases, at least two bits in every valid codeword are 0.
- For 64-bit (data) phases at least three bits in every valid codeword are 0.

Errors detected by the phase protection mechanism appear as if they were double bit errors and these errors cannot be distinguished from other double bit errors.

5.1.2.3. ECC Checking

When a codeword is received from the bus, the syndrome is determined by combining the expected ECC signature (obtained by repeating the calculation, including the expected phase protection signature, that the data source was expected to use to transmit the data that was received) with the ECC signature that was received. If the syndrome is zero, the codeword is valid and it is assumed that no error has occurred. For data received on the 16-bit bus, the "extended" syndrome is calculated when both halves of each 32-bit DWORD have been received, since the ECC covers both halves as a unit. The extended syndrome (i.e., as stored in the ECC Status register when an error is recognized) for a value received on the 16-bit bus includes the XOR of the received E16 against the expected value of E16 calculated from the data (essentially, the E16 bit is treated as just another check bit in the syndrome calculations).

If the syndrome is non-zero (expected ECC signature does not match the received ECC signature), an error is recognized. The ECC syndrome is looked up using the "code" columns of Table 5-1, (for eight bit ECC) Table 5-2, and Table 5-5 below. Alternatively, the syndrome can be looked up by bit-wise comparison against the individual rows of the combined ECC check bit columns in the tables, matching logic 1 to "X" and logic 0 to blank, with the syndrome found if it matches a complete row. If the syndrome is found in one of those tables it is considered to be correctable.

For 16-bit transactions, the extended syndrome is looked up in either Table 5-1 or Table 5-5 and must match both the "code" column (or the corresponding ECC Check Bit columns) and the E16 column to be considered correctable. Alternatively (and equivalently), the seven-bit part of the extended syndrome can be looked up in the tables to determine if there is a potential correction and the correction validated by checking the E16 bit of the extended syndrome against the E16 ("Chk") value specified for the line of the tables located using the seven-bit part of the extended syndrome. The latter method allows an implementation to share most of the correction logic between the 32- and 16-bit buses. Note that an extended syndrome where only the E16 bit is 1 (i.e., the seven-bit part of the syndrome is zero) indicates an error in the E16 bit itself. This error is considered to be correctable, just like the other single bit syndromes that indicate an error in one of the check bits (see Table 5-5).

Note that syndromes with even parity (for the 16-bit bus only the seven-bit part of the extended syndrome is considered) always indicate an uncorrectable error, usually caused either by the inversion of an even number of data bits or possibly a phase error. It is not possible to further identify or distinguish the cause of the error in this case. If the syndrome has odd parity, it will either be found in the tables, indicating a correctable error, or it will not be located and indicates an error usually caused either by the inversion of an odd number of data bits, or by "poisoning" of the ECC due to an error detected by a bridge as described in Sections 8.7.3.2 and 8.7.3.3.

The extra guard bit (E16) on the 16-bit bus protects against some mis-corrections. For example, a triple error on AD[31], AD[25], and AD[09] results in a (seven-bit) syndrome of (2Ch ⊕ 2Ah ⊕ 13h =) 15h, which would normally be mis-interpreted (and mis-corrected) as a single bit error on AD[20]. However, the extra E16 guard bit value that results from this triple-bit combination is (1 ⊕ 0 ⊕ 1 =) 0 while the specified E16 guard bit value for AD[20] is 1, so the error is recognized as uncorrectable. The extra guard bit does not pick up all

such cases (e.g., an error on AD[31], AD[30], and AD[15] still results in a mis-correction of AD[05]), but it does catch about 50% of such cases. The increased protection against mis-corrections offsets the increase in the probability of such errors caused by a fault on a single pin affecting two bits of a DWORD.

Note that a syndrome located in Table 5-4 indicates an uncorrectable error. If the syndrome is one of PEL, PEH, or PED, the codeword indicates that there was an error forwarded by a bridge (see Section 8.7.1.1), and, for 64-bit data, PEL and PEH indicate that one of the 32-bit halves of the data is valid. For the other syndromes, there is no indication as to whether the error was in the data itself, the ECC check bits, or the result of the phase protection mechanism, or a combination of such errors.

Table 5-5: ECC Check Bit Syndromes

		ECC Check Bits (E7-E0)								E16
	code	7	6	5	4	3	2	1	0	Chk
ECC[7]/E7	80h	X								
ECC[6]/E6	40h		X							
ECC[5]/E5	20h			X						
ECC[4]/E4	10h				X					
ECC[3]/E3	08h					X				
ECC[2]/E2	04h						X			
ECC[1]/E1	02h							X		
ECC[0]/E0	01h								X	
Chk/E16	Chk									X

If the syndrome is located in one of the tables, the error is assumed to be a correctable (single bit) error that can be corrected by inverting the bit associated with the syndrome (or extended syndrome) in the tables. If the syndrome is not found in the tables, the error is determined to be uncorrectable.

If a device detects a correctable ECC error in any transaction phase, it corrects the error, unless error correction is disabled, as described in Sections 7.2.3, 8.6.2.3, and 8.6.2.4. If correction is disabled, the ECC detects all single, double, and triple bit errors. The device also always latches the appropriate diagnostic information, as described in Sections 7.2 and 8.6.2.

In ECC mode, devices treat uncorrectable errors in address and attribute phases the same as conventional PCI devices and PCI-X devices in parity mode treat address parity errors. The device asserts SERR# and sets the Detected Parity Error bit in the Status register, as specified in PCI 2.3 for address parity errors. The device also latches the appropriate diagnostic information, as described in Sections 7.2 and 8.6.2.

PCI-X devices operating in ECC mode use the same Control register bits to enable data-phase ECC error checking and signaling that conventional and PCI-X devices operating in parity mode use to enable data-phase parity checking. The following participants in a PCI-X transaction are considered to receive data and, therefore, check data-phase ECC:

- 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 ECC on the read data or Split Completion Message.

5.1.2.4. ECC Timing

On any given address and attribute phase, the ECC bus is driven by the initiator. On any given data phase, the ECC bus is driven by the device that drives the data. In address, attribute, and common-clock data phases, ECC lags the information it protects by one clock (the same as parity). For source-synchronous transactions in Mode 2, the initiator drives ECC coincident with the data it protects in each subphase.

If the device receiving data detects an uncorrectable error in a common-clock data phase, the device asserts PERR# (if enabled) two clocks after the clock in which the ECC was valid (the same as parity). If the target detects an uncorrectable error in a source-synchronous data subphase (in Mode 2), the target asserts PERR# (if enabled) four clocks after the clock that nominally ends the data phase. That is, if the target detects an uncorrectable error for data subphases that the initiator drove between clocks N and N+1, the target asserts PERR# on clock N+5.

Figure 5-7 through Figure 5-11 illustrate ECC generation and checking. In each case, the address and attribute phases are driven in clock 3 and 4, respectively, and the ECC signatures for the address and attribute phases are driven in clock 4 and 5, respectively. Figure 5-7 through Figure 5-9 show that data-phase ECC generation and checking for DWORD transactions is the same as parity generation and checking. Common-clock burst ECC (not shown) would be the same as burst parity, also.

Figure 5-10 and Figure 5-11 illustrate ECC generation and checking on a (Mode 2) source-synchronous burst push transaction. Both figures show the transition from common-clock timing in clock 4 and 5 (in which ECC lags the data it protects) to source-synchronous timing between clocks 5 and 6 (in which ECC is driven coincident with the data it protects). Since TRDY# is not asserted in clock 6, ECC is not checked for the data driven between clocks 5 and 6. Figure 5-10 shows TRDY# asserted in clock 7, so ECC is checked for data driven between clocks 6 and 7, and if an error is detected, PERR# is asserted in clock 11.

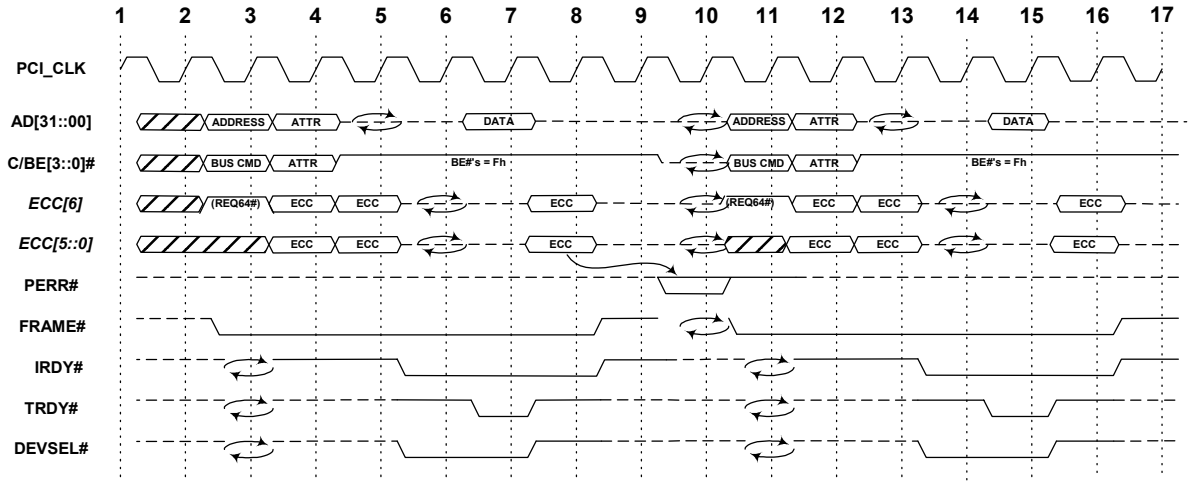

Figure 5-7: DWORD Read Transaction ECC Operation, Decode B and No Initial Wait States

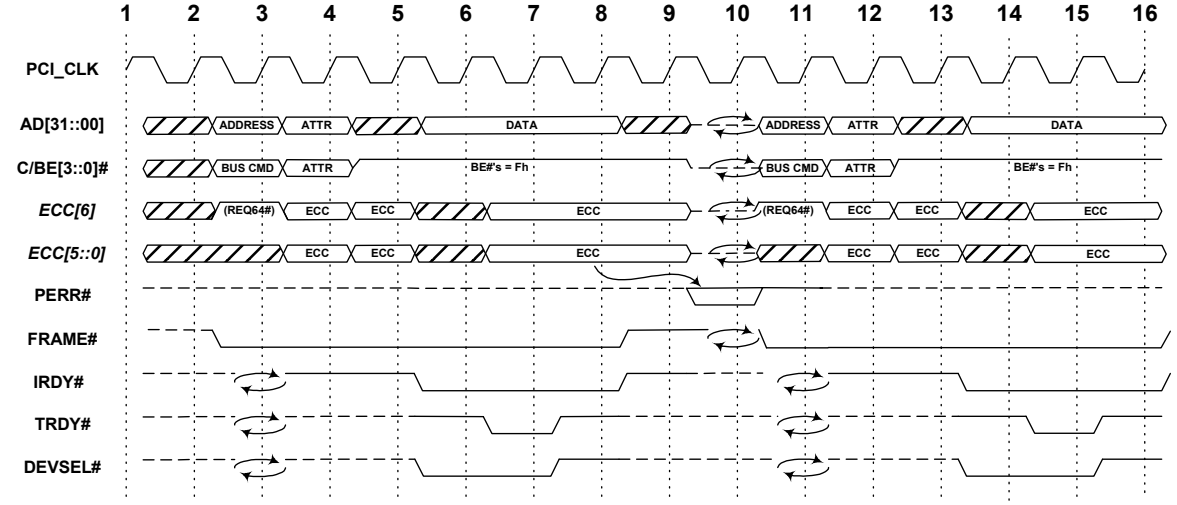

Figure 5-8: DWORD Write Transaction ECC Operation, Decode B and No Initial Wait States

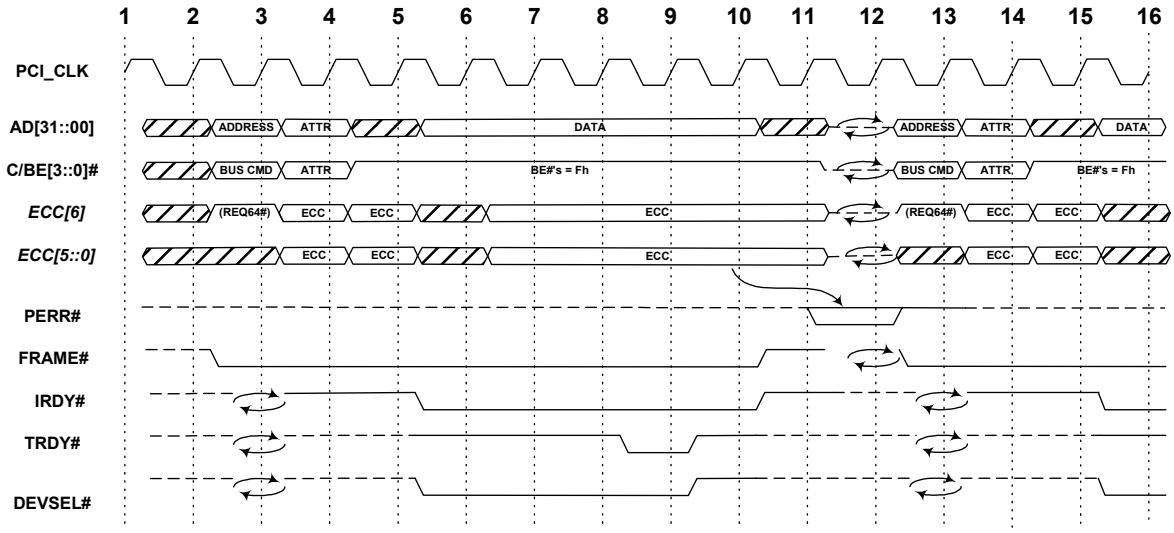

Figure 5-9: DWORD Write Transaction ECC Operation, Decode B and Two Initial Wait States

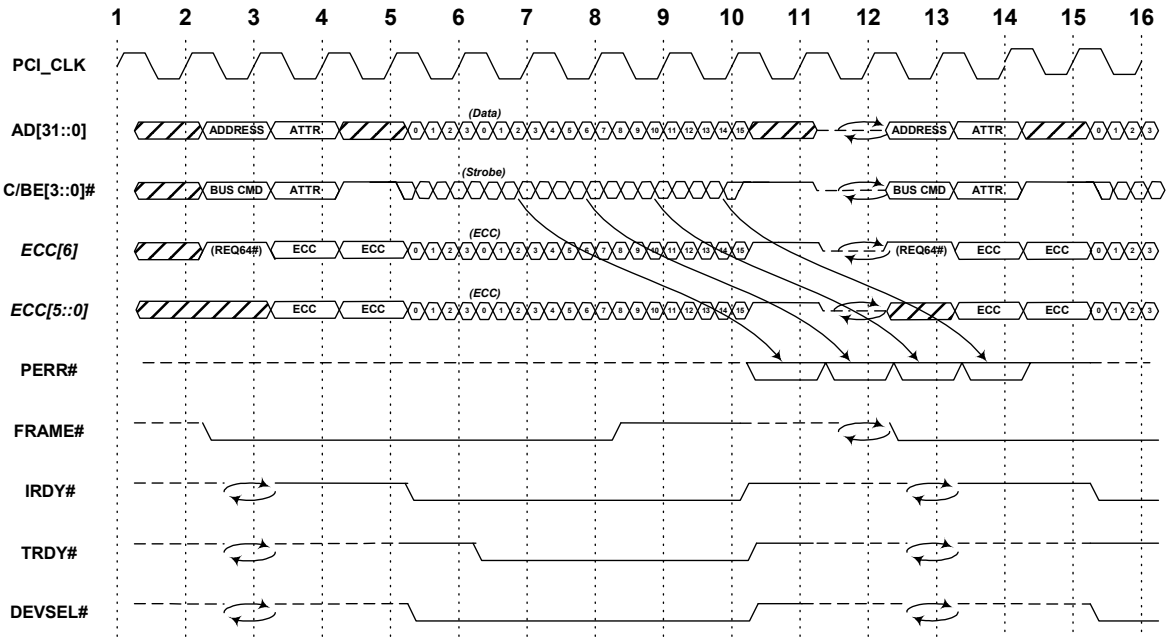

Figure 5-10: Source-Synchronous (PCI-X 533) Transaction ECC Operation, Decode B and No Initial Wait States

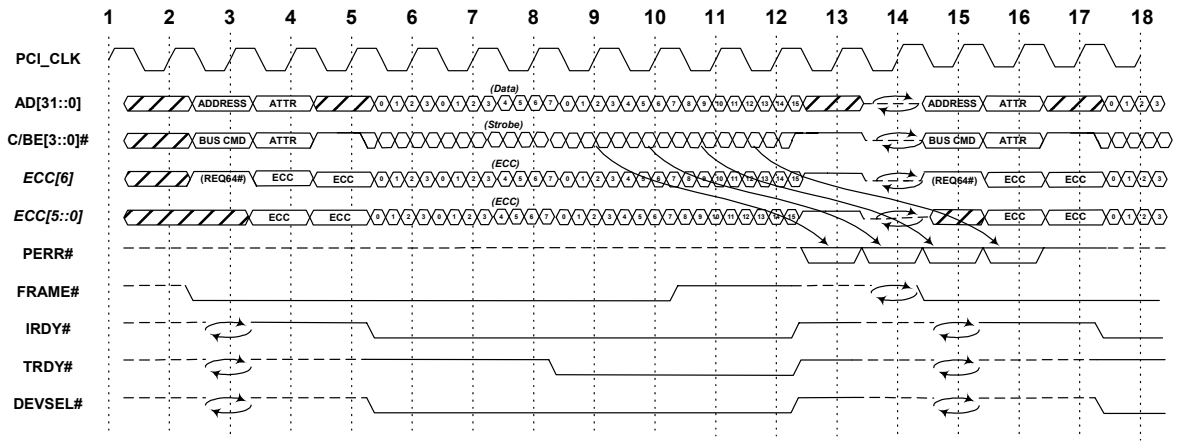

Figure 5-11: Source-Synchronous (PCI-X 533) Transaction ECC Operation, Decode B and Two Initial Wait States

5.1.3. ECC on 16-Bit Transfers

16-bit transactions always use seven-bit ECC plus one additional check bit, as described in Section 5.1.2.2.1. The data protected by these eight bits is driven in two data phases or subphases as described in Section 2.12.2.3. The eight check bits are driven in two phases or subphases as described in Table 2-22.

As in 64- and 32-bit transfers, address, attribute, and common-clock data phase ECC lags the address, attributes, and data it protects by one clock, as shown in the figures below. During source-synchronous data subphases, ECC is driven in two data subphases coincident with the data it protects.

If an uncorrectable data error is detected in a common-clock burst transaction, the target asserts PERR# (if enabled) after the second of the two data phases that contain the error with the same timing as for 64- and 32-bit transactions. Note that for common-clock burst transactions, PERR# is permitted to be asserted every other clock, because two data phases are required to transfer a full DWORD. If an uncorrectable data error is detected in a source-synchronous transaction, the target asserts PERR# (if enabled) after the single data phase that contains the error with the same timing as for 64- and 32-bit transactions.

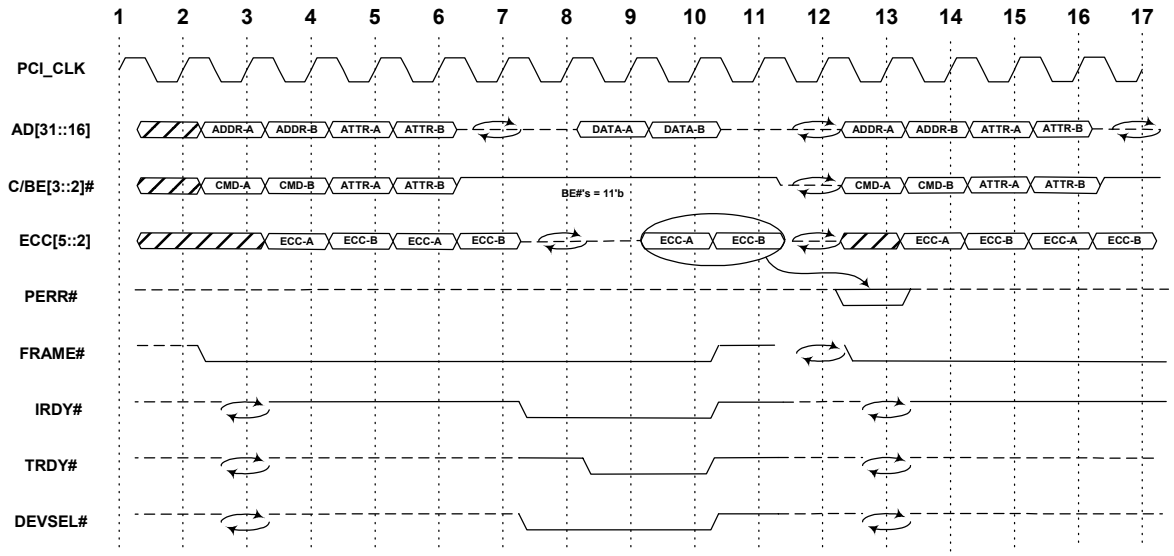

Figure 5-12: DWORD Read Transaction ECC Operation, Decode B and No Initial Wait States, 16-Bit Mode 2

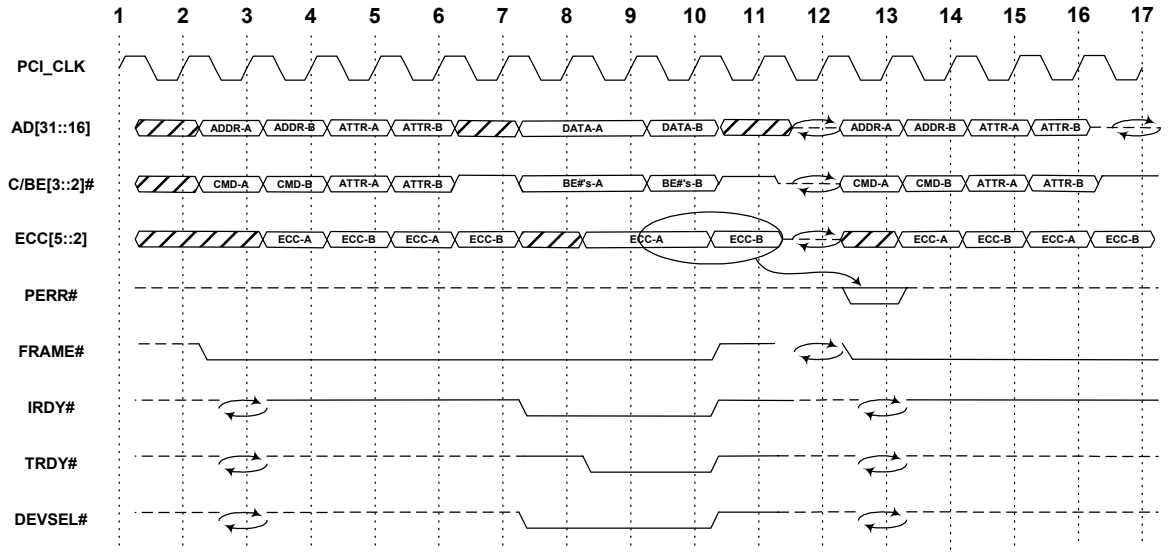

Figure 5-13: DWORD Write Transaction ECC Operation, Decode B and No Initial Wait States, 16-Bit Mode 2

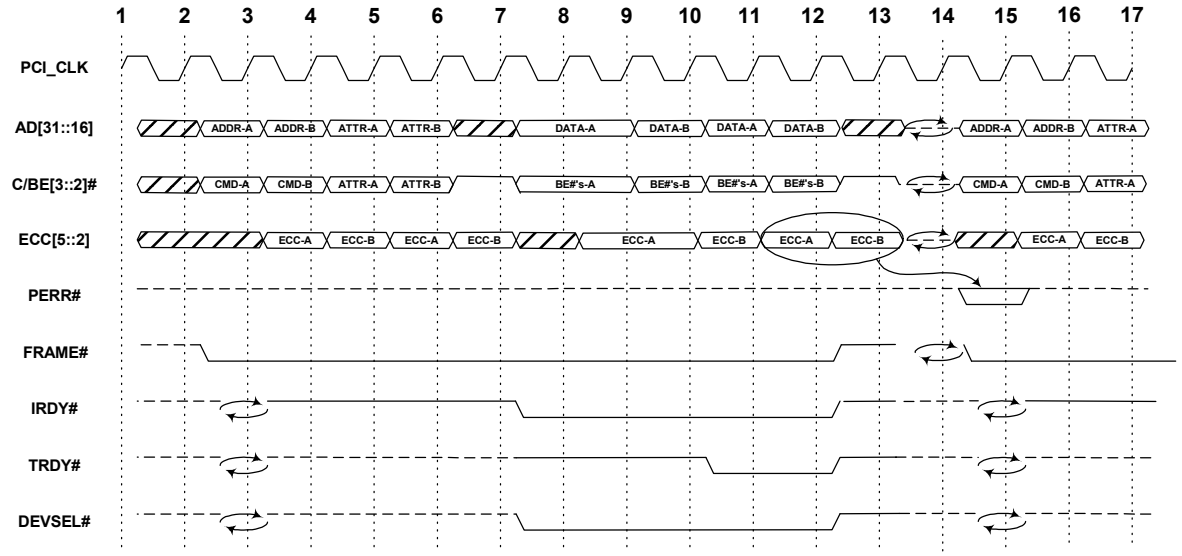

Figure 5-14: DWORD Write Transaction ECC Operation, Decode B and Two Initial Wait States, 16-Bit Mode 2

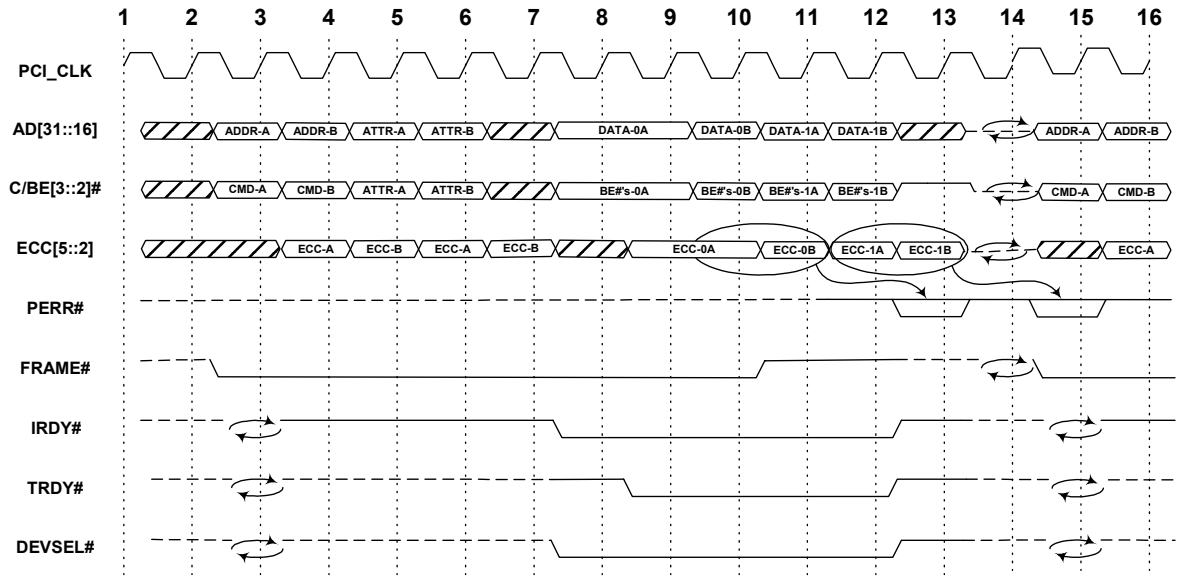

Figure 5-15: Common-Clock Burst Write Transaction ECC Operation, Decode B and No Initial Wait States, 16-Bit Mode 2

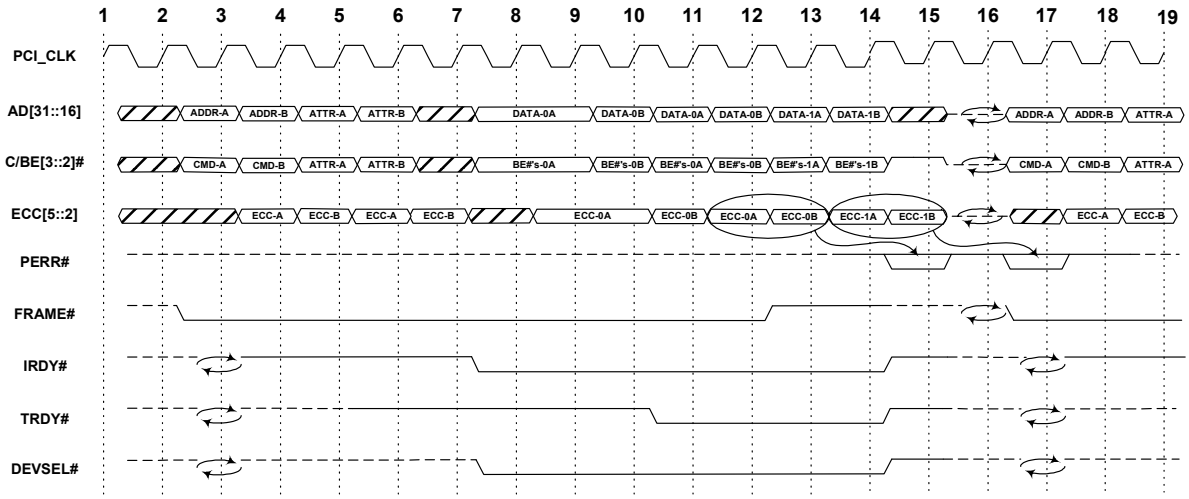

Figure 5-16: Common-Clock Burst Write Transaction ECC Operation with DEVSEL# B and Two Initial Wait States, 16-Bit Mode 2

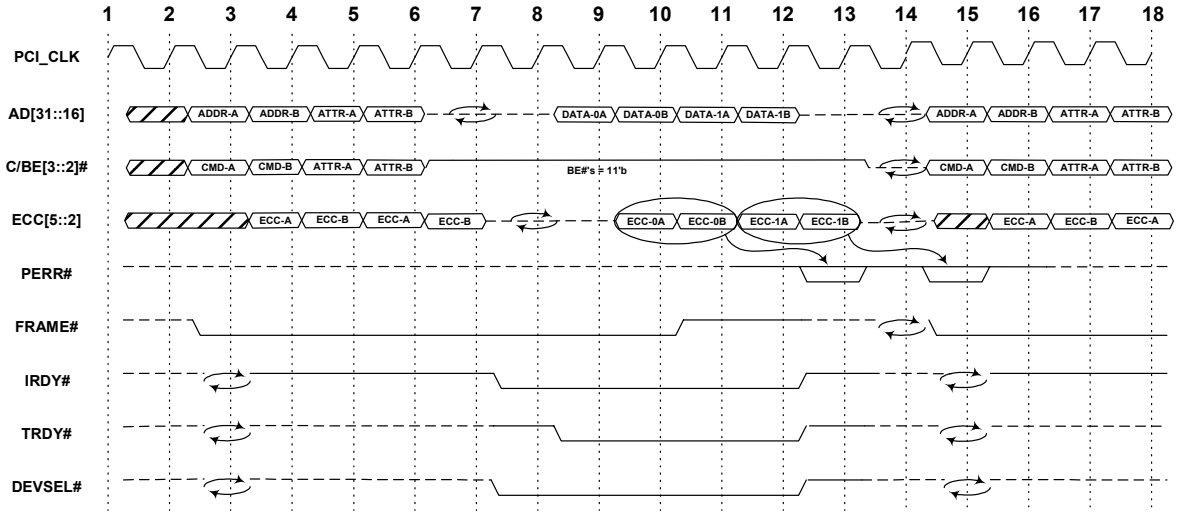

Figure 5-17: Common-Clock Burst Read Transaction ECC Operation, Decode B and No Initial Wait States, 16-Bit Mode 2

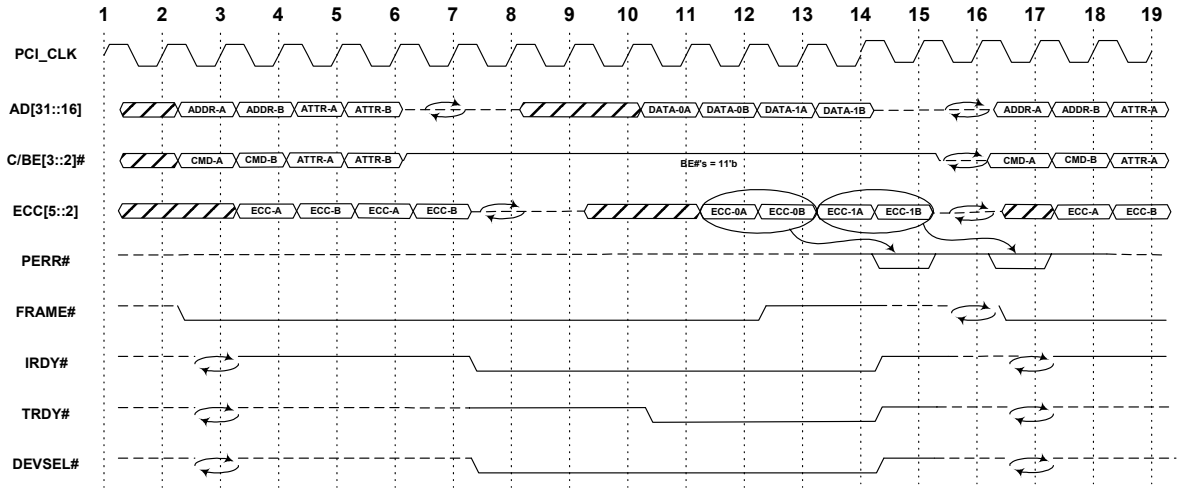

Figure 5-18: Common-Clock Burst Read Transaction ECC Operation, Decode B and Two Initial Wait States, 16-Bit Mode 2

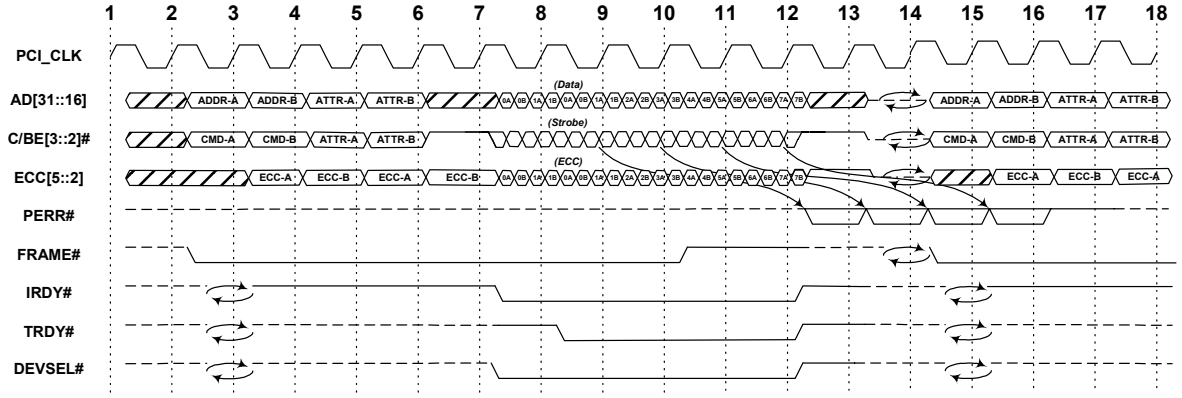

Figure 5-19: Source-Synchronous (PCI-X 533) Burst Write Transaction ECC Operation, Decode B and No Initial Wait States, 16-Bit Mode 2

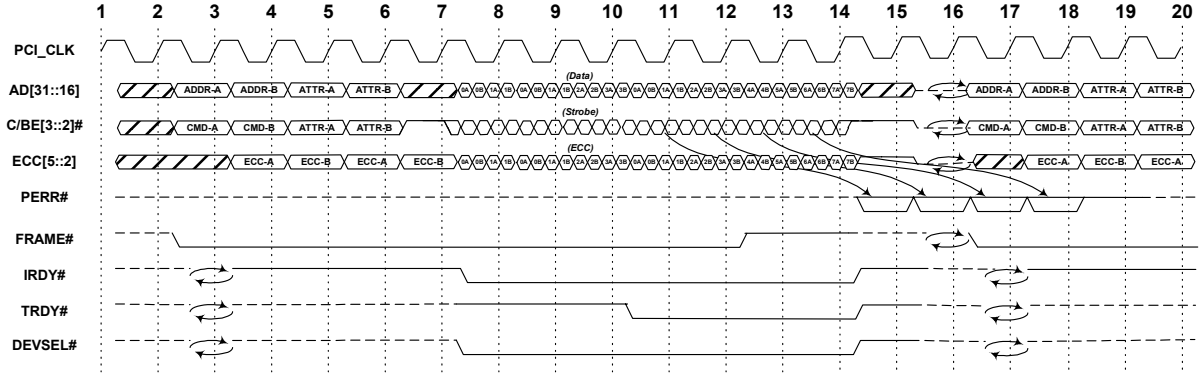

Figure 5-20: Source-Synchronous (PCI-X 533) Burst Write Transaction ECC Operation, Decode B and Two Initial Wait States, 16-Bit Mode 2

5.1.4. Driving SERR#

As in conventional PCI, SERR# is an open-drain signal with a passive pull-up resistor on the system board. The assertion of SERR# is synchronous to CLK and meets the setup and hold times specified in Section 2.1.2.4.2, "3.3V Environment Timing Parameters," in PCI-X EM 2.0. However, the restoring of SERR# to the deasserted state is accomplished by a weak pull-up, which typically takes more than one clock period to fully restore it.

5.2. Error Handling and Fault Tolerance

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

5.2.1. Uncorrectable Data Errors

In parity mode, 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. In ECC mode, if data error checking is enabled and a device receiving data detects an uncorrectable data error, it must assert PERR# as described in Section 5.1.2.4.

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 an uncorrectable data error.
- The initiator (requester or PCI-X bridge) of a read transaction that is terminated with Split Response calculates an uncorrectable data error in the Split Response.
- The initiator (requester or PCI-X bridge) of a write that is completed immediately or is terminated with Split Response observes PERR# asserted with the timing indicated in Section 5.1.1.3 in parity mode or Section 5.1.2.4 in ECC mode.
- The target (requester or PCI-X bridge) of a Split Completion calculates an uncorrectable data error in either read data or a Split Completion Message.
- The target (requester or PCI-X bridge) receives a Split Completion Message that indicates an uncorrectable data error occurred on one of this device's non-posted write transactions (see Section 5.2.6).

The Detected Parity Error (bit 15) bits in the Status register is set by the device whose data checking logic calculated the uncorrectable data 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 an uncorrectable data 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 uncorrectable data error conditions are always catastrophic to the system.

Devices are allowed to attempt to recover from an uncorrectable data 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 uncorrectable data 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 2.3.

5.2.1.1. Devices and Software Drivers that Support Recovery from Uncorrectable Data Errors

When RST# is asserted, all PCI-X devices clear the Uncorrectable Data Error Recovery Enable bit in the PCI-X Command register. If the operating system loads a software device driver that supports recovery from uncorrectable data errors, 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 an uncorrectable data error (that is, the device that sets the Master Data Parity Error bit as described in Section 5.2.1) 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 an uncorrectable data 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 add-in card and continue
- Take the add-in 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 an uncorrectable data error. If these bits are to be cleared, the operating system vendor must specify what software is responsible for clearing them.

5.2.1.2. Devices or Software Drivers That Do Not Support Recovery from Uncorrectable Data Errors

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

In addition to the requirements specified in Section 5.2.1, a PCI-X device asserts SERR# (if enabled) after an uncorrectable data error if both of the following are true:

- The Uncorrectable Data Error Recovery Enable bit in the PCI-X Command register is cleared.
- An uncorrectable data error occurred that caused the device to set the Master Data Parity Error bit (see Section 5.2.1).

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 System-Specific Recovery Routines for Uncorrectable Data Errors

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

Systems that support PERR# recovery using system-specific recovery routines provide system-specific ROM software that at power-up sets the Uncorrectable Data 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 an uncorrectable data error but assert PERR# as conventional PCI devices do. The system-specific service routines execute the system-specific uncorrectable data error recovery routines before returning control to the operating system.

5.2.1.3. Uncorrectable Data Errors in Split Response for Read Transactions

If an initiator (requester or bridge) calculates an uncorrectable data error when the target (completer or bridge) signals Split Response for a read transaction, the initiator must record the error as described in Section 5.2.1. Furthermore, if the initiator is enabled to assert PERR# and does not support recovery from uncorrectable data errors (i.e., Uncorrectable Data 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 uncorrectable data errors and the Uncorrectable Data 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 Uncorrectable Data Errors in Split Response

An uncorrectable data 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 bus 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 an uncorrectable data 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 uncorrectable data 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 an uncorrectable data 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.

5.2.2. 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 2.3 in which a Master-Abort does not indicate an error condition, like configuration or Special Cycle transactions and in PCI-X Mode, Device ID Message transactions). If notification of the device driver is not possible, the initiator must assert SERR# (if enabled) and update its Status register. See Section 5.2.4 for errors that occur on Split Completion transactions.

5.2.3. Uncorrectable Address and Attribute Errors

A PCI-X target that detects an uncorrectable 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 2.3. 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 2.3, if the device asserts DEVSEL# prior to detecting an uncorrectable 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). In ECC mode, the device must check the address phase before asserting DEVSEL#. 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.)

5.2.4. 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 uncorrectable data errors. (See Section 8.7.1.1.2 for the requirements for a bridge.) If the completer detects an uncorrectable data error on a DWORD write transaction and signals Data Transfer (an Immediate Transaction), the completer asserts PERR#. If the completer detects an uncorrectable data 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 an uncorrectable data 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).

5.2.5. 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.

5.2.6. 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 uncorrectable data 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 5-6 shows what bits in the Status register or Secondary Status register must be set when a requester or bridge receives these messages.

Table 5-6: 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	Uncorrectable Write Data Error	Master Data Parity Error
2	00h	Byte Count Out of Range	none
2	01h	Uncorrectable Split Write Data 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 an uncorrectable data 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 PCI Bridge 1.1 for a device that forwards a non-posted write as a Delayed Transaction. If such a bridge detects an uncorrectable data 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 an uncorrectable data error on a non-posted write transaction.

6

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 2.3 when operating in that mode. PCI-X devices may optionally meet the 66 MHz timing requirements of PCI 2.3.

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, so PCI-X 133 devices automatically meet the requirements for PCI-X 66 operation (see Section 2.1.2.4.1, "Clock Specification," in PCI-X EM 2.0).

PCI-X 533 and PCI-X 266 devices are required also to meet the requirements for PCI-X 133 devices, including operating in PCI-X Mode 1 and using 3.3V I/O signaling levels. PCI-X 533 devices are required also to meet the requirements for PCI-X 266 devices.

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. (PCI-X Mode 2 devices do not support 5V I/O conventional PCI buses.) 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 Mode 1 devices use 3.3V I/O signaling levels defined in Section 2.1.2, "3.3V Signaling Environment," in PCI-X EM 2.0 (compatible with that defined in PCI 2.3) when operating in PCI-X mode. PCI-X Mode 1 devices optionally also support the 5V I/O signaling levels defined in PCI 2.3 when operating in conventional 33-MHz PCI mode. PCI-X Mode 1 add-in cards must be keyed either for 3.3V I/O or Universal I/O as defined in PCI 2.3.

PCI-X Mode 2 devices use 1.5V I/O signaling levels defined in Section 2.1.3, "1.5V Signaling Environment," in PCI-X EM 2.0 when operating in Mode 2 and 3.3V I/O signaling defined in Section 2.1.2, "3.3V Signaling Environment," in PCI-X EM 2.0 when operating in Mode 1. PCI-X Mode 2 devices do not support 5V I/O signaling. PCI-X Mode 2 add-in cards must be keyed for 3.3V I/O (not 5V I/O and not Universal). (The I/O voltage is set to 1.5V if the system determines that all devices on the bus are capable of operating in PCI-X Mode 2.)

6.1.2. System Interoperability Requirements

The bus mode and speed can be set no higher than the maximum mode and transfer rate capability of the slowest device present on the bus and no higher than the maximum mode and transfer rate for which the bus was designed. How the system determines the maximum mode and transfer rate for which the bus was designed is beyond the scope of this specification.

If a bus includes at least one conventional 33-MHz 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 2.3 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 PCI-X Mode 2 bus includes only PCI-X Mode 2 devices, the system sets $V_{I/O}$ to 1.5V and the bus operates in PCI-X Mode 2. If the bus includes at least one PCI-X 266 device, the bus operates in PCI-X 266 mode with two data subphases per data phase.

If the bus includes only PCI-X Mode 1 devices or a mix of PCI-X Mode 1 and Mode 2 devices, the system sets $V_{I/O}$ to 3.3V and the bus operates in PCI-X Mode 1. 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 2.1.2.4.1, "Clock Specification," in PCI-X EM 2.0. This is generally done to support higher loading on the bus. For example, a bus with two expansion slots in PCI-X Mode 1 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 Add-in Cards			PCI-X Mode 1 Add-in Cards[1]		PCI-X Mode 2 Add-in Cards[7]	
		33 MHz (5V I/O)	**33 MHz (3.3V I/O or Universal)**	**66 MHz (3.3V I/O or Universal)**	**PCI-X 66 (3.3V I/O or Universal)**	**PCI-X 133 (3.3V I/O or Universal)**	**PCI-X 266 (3.3V I/O)[7]**	**PCI-X 533 (3.3V I/O)[7]**
Conventional System	PCI 33 (5V I/O) 10 ns[2,3]	PCI 33 (5V I/O)	PCI 33 (5V I/O)	PCI 33 (5V I/O)	PCI 33 (5V I/O)	PCI 33 (5V I/O)	Not supported	Not supported
	PCI 33 10 ns[2]		PCI 33	PCI 33	PCI 33	PCI 33	PCI 33	PCI 33
	PCI 66 5 ns[2]		PCI 33	PCI 66	a) PCI 33[5] b) PCI 66	a) PCI 33[5] b) PCI 66	a) PCI 33[5] b) PCI 66	a) PCI 33[5] b) PCI 66
PCI-X Mode 1 System	PCI-X 66 9 ns[2]		PCI 33	a) PCI 33[4,6] b) PCI 66	PCI-X 66	PCI-X 66	PCI-X 66	PCI-X 66
	PCI-X 133 2 ns[2]		PCI 33	a) PCI 33[4] b) PCI 66	PCI-X 66	PCI-X 133	PCI-X 133	PCI-X 133
PCI-X Mode 2 System	PCI-X 266 2 ns[2]		PCI 33	a) PCI 33[4,6] b) PCI 66	PCI-X 66	PCI-X 133	PCI-X 266	PCI-X 266
	PCI-X 533 2 ns[2]		PCI 33	a) PCI 33[4] b) PCI 66	PCI-X 66	PCI-X 133	PCI-X 266	PCI-X 533

Legend:

xx — Conventional PCI system or add-in card operating in conventional mode
xx = operating mode

xx — PCI-X system and add-in card operating in PCI-X mode
xx = operating mode

xx — Most popular cases

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

Notes:

1. Unless otherwise specified, all cases use an I/O signaling voltage of 3.3V.
2. Time indicates maximum value of T_{prop} for the system shown. 33 MHz and 66 MHz values are taken from PCI 2.3. PCI-X values come from Table 2-37, "Setup Time Budget for Clock Jitter Class 1, Mode 1," in PCI-X EM 2.0.
3. Most systems shipped prior to the development of the PCI-X definition fall into this row. 5V and Universal add-in cards work here but not 3.3V add-in cards.

4. PCI-X systems optionally operate in 33 MHz mode or 66 MHz mode when only 66 MHz conventional PCI add-in 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.
7. PCI-X Mode 2 cards are keyed for 3.3V I/O, even though the voltages is set to 1.5V in Mode 2.

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 2.3. This requirement is the same in PCI-X Mode 2 (even though the REQ64# pin is shared with ECC[6], as described in Section 2.4, "Mode 2 Pin Sharing," in PCI-X EM 2.0).

Add-in cards indicate whether they support PCI-X, and if so which frequency and mode, by the way they connect one pin called PCIXCAP. If the add-in card's maximum frequency and mode is PCI-X 133, it leaves this pin unconnected (except for a decoupling capacitor specified in Section 2.3.4, "PCIXCAP and MODE2 Connection," in PCI-X EM 2.0). If the add-in card's maximum frequency and mode is PCI-X 66, it connects PCIXCAP to ground through a resistor (and decoupling capacitor) specified in Section 2.3.4, "PCIXCAP and MODE2 Connection," in PCI-X EM 2.0. If the add-in card's maximum frequency and mode is PCI-X 266 or PCI-X 533, it connects PCIXCAP to ground through an appropriate resistor value (and decoupling capacitor, and other supporting circuitry) as specified in Section 2.3.4, "PCIXCAP and MODE2 Connection," in PCI-X EM 2.0. (The other supporting circuitry prevents PCI-X Mode 2 cards from being misinterpreted as PCI-X 66 cards in Mode 1 systems that contain multiple slots.) Conventional add-in cards connect this pin to ground. See Appendix A, "Detection of PCI-X Add-in Card Capability," in PCI-X EM 2.0 for recommendations for circuits to detect the types of add-in 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 (Note)	Conventional Device Frequency Capability	PCI-X Device Capability
Ground	Ground	33 MHz	Not capable
Not connected	Ground	66 MHz	Not capable
Ground	Pull-down, R1	33 MHz	PCI-X 66
Not connected	Pull-down, R1	66 MHz	PCI-X 66
Ground	Pull-down, R2	33 MHz	PCI-X 266
Not connected	Pull-down, R2	66 MHz	PCI-X 266
Ground	Pull-down, R3	33 MHz	PCI-X 533
Not connected	Pull-down, R3	66 MHz	PCI-X 533
Ground	Not connected	33 MHz	PCI-X 133
Not connected	Not connected	66 MHz	PCI-X 133

Note:
Resister values are specified in Section 2.3.4, "PCIXCAP and MODE2 Connection," in PCI-X EM 2.0.

When power is applied to the slot, the voltage level of $V_{I/O}$ is set according the mode at which the bus is about to operate. If the bus is to operate in PCI-X Mode 1, $V_{I/O}$ is set to the 3.3V range. If the bus is to operate in PCI-X Mode 2, $V_{I/O}$ is set to the 1.5V range. $V_{I/O}$ is required to be stable and within the appropriate voltage for the same time period before the rising edge of RST# that is specified in PCI 2.3 for all supply voltages.

The source bridge places all devices on that segment in the appropriate mode by driving a particular combination of control signals on the bus at the rising edge of RST#. This is called the PCI-X initialization pattern. The PCI-X initialization pattern is latched on the rising edge of RST#. For PCI-X Mode 1 source bridges, the initialization pattern uses FRAME#, IRDY#, DEVSEL#, STOP#, and TRDY#. For PCI-X Mode 2 source bridges, the initialization pattern also uses PERR# (see Table 6-2). A PCI-X source bridge that has at least one conventional device on its secondary bus initializes the bus in conventional mode. A PCI-X Mode 1 source bridge that has no conventional devices on its secondary bus initializes the bus in PCI-X Mode 1. A PCI-X Mode 2 source bridge that has no conventional devices on its secondary bus, but has at least one PCI-X Mode 1 device, initializes the bus in PCI-X Mode 1. A PCI-X Mode 2 source bridge that has no conventional or PCI-X Mode 1 devices on its secondary bus initializes the bus in PCI-X Mode 2.

Conventional 33 MHz and 66 MHz modes are set as described in PCI 2.3.

If the bus is initialized in PCI-X Mode 1, the initialization pattern also indicates whether the devices are to be initialized in parity mode or ECC mode. The system is permitted to initialize a bus in PCI-X Mode 1 and ECC mode only if all devices on the bus support ECC in Mode 1. The method by which the system determines that all devices support ECC in Mode 1 is not specified.

Table 6-2: PCI-X Initialization Pattern

PERR#	DEVSEL#	STOP#	TRDY#	Mode	Error Protection	Max Clock Period (ns)	Min Clock Period (ns)	Min Clock Freq (MHz) (ref)	Max Clock Freq (MHz) (ref)
D	D	D	D	Conventional 33 MHz	Parity	∞	30	0	33
D	D	D	D	Conventional 66 MHz	Parity	30	15	33	66
D	D	D	A	PCI-X Mode 1	Parity	20	15	50	66
D	D	A	D	PCI-X Mode 1	Parity	15	10	66	100
D	D	A	A	PCI-X Mode 1	Parity	10	7.5	100	133
D	A	D	D	PCI-X Mode 1	ECC	Reserved	Reserved	Reserved	Reserved
D	A	D	A	PCI-X Mode 1	ECC	20	15	50	66
D	A	A	D	PCI-X Mode 1	ECC	15	10	66	100
D	A	A	A	PCI-X Mode 1	ECC	10	7.5	100	133
A	D	D	D	PCI-X 266 (Mode 2)	ECC	Reserved	Reserved	Reserved	Reserved
A	D	D	A	PCI-X 266 (Mode 2)	ECC	20	15	50	66
A	D	A	D	PCI-X 266 (Mode 2)	ECC	15	10	66	100
A	D	A	A	PCI-X 266 (Mode 2)	ECC	10	7.5	100	133
A	A	D	D	PCI-X 533 (Mode 2)	ECC	Reserved	Reserved	Reserved	Reserved
A	A	D	A	PCI-X 533 (Mode 2)	ECC	20	15	50	66
A	A	A	D	PCI-X 533 (Mode 2)	ECC	15	10	66	100
A	A	A	A	PCI-X 533 (Mode 2)	ECC	10	7.5	100	133

Legend:
D Deasserted.

A Asserted.

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.

IMPLEMENTATION NOTE

Use of PERR# in the PCI-X Initialization Pattern

PCI-X Mode 2 expands the initialization pattern to include the PERR# signal. This means that PCI-X Mode 2 source bridges assert PERR# when the system is first initialized, or when a card is hot added. System logic that monitors PERR# for error reporting purposes must be masked whenever the source bridge drives an initialization pattern that includes asserting PERR#.

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 Mode 2 devices select input and output buffer electrical behavior based on the magnitude of $V_{I/O}$. That is, if $V_{I/O}$ is in the 3.3V ranges (as specified in Section 2.1.2.1, "3.3V DC Specifications," in PCI-X EM 2.0), the device selects input and output buffers for Mode 1 operation. If $V_{I/O}$ is in the 1.5V range (as specified in Section 2.1.3.1, "1.5V DC Specifications," in PCI-X EM 2.0), the device selects input and output buffers for Mode 2 operation. Devices are permitted combinatorially to change between modes while $V_{I/O}$ is settling to its final value. $V_{I/O}$ is guaranteed to be stable and in one of these ranges before the rising edge of RST# with the same timing as the other supply voltages specified in PCI 2.3. If RST# is asserted and $V_{I/O}$ is in the 1.5V range, the device must disable all of its electrical terminators, as described in Section 2.1.3.7, "1.5V Environment Signal Electrical Termination," in PCI-X EM 2.0.

Except for input and output buffer selection in PCI-X Mode 2, PCI-X devices enter conventional mode or PCI-X Mode 1 or PCI-X Mode 2 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, PCI-X Mode 1, and PCI-X Mode 2 based on this pattern at the rising edge of RST#. When the system powers up, FRAME#, IRDY#, DEVSEL#, STOP#, TRDY#, and PERR# 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 modes 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. If changing between Mode 1 and Mode 2, a $V_{I/O}$ change is also possible while RST# is asserted.

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 in a PCI-X Mode 1 device. A PCI-X Mode 2 device would be similar. 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#.

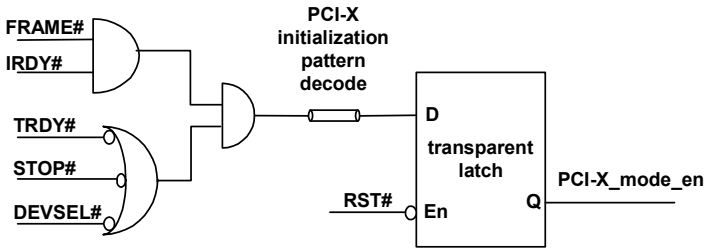

Figure 6-2: PCI-X Mode Latch, Mode 1 Device

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 2.3 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). A Mode 2 device must also ignore PERR#, except at the appropriate time to indicate an error after a data phase. 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).

If the PCI-X initialization pattern indicates the device is to enter PCI-X Mode 2, the device keeps its interface in the interface low-power state after the rising edge of RST# until the first bus turn-around alert, as described in Section 4.1.2.3.

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 2.1.2.4.1, "Clock Specification," in PCI-X EM 2.0) 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 used for conventional 66 MHz mode and 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 used for PCI-X mode and 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#.

IMPLEMENTATION NOTE

Resetting the Device with RST#

Some devices include state information that is not reset by RST#. For example, a device that downloads a large amount of configuration information from an external non-volatile memory component might sense the voltage level on the power supply voltages and begin downloading as soon as the power supply voltages reach a sufficient level, even if RST# is still asserted. Such a device would have no need to repeat the download the next time RST# was asserted, unless the supply voltages dropped low enough to prevent the device from saving the original configuration.

As specified in PCI 2.3, all devices must reset their entire interface whenever RST# is asserted. PCI 2.3 and the PCI-X definition do not specify any behavior of the power supply voltages if RST# asserts after the system is running. In some cases, power supply voltages remain stable while RST# is asserted. In other cases, power supply voltages go to zero. In

still other cases, the power supply voltages drop out of the specified limits for a brief period without going all the way to zero, and then return to the specified levels. The only requirement that the system must meet is to wait for T_{pvrh}, as specified in Section 2.1.2.4.2, "3.3V Environment Timing Parameters," in PCI-X EM 2.0, after the supply voltages are stable before deasserting RST#. If a device includes additional logic states that are not cleared by RST# (e.g., states that are cleared by detecting the voltage level of the power supplies), these states must not interfere with the normal reset of the device interface by RST#, or the normal initialization and operation of the device after the rising edge of RST#, regardless of the behavior of the supply voltages prior to the time they stabilize T_{pvrh} before 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 $V_{I/O}$ and 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 $V_{I/O}$ are the same as all the other voltages, as specified in PCI 2.3. The timing requirements for the PCI-X initialization pattern are shown in Figure 2-35, "RST# Timing for Switching to PCI-X Mode," in PCI-X EM 2.0. Timing parameter values are specified in Table 2-7, "3.3V General Timing Parameters," in PCI-X EM 2.0 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 PCI-X Mode 1 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 Appendix A, "Detection of PCI-X Add-in Card Capability," in PCI-X EM 2.0). Pull-down resistor values for PCI-X Mode 2 add-in cards are generally too close together for their values to be discriminated for a bus capable of operating in PCI-X Mode 2 if the PCIXCAP connections for more than one slot are bused together. 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.

IMPLEMENTATION NOTE

Changing $V_{I/O}$ When Changing Between Mode 1 and Mode 2

Since devices use the voltage level of $V_{I/O}$ to select between Mode 1 and Mode 2 I/O drive levels, the system must provide the appropriate setup time before the rising edge of RST# any time $V_{I/O}$ changes voltage levels. If the system changes from Mode 1 to Mode 2 or vice versa while the rest of the system is running, $V_{I/O}$ must be changed from 3.3V to 1.5V or vice versa, respectively.

To change from one mode to the other (ignoring any clock frequency change), the system must perform the following steps in order:

1. Assert RST#.
2. Change $V_{I/O}$ to the new value. The time required for $V_{I/O}$ to reach the new value is not specified.
3. Drive the appropriate PCI-X initialization pattern.
4. Wait for the appropriate length of time after $V_{I/O}$ is stable at the new value, and for the appropriate length of time after the initialization pattern has been driven, as specified in Section 2.1.2.4.2, "3.3V Environment Timing Parameters," in PCI-X EM 2.0.
5. Deassert RST#.

Note that the length of time that $V_{I/O}$ takes to reach the new value is not specified and varies from one implementation to another. Further note that there is no requirement for the other supply voltages either to change or to remain the same when $V_{I/O}$ changes. In some implementations, the other voltages remain stable. In other implementations, the other supply voltages go to zero. In still other implementation, the other supply voltages drop out of the specified limits for a brief period without going all the way to zero, and then return to the specified levels. In all cases, the system must wait for T_{pvrh}, as specified in Section 2.1.2.4.2, "3.3V Environment Timing Parameters," in PCI-X EM 2.0, after all the voltages are stable at the correct value before deasserting RST#.

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 following is an example of one way a PCI-X Mode 1 system and host bridge initializes the devices on its bus:

1. Assert RST# and 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 2.3). 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 add-in 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.

The following is an example of one way a PCI-X Mode 2 system and host bridge initialize the devices on its bus:

1. Sense the states of PCIXCAP for all devices on the bus and select the appropriate mode as described in Section 6.1.2 (except for selecting between conventional 33 MHz and conventional 66 MHz, which is not possible until step 3).
2. Assert RST# and apply power to all devices on the bus. If the mode is PCI-X Mode 2, set $V_{I/O}$ to 1.5V. Otherwise, set $V_{I/O}$ to 3.3V. 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 2.3). The pull-up resistors on these signals deassert them. To prevent AD, C/BE#, and ECC 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. (Note that the electrical terminators are disabled while RST# is asserted, which changes the DC characteristics of these signals.) When the power supply indicates that all of the supply voltages are within the proper tolerances, proceed to the next step.
3. If the mode is to be conventional PCI, sense the states of M66EN for all devices on the bus and select between PCI 33 and PCI 66.
4. If the mode is to be PCI 33, 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.
7. After the appropriate delay described in Section 4.1.2.3, the arbiter signals a bus turn-around alert to take the devices out of interface low-power state and grant the bus to the first owner.

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 systems 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 add-in 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 add-in card is found, no further action is required since the bus is already operating in that mode. However, if no conventional 33 MHz add-in 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 1.1, add-in cards cannot be connected to a bus whose clock is operating at a frequency higher than the add-in card can tolerate. Without connecting the add-in card to the bus, the Hot-Plug System Driver must determine the maximum frequency and mode capability of the add-in card. (Applying power to the add-in 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 1.1 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 of the PCIXCAP pin of each slot. The Hot-Plug System Driver must not connect a slot to a bus if the clock is too fast for the add-in card, or if the add-in card does not support the current operating mode of the bus.

If a PCI-X-capable add-in card is hot-inserted onto a bus that is operating in PCI-X mode at an acceptable frequency and mode, the Hot-Plug Controller must set $V_{I/O}$ and 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 2.3). 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

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 2.3. 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.

   Interrupt Disable: Implementation of this bit and the Interrupt Status bit in the Status register are optional but recommended for PCI-X Mode 1 devices and required for PCI-X Mode 2 devices. If one bit is implemented, both bits must be implemented.

   Fast Back-to-Back Enable: Ignored by the device in PCI-X mode.

   Parity Error Response: In parity mode, the function of this bit is defined in PCI 2.3. In ECC mode, this bit controls the device's response to uncorrectable ECC errors. If the bit is set, the device takes the action described in this specification for an uncorrectable ECC error. If the bit is cleared, the device records the error in the ECC Control and Status, ECC First Address, ECC Second Address, and ECC Attribute registers but in all other respects treats the transaction as if it had no error. Correctable ECC errors are corrected independent of the state of this bit.

   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.

   Interrupt Status: Implementation of this bit and the Interrupt Disable bit in the Command register are optional but recommended for PCI-X Mode 1 devices and required for PCI-X Mode 2 devices. If one bit is implemented, both bits must be implemented.

   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 5.2.1.

DEVSEL Timing: Indicates the device's conventional-PCI-mode DEVSEL# timing as defined in PCI 2.3 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 2.3. 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 2.12.1.1 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 2.3. System configuration software must initialize these registers as specified in PCI 2.3.

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 and ECC 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 as described above because range registers in PCI and PCI-X bridges for prefetchable memory are 64 bits 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 Capabilities List item in each function of the first device on the add-in card. That is, the connection of the PCIXCAP pin must indicate the add-in card is capable of operating in PCI-X mode if and only if the PCI-X Capabilities 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 the PCI-X Capabilities List item for PCI-X bridge functions. See Section 2.3.4, "PCIXCAP and MODE2 Connection," in PCI-X EM 2.0 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 is 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.

Bit location 0 is the least significant bit in each of the registers.

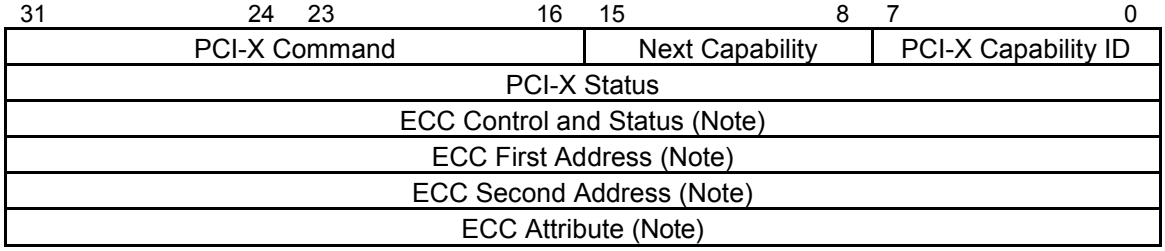

Figure 7-1: PCI-X Capabilities List Item for a Type 00h Configuration Header

Note:
These registers are included only in versions 1 and 2 of the PCI-X Capabilities List item.

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 2.3.

7.2.3. PCI-X Command Register

This register controls various modes and features of the PCI-X device.

Table 7-1: PCI-X Command Register

Bit Location	Description
0	**Uncorrectable Data Error Recovery Enable.** (read/write) The device driver sets this bit to enable the device to attempt to recover from uncorrectable data errors as described in Section 5.2.1.1. 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.

Bit Location	Description
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 Appendix B). 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.
3-2	**Maximum Memory Read Byte Count.** (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 D.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.) Register — Maximum Byte Count 0 — 512 1 — 1024 2 — 2048 3 — 4096 State after RST# is 0.

<table>
<tr><th>Bit Location</th><th>Description</th></tr>
<tr><td>6-4</td><td>
Maximum Outstanding Split Transactions. (read/write)
This register sets the maximum number of Split Transactions the device is permitted to have outstanding at one time as a requester. 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 RST# 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>11-7</td><td>Reserved</td></tr>
<tr><td>13-12</td><td>
PCI-X Capabilities List Item Version. (read only)
These bits indicate the format of the PCI-X Capabilities List item, and whether the device supports ECC in Mode 1.
ECC control and status bits appear in versions 1 and 2. Devices that do not support ECC use version 0. Devices that support ECC in Mode 2 but not in Mode 1 use version 1. Devices that support ECC both in Mode 2 and Mode 1 use version 2.
<table>
<tr><th>Register</th><th>Version</th><th>ECC Support</th><th>Capabilities List Item Size</th></tr>
<tr><td>00b</td><td>0</td><td>none</td><td>8 bytes</td></tr>
<tr><td>01b</td><td>1</td><td>Mode 2 only</td><td>24 bytes</td></tr>
<tr><td>10b</td><td>2</td><td>Modes 1 and 2</td><td>24 bytes</td></tr>
<tr><td>11b</td><td>reserved</td><td>reserved</td><td>reserved</td></tr>
</table>
</td></tr>
<tr><td>15-14</td><td>Reserved</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.

Bit Location	Description
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
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 add-in 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.

Bit Location	Description
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. All PCI-X 133, PCI-X 266 and PCI-X 533 devices must set this bit (see Section 6.1.1). The connection of the add-in 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 clock frequency is 66 MHz. 1 = The device's maximum clock 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 5.2.4. 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 5.2.5 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.
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

<table>
<tr><th>Bit Location</th><th>Description</th></tr>
<tr><td>22-21</td><td>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.
<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></td></tr>
<tr><td>25-23</td><td>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 as a requester. 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.
<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></td></tr>
</table>

<table>
<tr><th>Bit Location</th><th>Description</th></tr>
<tr><td>28-26</td><td>Designed Maximum Cumulative Read Size. (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 as a requester. 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>Received Split Completion Error Message. (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 5.2.6 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 error message received.
1 = A Split Completion error message has been received.</td></tr>
</table>

Bit Location	Description
30	**PCI-X 266 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 266 mode. All PCI-X 266 and PCI-X 533 devices set this bit. 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 PCI-X 266 operation. The connection of the add-in 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 is not capable of PCI-X 266 operation. (Not a PCI-X Mode 2 device.) 1 = The device is capable of PCI-X 266 operation. (PCI-X Mode 2 device.)
31	**PCI-X 533 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 533 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 PCI-X 533 operation. The connection of the add-in 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 is not capable of PCI-X 533 operation. 1 = The device is capable of PCI-X 533 operation

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 "Capable" Status Bits.

The 64-bit Device and speed capability bits (133 MHz Capable, PCI-X 266 Capable, PCI-X 533 Capable) 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, that is, the device connected directly to the PCI connector of the 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 add-in card.

A device designed for use both on 64-bit and 32-bit add-in cards must implement a method for the add-in 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.

PCI-X 266 and PCI-X 533 devices must be capable of operating in PCI-X Mode 1 at 133 MHz. (See Section 6.1.1.) Therefore, the 133 MHz Capable bit is set for all PCI-X 266 and PCI-X 533 devices. System management software and PCI-X hot-plug software written prior to publication of PCI-X 2.0 recognize and report PCI-X 266 and PCI-X 533 devices as PCI-X 133 devices.

The frequency indicated by the connection of the PCIXCAP pin must be consistent with the 133 MHz Capable, PCI-X 266 Capable, and PCI-X 533 Capable bits of the first device on the add-in card.

7.2.5. ECC Control and Status Register

This register provides additional information about ECC errors that occurred on the bus. Registers that store information from the failing transaction always store information directly from the bus (uncorrected), even if correction of the error is possible. This register is defined only in versions 1 and 2 of the PCI-X Capabilities List item.

In the following descriptions, contents of registers that are captured from the bus are described in their 64- or 32-bit forms. These bits are multiplexed in multiple phases or subphases on a 16-bit bus as described in Section 2.12.2.

Table 7-3: ECC Control and Status Register

Bit Location	Description
1-0	Reserved
2	**Additional Correctable ECC Error.** (write 1 to clear) This bit is set if the device detects a correctable ECC error, as described in Section 5.1.2, while error correction is enabled and the device is already indicating some other ECC error (i.e. the ECC Error Phase register is non-zero). Once set, this bit remains set until software writes a 1 to this location. State after RST# is 0. 0 = No additional correctable ECC error has been detected. 1 = One or more additional correctable ECC errors have been detected.
3	**Additional Uncorrectable ECC Error.** (write 1 to clear) This bit is set if the device detects an uncorrectable ECC error, or it detects a correctable error while error correction is disabled (as uncorrectable and correctable errors are described in Section 5.1.2), while the device is already indicating some other ECC error (i.e. the ECC Error Phase register is non-zero). Once set, this bit remains set until software writes a 1 to this location. State after RST# is 0. 0 = No additional uncorrectable ECC error has been detected. 1 = One or more additional uncorrectable ECC errors have been detected.

Bit Location	Description
6-4	**ECC Error Phase.** (write 1 to clear) If the device detects either a correctable or uncorrectable ECC error, as described in Section 5.1.2, this register indicates in which phase of the transaction the error occurred, and for data phase errors whether it was a 32-bit data error (seven-bit ECC) or 64-bit data error (eight-bit ECC). If this register is set to 0, the device is enabled to latch information about an ECC error. If the device detects an error, it latches the phase of the error in this register, and stores status information for the error in this register and in the ECC Address, and ECC Attribute registers. Writing a 1 to any of these bits clears this register and enables the device to capture the next error. State after RST# is 0. Register — ECC Error Phase 0 — No error 1 — First 32 bits of address 2 — Second 32 bits of address 3 — Attribute phase 4 — 32- or 16-bit data phase 5 — 64-bit data phase 6 — reserved 7 — reserved
7	**ECC Error Corrected.** (read-only) If the ECC Error Phase register is non-zero, this bit indicates whether the error that was captured was corrected. Correctable ECC errors that occur while error correction is enabled (see Disable ECC Correction bit) are the only errors that are corrected. If the ECC Error Phase register is zero, this bit is undefined. 0 = The error that was captured was not corrected. 1 = The error that was captured was corrected.
15-8	**Syndrome.** (read-only) The syndrome indicates information about the bit or bits that are in error, as described in Section 5.1.2.3. Bit — Syndrome 8 — E0 9 — E1 10 — E2 11 — E3 12 — E4 13 — E5 14 — E6 15 — E7 for 64-bit data, 0b for 32-bit data, or E16/Chk for 16-bit data

Bit Location	Description
19-16	**Error First (or only) Command.** (read-only) If the ECC Error Phase register is non-zero, this register indicates the contents of the C/BE[3::0]# bus for the first (or only) address phase of the transaction that included the error.
23-20	**Error Second Command.** (read-only) If the ECC Error Phase register is non-zero and the transaction that included the error used a dual address cycle, this register indicates the contents of the C/BE[3::0]# bus for the second address phase of the transaction that included the error.
27-24	**Error Upper Attributes.** (read-only) If the ECC Error Phase register is non-zero, this register indicates the contents of the C/BE[3::0]# bus for the attribute phase of the transaction that included the error.
28	**ECC Control Update Enable.** (write-transient) This bit always reads as a 0. If this bit is 1 in the data pattern being written, the Disable Single-Bit-Error Correction and ECC Mode bits are also updated (written). If this bit is 0 in the data pattern being written, the Disable Single-Bit-Error Correction and ECC Mode bits are not updated.
29	Reserved
30	**Disable Single-Bit-Error Correction.** (read/write) If the device is in ECC mode and this bit is 0, correctable errors (as described in Section 5.1.2.1) are corrected. If the device is in ECC mode and this bit is 1, correctable errors are not corrected and are treated as uncorrectable errors, including the setting of status bits and assertion of error indicator signals on the bus, as described in Sections 5.2.1 and 5.2.3. Disabling single-bit error correction enhances the error detection capability of the ECC as described in Section 5.1.2. In parity mode (ECC Mode bit is 0), this bit has no meaning and is ignored by the device. Writes to this register do not affect this bit unless the ECC Control Update Enable bit is a 1 in the data pattern being written. State after RST# is 0.
31	**ECC Mode.** (read/write in Mode 1, read-only in Mode 2) If this bit is 1, the device is in ECC mode. If this bit is 0, the device is in parity mode. Writes to this register do not affect this bit unless the ECC Control Update Enable bit is a 1 in the data pattern being written. The state of this bit after RST# is determined by the PCI-X initialization pattern (see Table 6-2). In PCI-X Mode 2, this bit is always a 1.

IMPLEMENTATION NOTE

ECC Control Update Enable.

The ECC Control Update Enable bit in the ECC Control and Status register has unique operational characteristics. Its value is never stored when the register is written. This enables software to perform read-modify write operations on the register without affecting the ECC Control bits.

When the ECC Control and Status register is written, the Disable Single-Bit-Error Correction and ECC Mode bits are updated only if the ECC Control Update Enable bit is 1 in the data pattern being written. This enables software to modify other bits in the register (e.g., to clear the "write 1 to clear" bits) without having to perform a read-modify-write operation or synchronize with other software that might be changing the Disable Single-Bit-Error Correction and ECC Mode bits. This feature is more significant in the ECC Control and Status register in PCI-X bridges (see Section 8.6.2.7), which includes other read/write bits. However, the behavior of the bits in this register of a non-bridge device are defined the same way to simplify system software design.

7.2.6. ECC Address Registers

There are two ECC address registers:

- ECC First 32 Bits of Address
- ECC Second 32 Bits of Address

These registers are defined only in versions 1 and 2 of the PCI-X Capabilities List item.

If the ECC Error Phase register is non-zero (indicating that an error has been captured), these registers indicate the contents of the AD[31::0] bus (for 64- and 32-bit buses) or the two phases of AD[31::16] bus (for 16-bit buses) for the address phase or phases of the transaction that included the error. If the ECC Error Phase register is zero, the contents of these registers are undefined.

The "First 32 Bits of Address" register records the least significant 32 bits of the address, regardless of the type, length, or width of the transaction, or the phase in which the error occurred. If the transaction used a dual address cycle, the "Second 32 Bits of Address" register records the most significant 32 bits of the address. If the transaction used a single address cycle, the contents of the "Second 32 Bits of Address" register are 0. Registers that store information from the failing transaction always store information directly from the bus (uncorrected), even if correction of the error is possible.

These registers are read-only.

7.2.7. ECC Attribute Register

If the ECC Error Phase register is non-zero (indicating that an error has been captured), the ECC Attribute register indicates the contents of the AD[31::0] bus (for 64- and 32-bit buses) or the two phases of AD[31::16] bus (for 16-bit buses) for the attribute phase of the transaction that included the error. If the ECC Error Phase register is zero, the contents of this register are undefined.

This register is defined only in versions 1 and 2 of the PCI-X Capabilities List item.

This register records the contents of the bus during the attribute phase, regardless of the type or length of the transaction, or the phase in which the error occurred. Registers that store information from the failing transaction always store information directly from the bus (uncorrected), even if correction of the error is possible.

This register is read-only.

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 2.3 recommends this mapping of the device into both address spaces.)

System configuration software in PC-compatible systems is discouraged from mapping devices into I/O ranges used by legacy functions (as defined in PCI 2.3), because these address are sometimes blindly probed by legacy operating system software.

7.4. Mode 2 Configuration Space

In PCI-X Mode 2, there is a total of 4096 bytes of Configuration Space for each device-function, as shown in Figure 7-2. The Base Configuration Space is the first 256 bytes and corresponds to the entire Configuration Space defined in PCI 2.3. The Extended Configuration Space is the next 3840 bytes.

The Base Configuration Space is accessible using either the mechanism defined in PCI 2.3 or the enhanced configuration access mechanism described in Section 7.4.1. All changes made using either access mechanism are equivalent. The Extended Configuration Space is accessible only using the enhanced configuration access mechanism.

PCI-X Mode 2 devices must respond to all configuration transactions (i.e., must assert DEVSEL#) anywhere in their Base Configuration Space or Extended Configuration Space. Offset 100h of Extended Configuration Space contains a header that serves either as the head of a list of Extended Capabilities List items or as a pattern indicating the absence of this list, as defined in Section 7.4.2. Devices are permitted to use Extended Configuration

Space both for Extended Capabilities List items and for device-specific purposes (after the header at offset 100h). However, any device-specific purpose that must be available in systems that do not support Extended Configuration Space (e.g., in a Mode 1 system or with an operating system designed for a Mode1 system) must use Base Configuration Space.

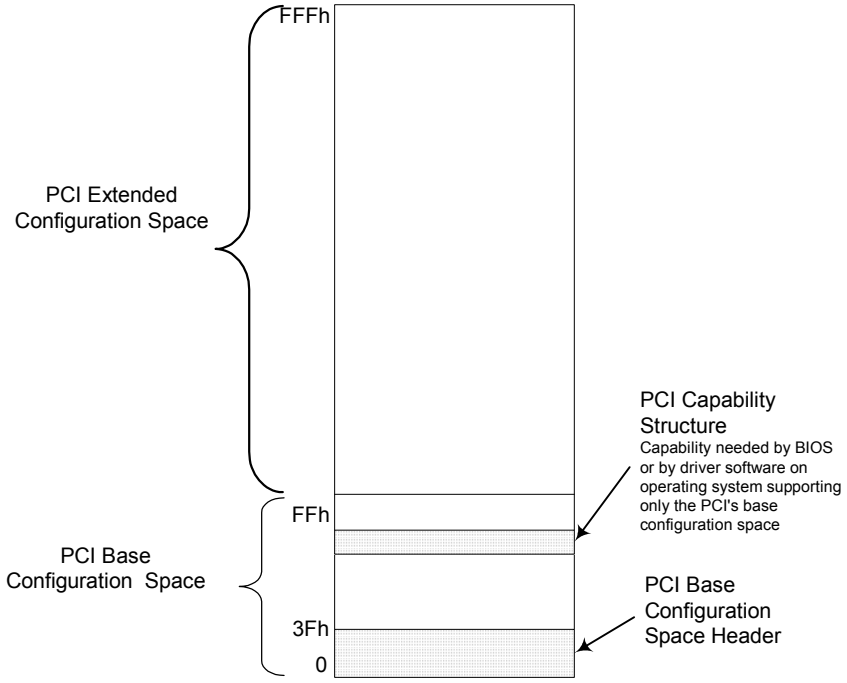

Figure 7-2: Mode 2 Configuration Space Layout

7.4.1. Enhanced Configuration Access Mechanism

The enhanced configuration access mechanism utilizes a flat memory-mapped address space to access device configuration registers. The memory address determines the configuration register accessed and the memory data updates (for a write) or returns the contents of (for a read) the addressed register.

PCI-X Mode 2 host bridges are required to translate the memory-mapped Configuration Space accesses from the host processor to configuration transactions. The mapping from memory address A[27::0] to Configuration Space is defined in Table 7-4. The base address A[63::28] is allocated in an implementation-specific manner and reported by the system firmware to the operating system.

The enhanced configuration access mechanism operates independently from the mechanism defined in PCI 2.3 for generation of configuration transactions. The logic in the host bridge that generates configuration transactions using one method must not interfere in any way with logic that generates configuration transaction using the other method.

Table 7-4: Configuration Address Mapping

Memory Address	Configuration Space
A[27:20]	Bus[7:0]
A[19:15]	Device[4:0]
A[14:12]	Function[2:0]
A[11:8]	Extended Register [3:0]
A[7:0]	Register[7:0]

7.4.2. Extended Capabilities List Items

Extended Capabilities List items are allocated using a linked list following a format resembling that defined in PCI 2.3 for Capabilities List items. Extended Capabilities List items are located in Extended Configuration Space (Configuration Space offset 100h or greater as shown in Figure 7-3.)

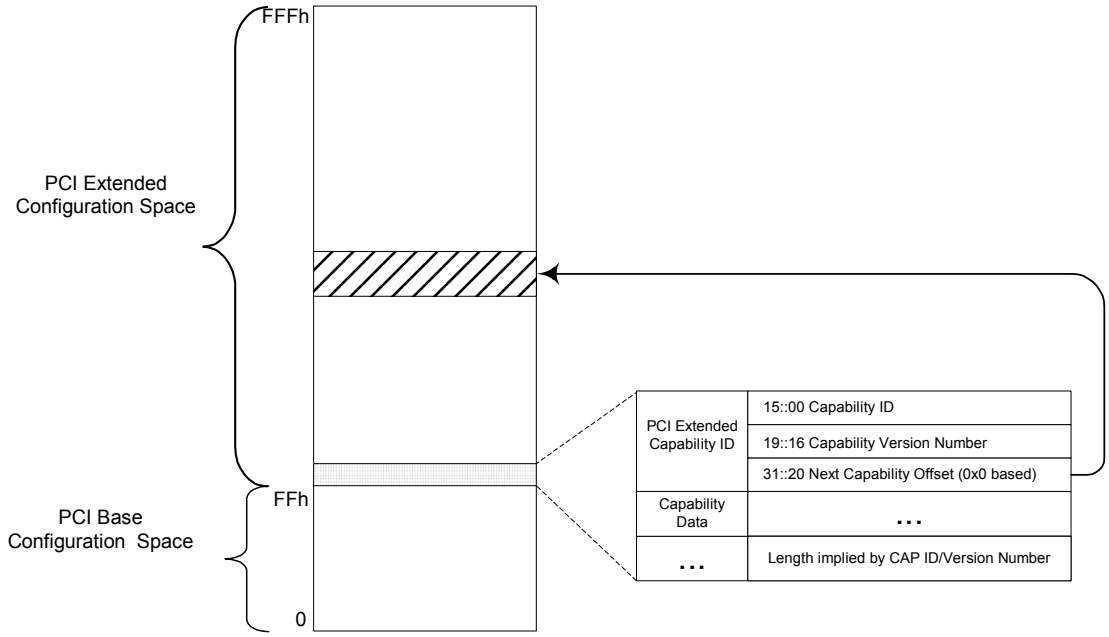

Figure 7-3: Extended Capabilities List Item Format

The first Extended Capabilities List item in device Configuration Space always begin at offset 100h. Absence of any Extended Capabilities List items is indicated by an Enhanced Capability Header with a Capability ID of 0000h, a Capability Version of 0h, and a Next Capability Offset of 0h.

Each Extended Capabilities List items begins with an Extended Capability Header aligned to a DWORD boundary as shown in Figure 7-4 and described in Table 7-5.

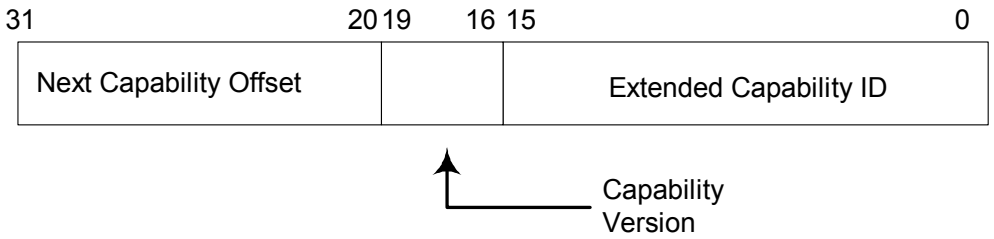

Figure 7-4: Extended Capability Header

Table 7-5: Extended Capability Header

Bit Location	Description
15:0	**Extended Capability ID.** (read-only) This field is a PCI-SIG defined ID number that indicates the nature and format of the Extended Capabilities List item.
19:16	**Capability Version.** (read-only) This field is a PCI-SIG defined version number that indicates the version of the Extended Capabilities List item present.
31:20	**Next Capability Offset.** (read-only) This field contains the offset to the next Extended Capabilities List item, or 000h if no other items exist in the linked list of Extended Capabilities List items. This offset is relative to the beginning of Base Configuration Space and thus must always be either 000h (for terminating the list) or greater than 0FFh. Since Extended Capabilities List items are always aligned to a DWORD boundary, the least significant two bits of the Next Capability Offset are always 00b.

8

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 operating in conventional mode, the bridge's behavior is governed by PCI 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 5.2.6

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 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. They do not support a 16-bit AD bus on the primary interface, and optionally support a 16-bit AD bus on the secondary 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 PCI 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 2.3, 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 8.4.7 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. Device ID Message transactions are forwarded based on the Route Type and Completer Bus Number (if applicable) fields in the DIM Address as described in Section 2.16.1.

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. (See Section 8.4.7 for the requirements to treat device ID message transactions the same as memory write transactions.) 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 8.7.1.1.2 for an exception for uncorrectable data 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. (See Section 8.4.7 for the requirements to treat device ID message transactions the same as 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 D.2 for recommendations for optimizing the setting of the Split Transaction Commitment Limit registers.

Unexpected Split Completion exceptions (see Section 5.2.5) 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:

$$TOST + NST \leq STCL$$

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 device.

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 8.7.1.2 and 8.7.1.3 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 2.3 and PCI Bridge 1.1. If both interfaces of a PCI-X bridge are operating in conventional mode, the bridge requirements are completely specified by PCI 2.3 and PCI 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 uncorrectable data 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.

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.

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

IMPLEMENTATION NOTE

Conventional PCI to PCI-X Prefetching

PCI-X bridges forwarding read transactions from an interface operating in conventional mode are generally subject to the same prefetching rules as conventional PCI bridges. PCI Bridge 1.1 specifically permits bridges to prefetch read data across a 4KB boundary, but notes that it is the responsibility of targets to disconnect a burst transaction if either a Base Address register boundary is reached, or a boundary is reached within a Base Address register range where the attributes of the access change (i.e., prefetchable vs. non-prefetchable). Furthermore, host bridges decoding addresses for main memory often have a similar issue at memory-page boundaries. Because each PCI-X transaction specifies its length, this responsibility shifts to the bridge issuing the transaction, if the bus is operating in PCI-X mode. Bridges are unlikely to have knowledge of any such boundaries; therefore, a bridge from a bus operating in conventional mode to a bus operating in PCI-X mode is strongly discouraged from prefetching across 4KB boundaries. Similarly, PCI-X targets that support memory resources that are designated as "prefetchable" in the Base Address register are encouraged to implement boundaries that are an integer multiple of 4KB.

Bridges that issue prefetch read transactions on an interface operating in PCI-X mode should be prepared to receive Split Completion Messages from the completer indicating a Byte Count Out of Range Completer Error (see Section 2.10.6.2) in lieu of the requested read data. Bridges should not treat this as an error condition but should instead cease prefetching on behalf of the conventional PCI transaction on the opposite interface. The bridge disconnects the conventional transaction after all prefetched data has been exhausted (if not terminated earlier by the initiator).

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 PCI 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 2.3 and PCI 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
Device ID Message	none

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 2.3, 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.

If the DIM Address of a transaction using the Device ID Message command is such that the bridge would have to forward it to an interface operating in conventional mode, the bridge either discards the transaction or terminates it with Target-Abort depending on the state of the Silent Drop bit in the DIM Address, as described in Section 2.16.

8.4.3.2.2. Split Transaction to Delayed Transaction Conversion

If the PCI-X bridge forwards a transaction other than a burst push transaction 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 PCI 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 burst push transaction 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 Section 2.5 and Appendix B 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 Appendix B 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 Appendix B 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.

For ordering and passing purposes, transactions using the Device ID Message command are treated by bridges the same as memory write transactions with the Relaxed Order bit cleared (not relaxed).

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, memory write transactions and device ID message transactions must not pass any other memory write or device ID message transactions. (Same as conventional PCI.)

A2b — For host bridges when the Relaxed Ordering attribute bit is set, that memory write or device ID message transaction is permitted to pass all previously posted memory write or device ID message transactions 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 2.3 for conventional bridges to accept memory writes even while executing Delayed Transactions.)

For purposes of signaling Retry and Disconnect at Next ADB, and for buffering, a bridge treats a device ID message transaction as if it were a memory write transaction.

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 2.3 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. See Section 8.4.7 for the requirement to treat device ID message transactions the same as memory write transactions.

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.

8.4.7. Forwarding Device ID Messages

If both interfaces of a bridge are operating in PCI-X mode and the bridge receives a device ID message transaction on one interface that it must forward to the other, it forwards the transaction as if it were a memory write transaction that started on an ADB. That is, the bridge does the following:

- If buffer space is not available to accept a memory write transaction as described in Section 8.4.5, the bridge signals Retry to the device ID message transaction.
- If buffer space is available to accept a memory write transaction, the bridge asserts DEVSEL# on the originating interface and begins accepting data phases (immediate completion).
- For disconnection purposes, the bridge treats the device ID message transaction on the originating bus as it would a memory write transaction that begins on an ADB. If buffer space for device ID message and memory write transactions fills, the bridge signals Disconnect at Next ADB as described in Section 2.11.2.2. The transaction also ends on any data phase in which the byte count is satified or when the initiator disconnects it as it would a memory write transaction that begins on an ADB.
- If the destination bus is operating in conventional PCI mode, the bridge either discards the device ID message transaction or terminates it with Target-Abort depending on the state of the Silent Drop bit in the DIM Address, as described in Section 2.16.
- The bridge treats device ID message transactions as it would a memory write transaction for all ordering rules.
- When the ordering rules have been satisfied, the bridge requests the destination bus and initates the device ID message Sequence.
    - For purposes of disconnecting the device ID message transaction on the destination bus, the bridge treats the transaction as it would a memory write transaction that begins on an ADB.
    - See Section 2.16 for the requirement for the bridge to discard transactions that terminate with Master-Abort if the Silent Drop bit is 1, and to treat all other Master-Abort and all Target-Abort cases as it would a memory write transaction.

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.

Exclusive access is supported only on 64- and 32-bit buses. There are no subordinate bridges on a 16-bit bus, so all devices on a 16-bit ignore LOCK#. (See Section 2.12.2.)

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.
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#

---

[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.

[4] Consecutive refers to back-to-back exclusive accesses and not a continuation of the current exclusive access.

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 in PCI-X Mode 1. Starting an exclusive access in Mode 2 is identical, except for drive and float requirements described in Section 2. 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.

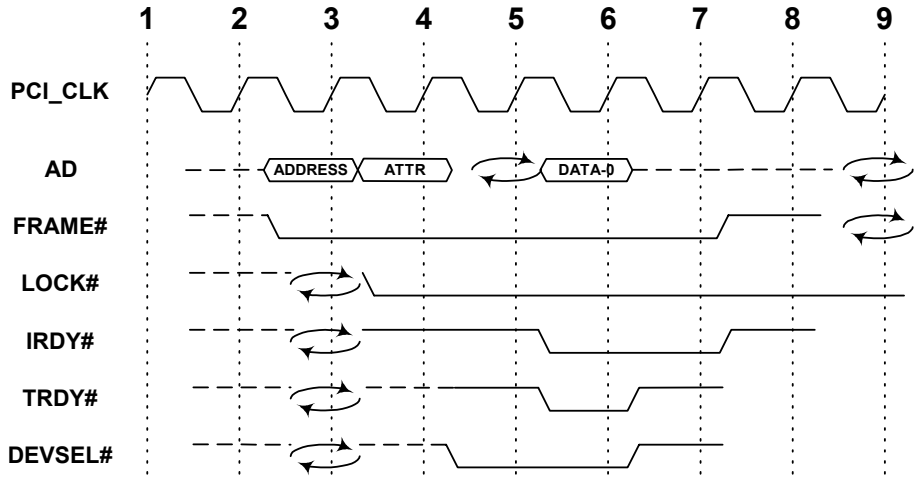

Figure 8-1: Starting an Exclusive Access with an Immediate Transaction, Mode 1

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 in PCI-X Mode 1. Starting an exclusive access in Mode 2 is identical, except for drive and float requirements described in Section 2.

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.

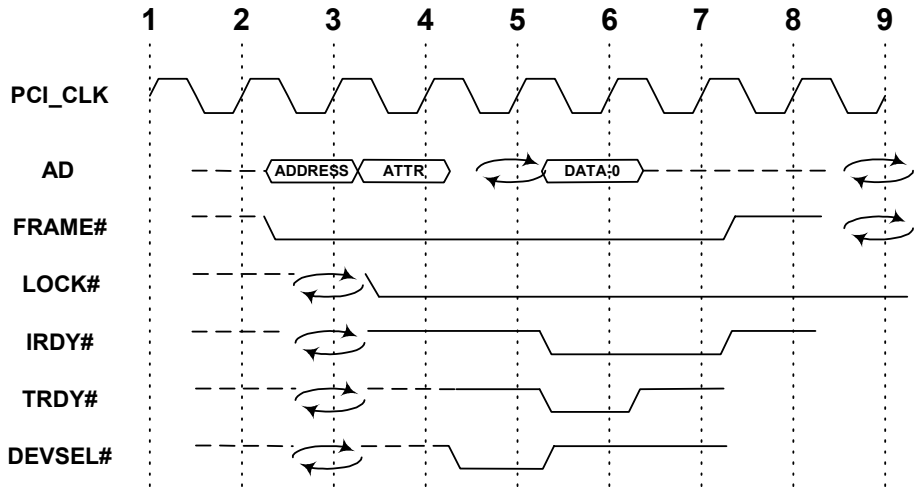

Figure 8-2: Starting an Exclusive Access with a Split Transaction, Mode 1

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 in PCI-X Mode 1 continuing an exclusive access, and the completer (on the same bus segment) executing the transaction as an Immediate Transaction. Continuing an exclusive access in Mode 2 is identical, except for drive and

float requirements described in Section 2. 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.

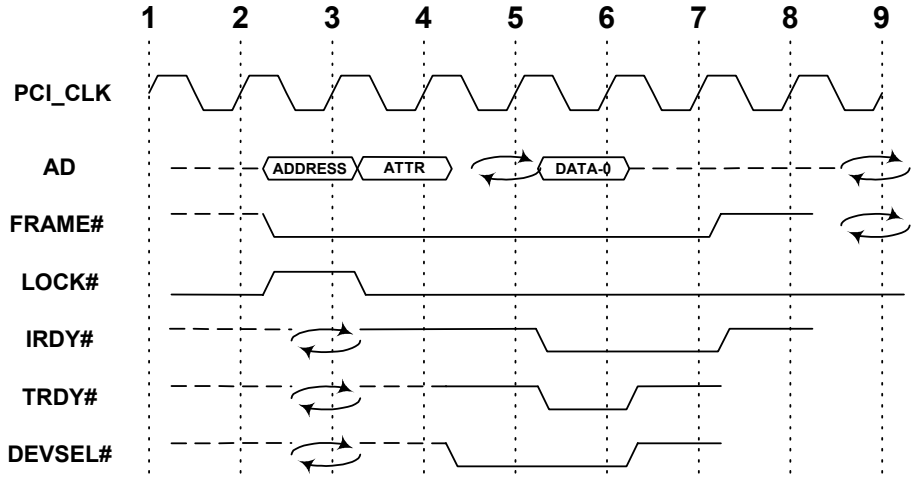

Figure 8-3: Continuing an Exclusive Access, Immediate Transaction, Mode 1

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.

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 in PCI-X Mode 1 trying a non-exclusive, downstream access (other than a Split Completion) to a locked bridge. Such a transaction in Mode 2 is identical, except for drive and float requirements described in Section 2. 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.

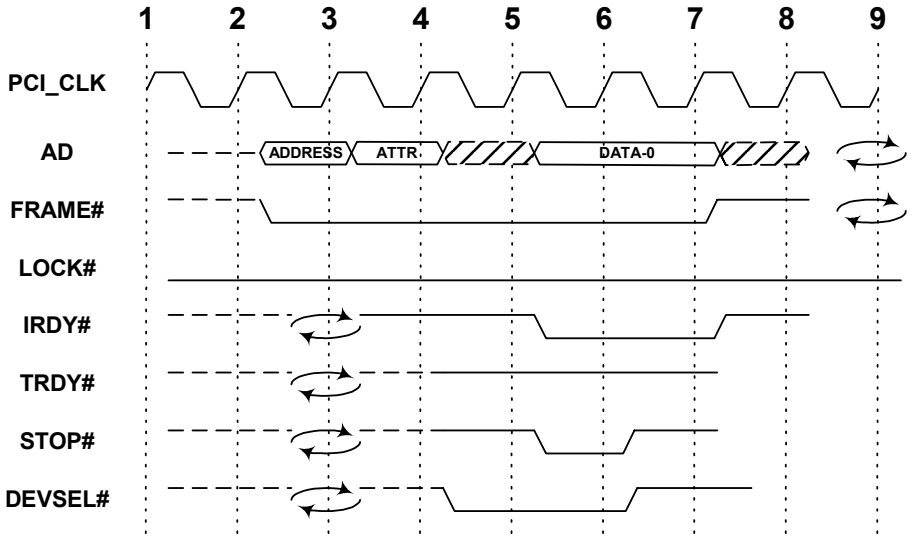

Figure 8-4: Accessing a Locked Downstream Bridge, Mode 1

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 PCI 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 2.3. The Prefetchable bit must be set unless the range contains locations with read side effects. (See Section 2.12.1.1 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 2.3.
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.

3. 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 5.2.1.

   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 PCI Bridge 1.1 and include a PCI-X Capabilities List item as shown in Figure 8-5.

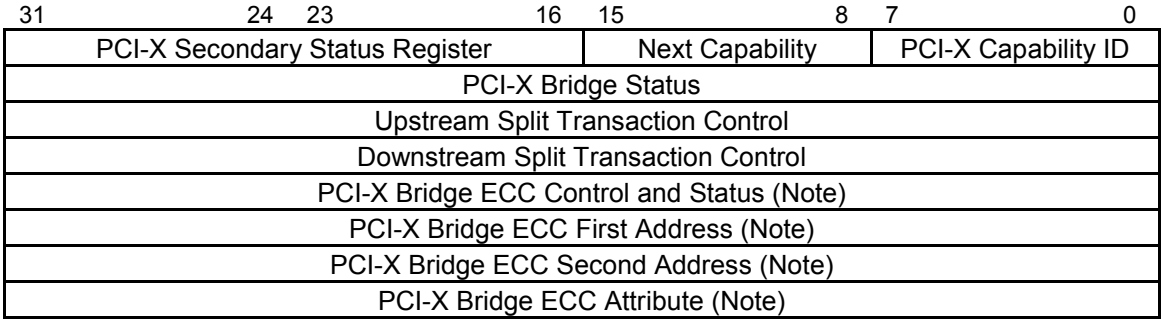

Figure 8-5: PCI-X Capabilities List Item for a Type 01h Configuration Header

Note:
These registers are included only in versions 1 and 2 of the PCI-X Capabilities List item.

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 Capabilities List item in the first bridge on the add-in card. That is, the connection of the PCIXCAP pin must indicate the add-in card

is capable of operating in PCI-X mode if and only if the PCI-X Capabilities 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 Capabilities List item. See Section 7.2 for the PCI-X Capabilities List item for non-bridge functions. See Section 2.3.4, "PCIXCAP and MODE2 Connection," in PCI-X EM 2.0 for the connection of the PCIXCAP pin.

Bit location 0 is the least significant bit in each of the registers.

IMPLEMENTATION NOTE

PCI-X Bridge Registers Optimized for Forwarding Transactions

The PCI-X Bridge Capabilities 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 uncorrectable data 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 Capabilities 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 2.3.

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. If the interface is capable of PCI-X 266 or PCI-X 533 operation, it is also capable of PCI-X 133 operation (see Section 6.1.2). 0 = The maximum clock frequency is 66 MHz. 1 = The maximum clock 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 8.7.1.2 and 8.7.1.3 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 5.2.5 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
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 D.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.
5	**Split Request Delayed.** (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 D.2 for more details. Once set, the bit remains set until software writes a 1 to this location. State after RST# is 0. 0 = The bridge has not delayed a Split Request. 1 = The bridge has delayed a Split Request.

<table>
<tr><th>Bit Location</th><th>Description</th></tr>
<tr><td>9-6</td><td>
Secondary Bus Mode and Frequency. (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 RST# 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 RST# was asserted.
<table>
<tr><th>Reg</th><th>Mode</th><th>Max Clock Frequency (MHz) (ref)</th><th>Minimum Clock Period (ns)</th></tr>
<tr><td>0h</td><td>conventional</td><td>N/A</td><td>N/A</td></tr>
<tr><td>1h</td><td>PCI-X Mode 1</td><td>66</td><td>15</td></tr>
<tr><td>2h</td><td>PCI-X Mode 1</td><td>100</td><td>10</td></tr>
<tr><td>3h</td><td>PCI-X Mode 1</td><td>133</td><td>7.5</td></tr>
<tr><td>4h</td><td>PCI-X Mode 1</td><td>reserved</td><td>reserved</td></tr>
<tr><td>5h</td><td>PCI-X Mode 1</td><td>reserved</td><td>reserved</td></tr>
<tr><td>6h</td><td>PCI-X Mode 1</td><td>reserved</td><td>reserved</td></tr>
<tr><td>7h</td><td>PCI-X Mode 1</td><td>reserved</td><td>reserved</td></tr>
<tr><td>8h</td><td>PCI-X 266 (Mode 2)</td><td>reserved</td><td>reserved</td></tr>
<tr><td>9h</td><td>PCI-X 266 (Mode 2)</td><td>66</td><td>15</td></tr>
<tr><td>Ah</td><td>PCI-X 266 (Mode 2)</td><td>100</td><td>10</td></tr>
<tr><td>Bh</td><td>PCI-X 266 (Mode 2)</td><td>133</td><td>7.5</td></tr>
<tr><td>Ch</td><td>PCI-X 533 (Mode 2)</td><td>reserved</td><td>reserved</td></tr>
<tr><td>Dh</td><td>PCI-X 533 (Mode 2)</td><td>66</td><td>15</td></tr>
<tr><td>Eh</td><td>PCI-X 533 (Mode 2)</td><td>100</td><td>10</td></tr>
<tr><td>Fh</td><td>PCI-X 533 (Mode 2)</td><td>133</td><td>7.5</td></tr>
</table>
An equivalent feature is required for host bridges.
</td></tr>
<tr><td>11-10</td><td>Reserved</td></tr>
<tr><td>13-12</td><td>
PCI-X Capabilities List Item Version. (read only)
These bits indicate the format of the PCI-X Capabilities List item, and whether the bridge supports ECC in Mode 1.
ECC control and status bits appear in versions 1 and 2. Bridges that do not support ECC use version 0. Bridges that support ECC in Mode 2 but not in Mode 1 use version 1. Bridges that support ECC both in Mode 2 and Mode 1 use version 2.
<table>
<tr><th>Register</th><th>Version</th><th>ECC Support</th><th>Capabilities List Item Size</th></tr>
<tr><td>00b</td><td>0</td><td>none</td><td>16 bytes</td></tr>
<tr><td>01b</td><td>1</td><td>Mode 2 only</td><td>32 bytes</td></tr>
<tr><td>10b</td><td>2</td><td>Modes 1 and 2</td><td>32 bytes</td></tr>
<tr><td>11b</td><td>reserved</td><td>reserved</td><td>reserved</td></tr>
</table>
</td></tr>
</table>

Bit Location	Description
14	**PCI-X 266 Capable.** (read-only) This bit indicates that the bridge's secondary interface is capable of PCI-X 266 operation. 0 = The secondary interface is not capable of PCI-X 266 operation. 1 = The secondary interface is capable of PCI-X 266 operation.
15	**PCI-X 533 Capable.** (read-only) This bit indicates that the bridge's secondary interface is capable of PCI-X 533 operation. 0 = The secondary interface is not capable of PCI-X 533 operation. 1 = The secondary interface is capable of PCI-X 533 operation.

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).

Bit Location	Description
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.
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 add-in 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.

Bit Location	Description
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. All PCI-X 133, PCI-X 266 and PCI-X 533 devices must set this bit (see Section 6.1.1). The connection of the add-in 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 clock frequency is 66 MHz. 1 = The maximum clock 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 8.7.1.2 and 8.7.1.3 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 5.2.5 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 D.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 D.2 for more details. Once set, the bit remains set until software writes a 1 to this location. State after RST# is 0. 0 = The bridge has not delayed a Split Request. 1 = The bridge has delayed a Split Request.
28-22	Reserved
29	**Device ID Messaging Capable.** (read-only) This bit indicates that the bridge is capable of forwarding Device ID Message transactions. All PCI-X Mode 2 bridges must set this bit. PCI-X Mode 1 bridges optionally set this bit.

Bit Location	Description
30	**PCI-X 266 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 266 mode. All PCI-X 266 and PCI-X 533 bridges set this bit. 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 PCI-X 266 operation. The connection of the add-in 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 bridge is not capable of PCI-X 266 operation. (Not a PCI-X Mode 2 bridge.) 1 = The bridge is capable of PCI-X 266 operation. (PCI-X Mode 2 bridge.)
31	**PCI-X 533 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 533 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 is capable of PCI-X 533 operation. The connection of the add-in 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 bridge is not capable of PCI-X 533 operation. 1 = The bridge is capable of PCI-X 533 operation.

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 D.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 D.2 for more details and setting recommendations. State after RST# is the same as the Split Transaction Capacity register.

8.6.2.7. PCI-X Bridge ECC Control and Status Register

This register provides information about ECC errors detected by the bridge while operating in ECC mode. The register displays and controls either primary or secondary interface information according to the Select Secondary ECC Registers bit. This register is defined only in versions 1 and 2 of the PCI-X Capabilities List item.

Registers that store information from the failing transaction always store information directly from the bus (uncorrected), even if correction of the error is possible.

Table 8-8: PCI-X Bridge ECC Control and Status Register

Bit Location	Description
0	**Select Secondary ECC Registers.** (read/write) There is a single Select Secondary ECC Registers bit that controls reading and writing of both primary and secondary ECC registers in the bridge. If this bit is 1, reading from the ECC error logging registers (PCI-X Bridge ECC Control and Status, PCI-X Bridge ECC First Address, PCI-X Bridge ECC Second Address, and PCI-X Bridge ECC Attribute registers) reads the values latched for the secondary interface. If the bit is 1 in the data pattern being written to the PCI-X Bridge ECC Control and Status register, the PCI-X Bridge ECC Control and Status register for the secondary interface is affected. If this bit is 0, reading from the ECC error logging registers reads the values latched for the primary interface. If this bit is 0 in the data pattern being written to the PCI-X Bridge ECC Control and Status register, the PCI-X Bridge ECC Control and Status register for the primary interface is affected. State after RST# is 0.
1	**Error Present in Other ECC Register Bank.** (read-only) If this bit is 1, the ECC error logging registers for the other interface hold information about an ECC error. That is, if the Select Secondary ECC Registers bit is 1, the primary ECC error logging registers hold information about an ECC error for the primary interfaced. If the Select Secondary ECC Registers bit is 0, the secondary ECC error logging registers hold information about an ECC error for the secondary interfaced.
2	**Additional Correctable ECC Error.** (write 1 to clear) This bit is set if the bridge detects a correctable ECC error, as described in Section 5.1.2, while the device is already indicating some other ECC error on the same interface (i.e. the ECC Error Phase register is non-zero). Once set, this bit remains set until software writes a 1 to this location. State after RST# is 0. 0 = No additional correctable ECC error has been detected. 1 = One or more additional correctable ECC errors have been detected.

Bit Location	Description
3	**Additional Uncorrectable ECC Error.** (write 1 to clear) This bit is set if the bridge detects an uncorrectable ECC error, as described in Section 5.1.2, while the device is already indicating some other ECC error on the same interface (i.e. the ECC Error Phase register is non-zero). Once set, this bit remains set until software writes a 1 to this location. State after RST# is 0. 0 = No additional uncorrectable ECC error has been detected. 1 = One or more additional uncorrectable ECC errors have been detected.
6-4	**ECC Error Phase.** (write 1 to clear) If the bridge detects either a correctable or uncorrectable ECC error, as described in Section 5.1.2, this register indicates in which phase of the transaction the error occurred, and for data phase errors whether it was a 32-bit data error (7-bit ECC) or 64-bit data error (8-bit ECC). If this register is set to 0, the bridge is enabled to latch information about an ECC error. If the device detects an error, it latches the phase of the error in this register, and stores status information for the error in the ECC Status, ECC Address, and ECC Attribute registers. Writing a 1 to any of these bits clears this register and enables the device to capture the next error. State after RST# is 0. Register — ECC Error Phase 0 — No error 1 — First 32 bits of address 2 — Second 32 bits of address 3 — Attribute phase 4 — 32- or 16-bit data phase 5 — 64-bit data phase 6 — reserved 7 — reserved
7	**ECC Error Corrected.** (read-only) If the ECC Error Phase register is non-zero, this bit indicates whether the error that was captured was corrected. Correctable ECC errors that occur on either primary or secondary interfaces while the Disable Single-Bit-Error Correction bit for that interface is 0 are the only errors that are corrected. If the ECC Error Phase register is zero, this bit is undefined. 0 = The error that was captured was not corrected. 1 = The error that was captured was corrected.

Bit Location	Description
15-8	**Syndrome.** (read-only) The syndrome indicates information about the bit or bits that are in error, as described in Section 5.1.2.3. Bit — Syndrome 8 — E0 9 — E1 10 — E2 11 — E3 12 — E4 13 — E5 14 — E6 15 — E7 for 64-bit data, 0b for 32-bit data, or E16/Chk for 16-bit data
19-16	**Error First (or only) Command.** (read-only) If the ECC Error Phase register is non-zero, this register indicates the contents of the C/BE[3::0]# bus for the first (or only) address phase of the transaction that included the error.
23-20	**Error Second Command.** (read-only) If the ECC Error Phase register is non-zero and the transaction that included the error used a dual address cycle, this register indicates the contents of the C/BE[3::0]# bus for the second address phase of the transaction that included the error.
27-24	**Error Upper Attributes.** (read-only) If the ECC Error Phase register is non-zero, this register indicates the contents of the C/BE[3::0]# bus for the attribute phase of the transaction that included the error.
28	**ECC Control Update Enable.** (write-transient) This bit always reads as a 0. If this bit is 1 in the data pattern being written, the Disable Single-Bit-Error Correction and ECC Mode bits are also updated (written). If this bit is 0 in the data pattern being written, the Disable Single-Bit-Error Correction and ECC Mode bits are not updated.
29	Reserved

Bit Location	Description
30	**Disable Single-Bit-Error Correction.** (read/write) If the appropriate interface of the bridge is in ECC mode and this bit is 0, correctable errors (as described in Section 5.1.2.1) that occur on that interface are corrected. If the appropriate interface of the bridge is in ECC mode and this bit is 1, correctable errors that occur on that interface are not corrected and are treated as uncorrectable errors, including the setting of status bits and assertion of error indicator signals on the bus, as described in Sections 5.2.1 and 5.2.3. Disabling single-bit error correction enhances the error detection capability of the ECC as described in Section 5.1.2. In the appropriate interface of the bridge is in parity mode (ECC Mode bit is 0), this bit has no meaning and is ignored by the bridge. Writes to this register do not affect this bit unless the ECC Control Update Enable bit is a 1 in the data pattern being written. State after RST# is 0.
31	**ECC Mode.** (read/write in Mode 1, read-only in Mode 2) If this bit is 1, the appropriate interface of the bridge is in ECC mode. If this bit is 0, the appropriate interface of the bridge is in parity mode. Writes to this register do not affect this bit unless the ECC Control Update Enable bit is a 1 in the data pattern being written. The state of this bit after RST# is determined by the PCI-X initialization pattern (see Table 6-2) for the appropriate interface. In PCI-X Mode 2, this bit is always a 1.

IMPLEMENTATION NOTE

Select Secondary ECC Registers.

The Select Secondary ECC Registers bit in the PCI-X Bridge ECC Control and Status register has unique operational characteristics. When reading from the PCI-X Bridge ECC Control and Status, PCI-X Bridge ECC First Address, PCI-X Bridge ECC Second Address, and PCI-X Bridge ECC Attribute registers, the state of this bit acts as a selector between primary and secondary registers. Since this bit is in the PCI-X Bridge ECC Control and Status register, this bit position always reads as a 1 when reading the secondary interface information and always reads as a 0 when reading the primary interface information.

When the PCI-X Bridge ECC Control and Status register is written, the value in this bit position in the data pattern being written determines whether secondary or primary control and status information is affected. That is, if this bit position in the data pattern being written is a 1, secondary control and status information is affected. If this bit position in the data pattern being written is a 0, primary control and status information is affected.

8.6.2.8. PCI-X Bridge ECC Address Registers

There are two ECC address registers. The registers display secondary interface information if the Select Secondary ECC Registers bit is set in the PCI-X Bridge Status register. Otherwise, the register displays primary interface information.

- ECC First 32 Bits of Address
- ECC Second 32 Bits of Address

These registers are defined only in versions 1 and 2 of the PCI-X Capabilities List item.

If the ECC Error Phase registers for the appropriate interface is non-zero (indicating that an error has been captured), these registers indicates the contents of the AD[31::00] bus (for 64- and 32-bit buses) or the two phases of AD[31::16] bus (for 16-bit secondary buses) for the address phase or phases of the transaction that included the error. If the ECC Error Phase registers for the appropriate interface is zero, the contents of these registers are undefined.

The "First 32 Bits of Address" register records the least significant 32 bits of the address, regardless of the type, length, or width of the transaction, or the phase in which the error occurred. If the transaction used a dual address cycle, the "Second 32 Bits of Address" register records the most significant 32-bit of the address. If the transaction used a single address cycle, the contents of the "Second 32 Bits of Address" register are 0. Registers that store information from the failing transaction always store information directly from the bus (uncorrected), even if correction of the error is possible.

These registers are read-only.

8.6.2.9. PCI-X Bridge ECC Attribute Register

This register displays secondary interface information if the Select Secondary ECC Registers bit is set in the PCI-X Bridge Status register. Otherwise, the register displays primary interface information. This register is defined only in versions 1 and 2 of the PCI-X Capabilities List item.

If the ECC Error Phase registers for the appropriate interface is non-zero (indicating that an error has been captured), the ECC Attribute register indicates the contents of the AD[31::00] bus (for 64- and 32-bit buses) or the two phases of AD[31::16] bus (for 16-bit secondary buses) for the attribute phase of the transaction that included the error. If the ECC Error Phase registers is zero, the contents of this register are undefined.

This register records the contents of the bus during the attribute phase, regardless of the type or length of the transaction, or the phase in which the error occurred. Registers that store information from the failing transaction always store information directly from the bus (uncorrected), even if correction of the error is possible.

This register is read-only.

8.7. PCI-X Bridge Error Support

PCI-X Mode 1 bridges optionally provide ECC protection for both interfaces. PCI-X Mode 2 bridges are required to provide ECC protection for both the primary and secondary interfaces when those interfaces are operating in Mode 2. PCI-X Mode 2 bridges optionally provide ECC protection to an interface operating in Mode 1. If a Mode 1 or a Mode 2 bridge provides ECC support to one interface when operating in Mode 1, it must provide ECC support to both interfaces when those interfaces are operating in Mode 1.

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.

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. Uncorrectable Data Errors

If a PCI-X bridge detects an uncorrectable data error, in most cases, it forwards the transaction with the error. The error forwarding requirements vary according to the following:

- The protection protocol (parity or ECC) on each interface.
- The width of the transaction on each interface.
- The type of error, i.e., parity, correctable ECC, or uncorrectable ECC. For correctable ECC errors, the forwarding method is controlled as described in Sections 8.6.2.3, and 8.6.2.4.
- For 64-bit data, the bus segment (upper or lower) on which the error occurred.

In the following discussion, the phrase "drives bad parity or ECC" is used to describe the error forwarding protocols specified on a case-by-case basis in Section 8.7.3.

8.7.1.1.1. Uncorrectable Data Error on an Immediate Read

If the bridge detects an uncorrectable data 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 5.1.1.2 for that interface. When the bridge creates the Split Completion and returns it to the requester, the bridge drives bad

parity or ECC. The uncorrectable read data error does not affect the bridge's behavior in any other way. After an uncorrectable data 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.

8.7.1.1.2. Uncorrectable Data Error on a Non-Posted Write

If the bridge detects an uncorrectable data 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 5.2.1 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 or ECC. The bridge must not signal Retry or Target-Abort solely because of an uncorrectable data error.

IMPLEMENTATION NOTE

Discarding or Forwarding a Non-Posted Write with an Uncorrectable Data Error

If a non-posted write transaction has an uncorrectable data 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 or ECC is sampled and checked.

Signaling Split Response before write data parity or ECC 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 5.2.1 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.)

8.7.1.1.3. Uncorrectable Data Error on a Split Completion

If the bridge detects an uncorrectable data 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 5.2.1 for that interface. The bridge then

drives bad parity or ECC when it forwards the Split Completion. The bridge takes no other action on that uncorrectable data error.

If the bridge detects an uncorrectable data 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 5.2.1 for that interface. The bridge takes no other action on that uncorrectable data error.

Except as noted in Section 5.2.6, 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.

8.7.1.1.4. Uncorrectable Data Error on a Posted Write

PCI-X bridge behavior for uncorrectable data errors on posted write transactions is the same as for conventional PCI bridges. If the bridge detects an uncorrectable data 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 5.2.1 for that interface. The bridge then forwards the posted write transaction and drives bad parity or ECC. The bridge takes no other action on that uncorrectable data 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 5.2.1 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 5.2.1 for that interface and asserts SERR# (if enabled).

8.7.1.2. 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 burst push transactions, the bridge sets the appropriate Received Master-Abort status bit (as specified in PCI 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 PCI 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 PCI 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 PCI 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.

Bridges treat device ID messages in the same manner as memory write transactions when the Silent Drop bit in the DIM address is 0. See Section 2.16 for the requirements for the bridge to discard device ID message transactions that are terminated with Master-Abort on the destination bus when the Silent Drop bit is 1.

8.7.1.3. 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. Another example is when a PCI-X bridge is

unable to forward a device ID message due to the operating mode of the destination bus and the Silent Drop bit in the DIM Address is 0. 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 burst push transactions, the bridge sets the appropriate Received Target-Abort status bit (as specified in PCI 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 PCI 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 PCI 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 PCI 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.

Bridges treat device ID messages in the same manner as memory write transactions when the Silent Drop bit in the DIM Address is 0. See Section 2.16 for the requirements for the bridge to discard device ID message transactions that are terminated with Target-Abort on the destination bus when the Silent Drop bit is 1.

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 PCI Bridge 1.1 in all cases except as follows:

- Errors that occur in the PCI-X environment and are reported to the bridge in the form of a Split Completion Message.
- Generation of the ECC signature if the destination bus is protected by ECC. If the bridge detects a parity error on data received on the originating (parity protected) interface, the generated ECC signature is "poisoned" as described in Section 8.7.3.3.

Note that if the bridge detects a parity error on the originating interface and the transaction is an I/O write, the bridge discards the transaction as described in PCI Bridge 1.1. 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 an uncorrectable write data error (i.e., PCI-X Bridge class and Uncorrectable Write Data 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.

8.7.3. Forwarding Data-Phase Parity and ECC Errors

Various cases described in Sections 8.7.1 and 8.7.2 require the bridge to forward a data phase with an error. The protocols for forwarding those data phase errors are specified on a case-by-case basis in this section. Forwarding the error as specified here enables the error to be accurately returned to the requester so the requester can attempt to recover or report the error to the system. The error forwarding protocols are designed to ensure that an error detected at one bridge flows (in the absence of further errors) correctly and accurately through any other bridges in the path to the requester.

8.7.3.1. Parity to Parity

If both interfaces of the bridge are protected by parity and the bridge detects a parity error on data received at one interface, the bridge drives AD, C/BE#, PAR, and (if the transfer width is 64 bits on the destination bus) PAR64 on the destination bus exactly as observed on the originating bus, including the parity error(s). This action is summarized in summarized in Table 8-9.

Table 8-9: Bridge Data-Phase Forwarding Actions, Parity to Parity

Orig I/F	Dest I/F	Error Case	Forwarding Action	
			Data	Parity
Parity Protected 64-bit *possible errors:* *PAR bad* *PAR64 bad* *Both bad*	Parity protected 64-bit	No Error	Data as is (good)	PAR and PAR64 good parity
		Upper Parity Error (PAR64)	Data as is (bad)	PAR64 bad parity, PAR good parity
		Lower Parity Error (PAR)		PAR bad parity, PAR64 good parity
		Both Parity Error		PAR64 bad parity, PAR bad parity
	Parity protected 32-bit	No Error	Data as is (good)	PAR good parity
		Upper Parity Error (PAR64)	Data as is (bad)	PAR bad parity for the DWORD with the error
		Lower Parity Error (PAR)		
		Both Parity Error		PAR bad parity for both DWORDs
Parity protected 32-bit *possible errors:* *PAR bad*	Parity protected 64-bit	No Error	Data as is (good)	PAR and PAR64 good parity
		Parity Error AD2=1 (PAR)	Data as is (bad)	PAR64 bad parity, PAR good parity
		Parity Error AD2=0 (PAR)		PAR bad parity, PAR64 good parity
		Parity Error both (PAR)		PAR64 bad parity, PAR bad parity
	Parity protected 32-bit	No Error	Data as is (good)	PAR good parity
		PAR Parity Error	Data as is (bad)	PAR bad parity

8.7.3.2. ECC to ECC

If both interfaces of the bridge are protected by ECC, and correction is enabled, and the bridge detects a correctable ECC error on data received at one interface, the bridge corrects the error and continues as if the originally received codeword had no error.

If both interfaces of the bridge are protected by ECC and for data received on one interface the bridge detects an uncorrectable ECC error or the bridge detects a correctable ECC error when correction is disabled, the bridge does the following for each subphase with an error:

- Generate the ECC signature(s) for the destination bus using the data actually received (including the error).
- Modify ("poison") the ECC signature(s) to indicate that there is a forwarded uncorrectable error. Poisoning is done by including (using XOR) one of the error signatures from Table 5-4, selected as follows:
  - For 32-bit data (on the destination bus), use the PEL signature. If 64-bit data is received with an ECC error, both 32-bit phases are poisoned with PEL.
  - For 64-bit data (on the destination bus) and:
    - 32-bit data received with an ECC error when AD[2]=0 and correct ECC when AD[2]=1, use the PEL signature.
    - 32-bit data received with correct ECC when AD[2]=0 and an ECC error when AD[2]=1, use the PEH signature.
    - 32-bit data received with an ECC error both when AD[2]=0 and when AD[2]=1, use the PED signature.
    - 64-bit data received with an ECC error, use PED.
- Drive AD and C/BE# on the destination bus exactly as observed on the originating bus, including the error, for each data subphase.
- Drive the newly generated (poisoned) ECC signature(s) on ECC on the destination bus for each data subphase.

The resulting combination of AD, C/BE#, and ECC is still protected against single bit failures (if one occurs, the codeword appears to have a double bit error).

Note that the use of differentiated signatures for 64-bit data forwards the specific error indication and indicates the partial correctness of the data. The received codeword is used in any case where 64-bit parity-protected data, with only one of PAR or PAR64 incorrect, is used.

The inclusion (using XOR) of one of the PEL, PEH, or PED signatures in the ECC signature forces even parity in the forwarded codeword. (An error indication codeword always has even parity.)

This action is summarized in summarized in Table 8-10.

Table 8-10: Bridge Data-Phase Forwarding Actions, ECC to ECC

Orig I/F	Dest I/F	Error Case	Forwarding Action	
			Data	ECC
ECC protected 64-bit *possible errors:* *Correct-able* *PEL, PEH* *PED and other uncor-rectable* (Note)	ECC protected 64-bit	No Error	Data as is (good)	Generate 8-bit ECC
		Correctable	Corrected data	Generate 8-bit ECC
		PEH	Data as is (bad)	Generate 8-bit ECC with PEH inserted
		PEL		Generate 8-bit ECC with PEL inserted
		Other uncorrectable		Generate 8-bit ECC with PED inserted
	ECC protected 32-bit	No Error	Data as is (good)	Generate 7-bit ECC
		Correctable	Corrected data	
		PEH	Data as is (bad)	Generate 7-bit ECC with PEL inserted for the DWORD with the error
		PEL		
		Other uncorrectable		Gen 7-bit ECC with PEL both DWORDs
ECC protected 32-bit *possible errors* *Correct-able* *PEL and other uncor-rectable* (Note)	ECC protected 64-bit	No Error	Data as is (good)	Generate 8-bit ECC
		Correctable	Corrected data	
		OK or Correctable AD[2]=0 Uncorrectable AD[2]=1	Good/Corrected Data as is (bad)	Generate 8-bit ECC with PEH inserted
		Uncorrectable AD[2]=0 OK or Correctable AD[2]=1	Data as is (bad) Good/Corrected	Generate 8-bit ECC with PEL inserted
		Uncorrectable both	Data as is (bad)	Generate 8-bit ECC with PED inserted
	ECC protected 32-bit	No Error	Data as is (good)	Generate 7-bit ECC
		Correctable	Corrected data	Generate 7-bit ECC
		Uncorrectable	Data as is (bad)	Generate 7-bit ECC with PEL inserted

Note:
"Other uncorrectable" includes nominally correctable errors that occur while error correction is disabled.

8.7.3.3. Parity to ECC

If one interface of the bridge is protected by parity and the other is protected by ECC and the bridge detects a parity error on data received on the parity protected interface, the bridge does the following:

- Generate the ECC check bits by generating the ECC signature(s) for the data received (including the error).
- Modify ("poison") the signature(s) to indicate that there is a forwarded uncorrectable error by XORing in one of the error signatures from Table 5-4, selected as follows:
  - For 32-bit data (on the destination bus) and:
    - 32-bit data received with incorrect PAR, use the PEL signature.

- 64-bit data received with incorrect PAR and correct PAR64, use the PEL signature when AD[2]=0 and do not poison the ECC signature for AD[2]=1.
- 64-bit data received with correct PAR and incorrect PAR64, do not poison the ECC signature when AD[2]=0 and use the PEL signature when AD[2]=1.
- 64-bit data received with both PAR and PAR64 incorrect, use the PEL signature when AD[2]=0 and when AD[2]=1.

- For 64-bit data (on the destination bus) and:
  - 32-bit data received with incorrect PAR when AD[2]=0 and correct PAR when AD[2]=1, use the PEL signature.
  - 32-bit data received with correct PAR when AD[2]=0 and incorrect PAR when AD[2]=1, use the PEH signature.
  - 32-bit data received with incorrect PAR both when AD[2]=0 and when AD[2]=1, use the PED signature.
  - 64-bit data received with incorrect PAR and correct PAR64, use PEL.
  - 64-bit data received with correct PAR and incorrect PAR64, use PEH.
  - 64-bit data received with both PAR and PAR64 incorrect, use PED.

- Drive AD and C/BE# on the destination bus exactly as observed on the originating bus.
- Drive the newly generated (poisoned) ECC signature(s) on ECC.

The resulting combination of AD, C/BE#, and ECC is still protected against single bit failures (if one occurs, the codeword appears to have a double bit error).

Note that the use of differentiated signatures for 64-bit data forwards the specific parity-error indication and indicates the partial correctness of the data. The received codeword is used in any case where 64-bit parity-protected data, with only one of PAR or PAR64 incorrect, is used.

The inclusion (using XOR) of one of the PEL, PEH, or PED signatures in the ECC signature forces even parity in the forwarded codeword. (An error indication codeword always has even parity.)

This action is summarized in summarized in Table 8-11.

Table 8-11: Bridge Data-Phase Forwarding Actions, Parity to ECC

Orig I/F	Dest I/F	Error Case	Forwarding Action	
			Data	ECC
Parity Protected 64-bit *possible errors:* *PAR bad* *PAR64 bad* *Both bad*	ECC protected 64-bit	No Error	Data as is (good)	Generate 8-bit ECC
		Upper Parity Error (PAR64)	Data as is (bad)	Generate 8-bit ECC with PEH inserted
		Lower Parity Error (PAR)		Generate 8-bit ECC with PEL inserted
		Both Parity Error		Generate 8-bit ECC with PED inserted
	ECC protected 32-bit	No Error	Data as is (good)	Generate 7-bit ECC
		Upper Parity Error (PAR64)	Data as is (bad)	Generate 7-bit ECC with PEL inserted for the DWORD with the error
		Lower Parity Error (PAR)		
		Both Parity Error		Generate 7-bit ECC with PEL inserted for both DWORDs
Parity protected 32-bit *possible errors:* *PAR bad*	ECC protected 64-bit	No Error	Data as is (good)	Generate 8-bit ECC
		Parity Error AD2=1 (PAR)	Data as is (bad)	Generate 8-bit ECC with PEH inserted
		Parity Error AD2=0 (PAR)		Generate 8-bit ECC with PEL inserted
		Parity Error both (PAR)		Generate 8-bit ECC with PED inserted
	ECC protected 32-bit	No Error	Data as is (good)	Generate 7-bit ECC
		PAR Parity Error	Data as is (bad)	Generate 7-bit ECC with PEL inserted

8.7.3.4. ECC to Parity

If one interface of the bridge is protected by parity and the other is protected by ECC, and correction is enabled (on the ECC-protected interface), and the bridge detects a correctable ECC error on data received at one interface, the bridge corrects the error and continues as if the originally received codeword had no error.

If one interface of the bridge is protected by parity and the other is protected by ECC and for data received on the ECC-protected interface the bridge detects an uncorrectable ECC error or the bridge detects a correctable ECC error when correction is disabled, the bridge drives AD, C/BE#, on the destination bus exactly as observed on the originating bus, and drives incorrect parity as follows:

- If the uncorrectable ECC syndrome is PEL, drive PAR and (if the transfer width is 64 bits on the destination bus) PAR64 for each data phase or subphase that had an error, as follows:
  - For 32-bit data (on the destination bus) and:
    - 32-bit data received with PEL, drive incorrect parity on PAR.
    - 64-bit data received with PEL, drive incorrect parity on PAR when AD[2]=0 and correct parity on PAR when AD[2]=1.
  - For 64-bit data (on the destination bus) and:

- 32-bit data received with PEL when AD[2]=0 and with correct ECC when AD[2]=1, drive incorrect parity on PAR and drive correct parity on PAR64.
- 32-bit data received with correct ECC when AD[2]=0 and with PEL when AD[2]=1, drive correct parity on PAR and drive incorrect parity on PAR64.
- 32-bit data received with PEL both when AD[2]=0 and when AD[2]=1, drive incorrect parity on both PAR and PAR64.
- 64-bit data received with PEL, drive incorrect parity on PAR and drive correct parity on PAR64.

- If the uncorrectable ECC syndrome is PEH (the transfer width must be 64 bits on the originating bus), drive PAR and (if the transfer width is 64 bits on the destination bus) PAR64 for each data phase or subphase that had an error, as follows:
  - For 32-bit data (on the destination bus) drive correct parity on PAR when AD[2]=0 and incorrect parity on PAR when AD[2]=1.
  - For 64-bit data (on the destination bus), drive correct parity on PAR and drive incorrect parity on PAR64.
- For other uncorrectable ECC syndromes, and for a correctable ECC syndrome when correction is disabled, drive PAR and (if the transfer width is 64 bits on the destination bus) PAR64 for each data phase or subphase that had an error, as follows:
  - For 32-bit data (on the destination bus), drive incorrect parity on PAR. If 64-bit data is received with an ECC error, both 32-bit phases are driven with incorrect parity on PAR.
  - For 64-bit data (on the destination bus) and:
    - 32-bit data received with an ECC error when AD[2]=0 and with correct ECC when AD[2]=1, drive incorrect parity on PAR and correct parity on PAR64.
    - 32-bit data received with correct ECC when AD[2]=0 and with an ECC error when AD[2]=1, drive correct parity on PAR and incorrect parity on PAR64.
    - 32-bit data received with an ECC error both when AD[2]=0 and when AD[2]=1, drive incorrect parity on both PAR and PAR64.
    - 64-bit data received with an uncorrectable ECC error, drive incorrect parity on both PAR and PAR64.

This action is summarized in summarized in Table 8-12.

Table 8-12: Bridge Data-Phase Forwarding Actions, ECC to Parity

Orig I/F	Dest I/F	Error Case	Forwarding Action	
			Data	Parity
ECC protected 64-bit *possible errors: Correct-able PEL PEH PED and other uncor-rectable* (Note)	Parity protected 64-bit	No Error	Data as is (good)	Generate good parity (PAR and PAR64)
		Correctable	Corrected data	
		PEH	Data as is (bad)	Generate parity, bad PAR64, good PAR
		PEL		Generate parity, bad PAR, good PAR64
		Other uncorrectable		Generate parity, bad PAR64, bad PAR
	Parity protected 32-bit	No Error	Data as is (good)	Generate good parity (PAR)
		Correctable	Corrected data	
		PEH	Data as is (bad)	Generate bad parity (PAR) for the DWORD with the error
		PEL		
		Other uncorrectable		Generate bad parity for both DWORDs
ECC protected 32-bit *possible errors Correct-able PEL and other uncor-rectable* (Note)	Parity protected 64-bit	No Error	Data as is (good)	Generate good parity (PAR and PAR64)
		Correctable	Corrected data	
		OK or Correctable AD[2]=0 Uncorrectable AD[2]=1	Good/Corrected Data as is (bad)	Generate parity, bad PAR64, good PAR
		Uncorrectable AD[2]=0 OK or Correctable AD[2]=1	Data as is (bad) Good/Corrected	Generate parity, bad PAR, good PAR64
		Uncorrectable both	Data as is (bad)	Generate parity, bad PAR64, bad PAR
	Parity protected 32-bit	No Error	Data as is (good)	Generate good parity (PAR)
		Correctable	Corrected data	
		Uncorrectable	Data as is (bad)	Generate bad parity (PAR)

Note:
"Other uncorrectable" includes nominally correctable errors that occur while error correction is disabled.

8.7.4. Bridge Internal Error Protection

Bridges with ECC support are required to provide protection against single bit errors in all phases of transactions they forward from one bus operating in ECC mode to another bus operating in ECC mode. That is, if both interfaces of the bridge are operating in ECC mode, the bridge must do the following:

- Check ECC on the originating interface.
- In all phases of the transaction, protect against data corruption that occurs between the originating interface and the destination interface. The protection mechanism must be commensurate with the level of protection provided by the ECC on the bus (i.e., the probability of an uncorrectable error and an undetected error must be no worse than it is on the bus). The means by which the bridge does this is not specified.

- Drive the appropriate ECC for the width of the transaction on the destination bus.

IMPLEMENTATION NOTE

Error Protection for Transactions that Cross a Bridge

A PCI-X Mode 2 bridge with both buses operating in ECC mode must protect against data corruption in all phases of a transaction forwarded from the originating interface to the destination interface. The protection mechanism must be commensurate with the level of protection provided by the ECC on the bus (i.e., the probability of an uncorrectable error and an undetected error must be no worse than it is on the bus). This provides end-to-end data protection for systems that have PCI buses operating with ECC protection between the requester and completer. If a bridge were to violate this requirement, data could be silently corrupted.

One implementation of this requirement is to store the check bits with each phase of the transaction on the originating bus. If the transaction on the destination bus is the same width as it was on the originating bus, the same check bits can be driven on the destination bus. However, if the transaction width is different on the destination bus, the bridge must check that no error have occurred inside the bridge, and then generate new check bits for the width of the transaction on the destination bus.

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 8-13 shows the index values defined for this message class. All other indexes are reserved.

Table 8-13: 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 8.7.1.2.)
01h	Target-Abort. The PCI-X bridge encountered a Target-Abort on the destination bus. (See Section 8.7.1.3.)
02h	Uncorrectable Write Data Error. The PCI-X bridge encountered an uncorrectable data error on a non-posted write transaction on the destination bus. (See Section 8.7.1.1.2.)

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 secondary bus and 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. This section also describes Mode 2 bridges as if they directly control $V_{I/O}$ for their secondary interface. Such discussion is not intended to preclude other alternatives such as selecting between Mode 1 and Mode 2 and setting $V_{I/O}$ external to 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.

A PCI-X Mode 2 bridge is permitted not to clear the state of the logic controlling $V_{I/O}$ for its secondary interface and to wait until the rising edge of primary RST# to set it to the appropriate value. Mode 2 bridges are also permitted to sense the state of PCIXCAP and to select the appropriate value for $V_{I/O}$ while RST# is asserted. Setting $V_{I/O}$ prior to the rising edge of primary RST# generally reduces the delay between the time primary RST# deasserts and secondary RST# deasserts.

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 Appendix A, "Detection of PCI-X Add-in Card Capability," in PCI-X EM 2.0 for examples of methods for detecting the state of PCIXCAP.
2. Select the appropriate mode and clock frequency for the add-in 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. Apply the appropriate voltage to $V_{I/O}$ (if not already done prior to the rising edge of primary RST#).
5. 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 specified in Section 2.3.2, "Reset," in PCI-X EM 2.0.
6. 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 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 2-7, "3.3V General Timing Parameters," in PCI-X EM 2.0. A PCI-X Mode 2 bridge must also guarantee that $V_{I/O}$ is stable at the appropriate voltage for the length of time specified in Table 2-7, "3.3V General Timing Parameters," in PCI-X EM 2.0. In some instances, these requirements significantly increase the delay between the time primary RST# deasserts and the time the bridge deasserts its secondary RST#.

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 PCI Bridge 1.1.

A. Appendix—Conventional PCI vs. PCI-X Protocol Rule Comparison

Table A-1: Conventional PCI vs. PCI-X Protocol Comparison

Arbitration	**Conventional PCI**	**PCI-X Mode 1**	**PCI-X Mode 2**
Arbiter Monitoring Bus	No	YES	YES
Transfer Rates	**Conventional PCI** **32-bit bus, 64-bit bus**	**PCI-X Mode 1** **32-bit bus, 64-bit bus**	**PCI-X Mode 2** **16-bit bus, 32-bit bus, 64-bit bus**
33 MHz	133, 266 MB/sec	Not Supported	Not Supported
66 MHz	266, 533 MB/sec	266, 533 MB/sec	133, 266, 533 MB/sec
100 MHz	Not Supported	400, 800 MB/sec	200, 400, 800 MB/sec
133 MHz	Not Supported	533, 1066 MB/sec	266, 533, 1066 MB/sec
266 MHz	Not Supported	Not Supported	533, 1066, 2133 MB/sec
533 MHz	Not Supported	Not Supported	1066, 2133, 4267 MB/sec
Transaction Types	**Conventional PCI**	**PCI-X Mode 1**	**PCI-X Mode 2**
Memory	Supported	Supported	Supported
I/O	Supported	Supported	Supported

Config	Supported	Supported	Supported
Interrupt Acknowledge	Supported	Supported	Supported
Special Cycle	Supported	Supported	Supported
Dual Address Cycle	Supported	Supported	Supported
Split Transactions	Not supported	Supported	Supported
Priority Transactions	Not supported	Not supported	Not supported
Non-Cache-Coherent Transactions	Not supported	Supported	Supported
No/Relax Ordering Rules	Not supported	Supported	Supported
Address Re-mapping	Not supported	Not supported	Not supported
Address and Data Bus	Multiplexed	Multiplexed	Multiplexed
# of New Pins	N/A	1 new pin	5 new pins
Isochronous Transactions	Not supported	Not supported	Not supported
Target Write Buffer Flush	No, supported using a standard read command	No, supported using a standard read command	No, supported using a standard read command
Transaction Ordering	No (Bus Ordering only)	Yes, device controls transaction ordering (Relaxed Ordering)	Yes, device controls transaction ordering (Relaxed Ordering)
Orthogonal Protocol Support	Yes	Yes	Yes
Power Management Support (PME)	Optional	Yes	Yes
Non-Snooped Accesses to Host Memory Allowed While Host Memory Locked	Typically not supported	Yes	Yes
Transaction Termination	**Conventional PCI**	**PCI-X Mode 1**	**PCI-X Mode 2**
Initiator Termination	Supported	Supported	Supported
Master-Abort	Supported	Supported	Supported
Target Disconnect with data	Supported	Supported	Supported

Target Disconnect without data	Supported	Not supported	Not supported
Target Retry	Supported	Supported	Supported
Target-Abort	Supported	Supported	Supported
Burst Transaction	**Conventional PCI**	**PCI-X Mode 1**	**PCI-X Mode 2**
# burst data clocks	Two or more data clocks	One or more data clocks	One or more data clocks
Wait states	Target and/or Initiator can inject wait states	Initiator cannot inject wait states. Target can only inject initial wait states before data transfer starts	Initiator cannot inject wait states. Target can only inject initial wait states before data transfer starts
Cacheline Size	Programmable	Not used (replaced with ADB)	Not used (replaced with ADB)
Latency Timer	Programmable	Programmable	Programmable
Memory Read	BE are valid	BE only for DWORD transactions	BE only for DWORD transactions
Memory Read Line	BE are valid (ignored by Target)	Replaced with burst transactions	Replaced with burst transactions
Memory Read Multiple	Yes	Not supported	Not supported
Memory Write	BE are valid	BE are valid	BE are valid
Memory Write and Invalidate	BE are valid (ignored by Target)	Replaced with Memory Write Block transaction	Replaced with Memory Write Block transaction
# data transfers per clock	1	1	1, 2, or 4
Burst Length	**Conventional PCI**	**PCI-X Mode 1**	**PCI-X Mode 2**
Minimum data clocks	2 Data Clocks	Byte count or 32 clocks for 32 bit bus Byte count or 16 clocks for 64 bit bus	Byte count or 64, 32, or 16 clocks (common-clock, 266, and 533, respectively) for 16 bit bus Byte count or 32, 16, or 8 clocks (common-clock, 266, and 533, respectively) for 32 bit bus Byte count or 16, 8, or 4 clocks (common-clock, 266, and 533, respectively) for 64 bit bus
32 bit Bus	8 bytes	8 bytes	8 bytes

32 bit Minimum Burst Length	8 bytes	Byte count or 128 bytes	Byte count or 128 bytes
64 bit Bus	16 bytes	16 bytes	16 bytes
64 bit Minimum Burst Length	16 bytes	Byte count or 128 bytes	Byte count or 128 bytes
Maximum block size	Open (Unlimited)	4096 bytes	4096 bytes
Standard block size	Cache Line Size	Fixed 128 bytes	Fixed 128 bytes
Topology	**Conventional PCI**	**PCI-X Mode 1**	**PCI-X Mode 2**
Port/Bus/Hierarchical Bus	No/Yes/Yes	Yes/Yes/Yes	Yes/Yes/Yes
PCI Slot Compatibility	N/A	Yes	Yes
Decode Speeds	**Conventional PCI**	**PCI-X Mode 1**	**PCI-X Mode 2**
1 clock after address phase(s)	Decode Speed FAST	Not supported	Not supported
2 clock after address phase(s)	Decode Speed MED	Decode Speed A (parity mode only)	Not supported
3 clock after address phase(s)	Decode Speed SLOW	Decode Speed B	Decode Speed B
4 clock after address phase(s)	Decode Speed Subtractive	Decode Speed C	Decode Speed C
6 clock after address phase(s)	Not supported	Decode Speed Subtractive	Decode Speed Subtractive
Config Access	**Conventional PCI**	**PCI-X Mode 1**	**PCI-X Mode 2**
Address Predrive	Optional, system dependent	Required	Required (64- and 32-bit buses only)
Configuration Space Size	256 bytes	256 bytes	4096 bytes
Transaction Phases	**Conventional PCI**	**PCI-X Mode 1**	**PCI-X Mode 2**
Address and Command Phase	Supported	Supported	Supported
Response Phase	Decode Speed	Decode Speed	Decode Speed
Attribute Phase	Not Available	Supported	Supported

Data Phase	Supported	Supported	Supported
Termination Phase	Initiator and Target	Initiator and Target	Initiator and Target
Turn-around	Required	Required	Required
Attribute Field	**Conventional PCI**	**PCI-X Mode 1**	**PCI-X Mode 2**
Byte Count	Not supported	Supported	Supported
Don't Snoop	Not supported	Supported	Supported
Relaxed Ordering	Not supported	Supported	Supported
Function #	Not supported	Supported	Supported
Device #	Not supported	Supported	Supported
Bus #	Not supported	Supported	Supported
Tag	Not supported	Supported	Supported
Bus Width	**Conventional PCI**	**PCI-X Mode 1**	**PCI-X Mode 2**
Memory Address	32 or 64 bits	64 bits	64 bits
I/O Address	32 bits	32 bits	32 bits
Data	32 or 64 bits	32 or 64 bits	16, 32, or 64 bits

B

B. 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 2.3 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.

B.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.

B.2. Relaxed Read Ordering

PCI 2.3 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 2.3. 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.

B.3. Co-location of Payload and Control

Programming models that require the payload 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 writes to the control space, the requester must not set the Relaxed Ordering attribute on writes to the control space.

B.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.

B.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.

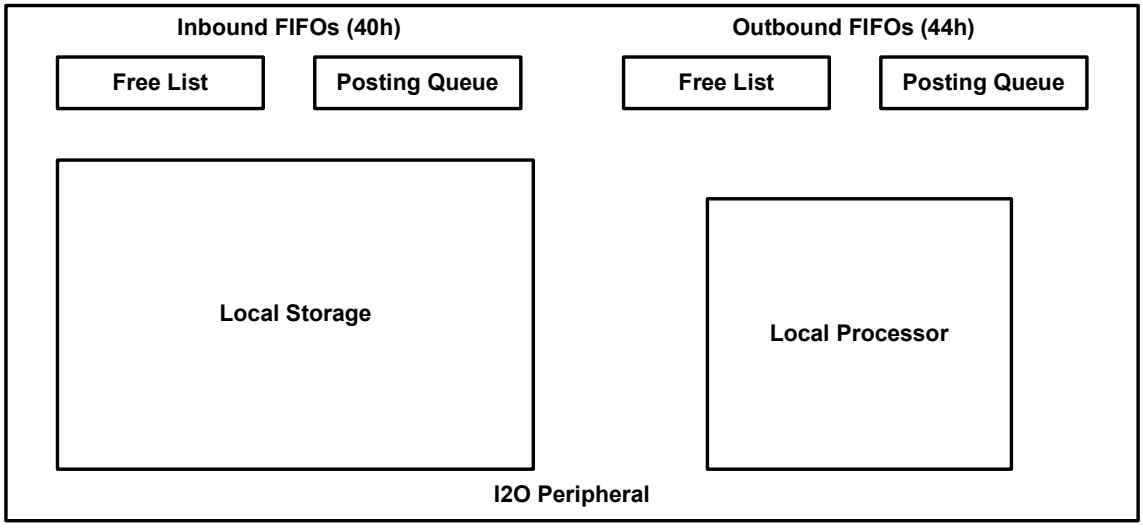

Figure B-1: I_2O Standard Components

Every I_2O device implements a standard programming interface shown in Figure B-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.)

B.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.

B.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 B-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.

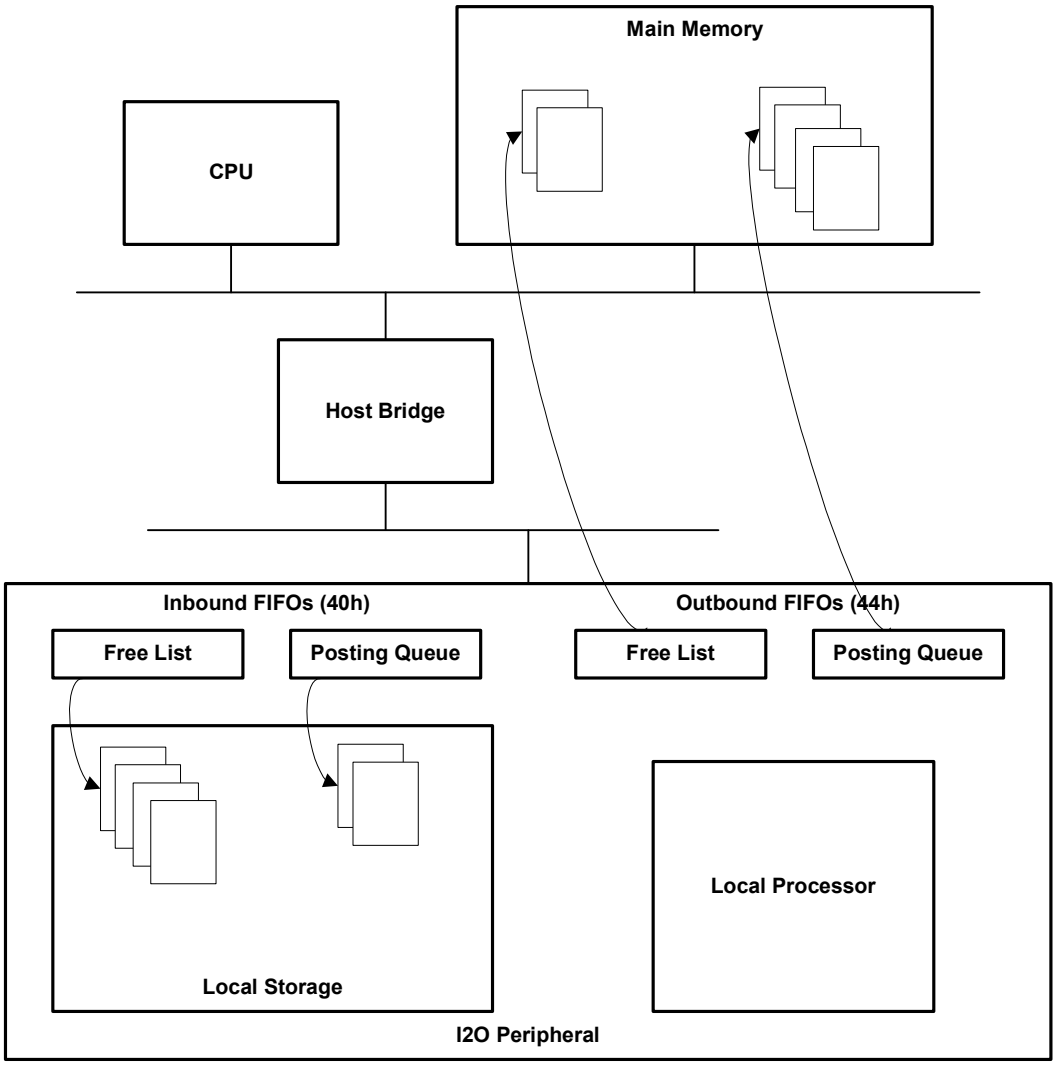

Figure B-2: I_2O Push Model

B.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 B-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.

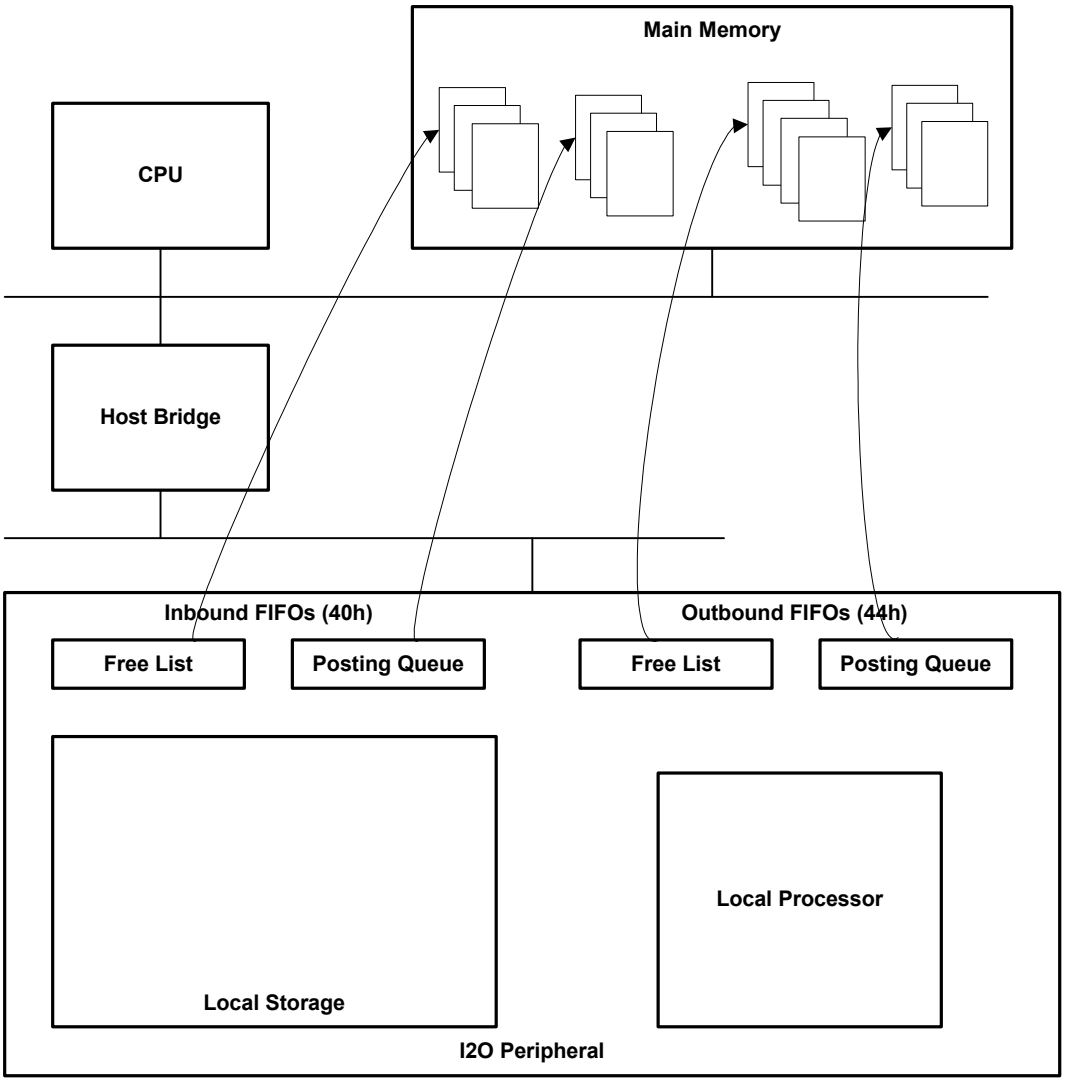

Figure B-3: I_2O Pull Model

B.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.

B.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.

C

C. 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 Capabilities List
   - ***Cap_Ptr*** register (PCI Configuration 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 (such as a local-area-network 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.

D

D. Appendix—Setting Performance Registers

D.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.

D.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.

E. Appendix—ECC Applications

E.1. ECC in Mode 1

E.1.1. Mode 1 ECC in Embedded Systems

In an embedded application, such as one in which all devices on a bus segment are soldered onto a board, it is desirable to be able to default that bus segment to ECC rather than parity protection. The method by which the central resource determines that it is in an embedded application that defaults to ECC protection in Mode 1 is not specified, however, pin strapping on the host bridge would be one alternative. The designer of the embedded application may thus choose to operate all devices on a bus segment in ECC mode and gain the advantages of ECC protection without requiring any additional software to switch from parity to ECC protection.

E.1.2. Mode 1 ECC in Slot-Based Systems

In slot-based systems operating in Mode 1, system configuration software must determine whether all devices on a bus segment support ECC before enabling ECC on that segment. Enabling ECC is complicated by the fact that the devices switch from parity protection to ECC protection one device at a time, and, therefore, some devices are in ECC mode and some are in parity mode while the enabling process is taking place.

An example method for enabling ECC support on a bus operating in Mode 1 follows:

1. Read the PCI-X Capabilities List Item Version field of each function of each device on the bus and verify that all functions of all devices support ECC in Mode 1.
2. For each function of each device, do the following:
   a. Write a 0 to the Parity Error Response bit (bit 6) in the Command register (04h in the Configuration Space header).
   b. Write a 1 to the ECC Mode bit in the ECC Control and Status register.

   As the Parity Error Response bit is cleared in each function, that function ceases to provide any data protection but is prevented from falsely detecting "ECC errors" on the parity-protected transactions still being addressed to the other functions on the bus segment. Note that the Parity Error Response bit is cleared by default after RST# is asserted, so only the ECC Mode bit must be set enabling ECC immediately following the assertion of RST#.

3. After Step 2 is complete for all functions of all devices, set the source bridge's ECC Mode bit to enable ECC generation, checking, and correction. At this point all devices generate ECC, so no false errors would be detected due to parity-protected transactions on the bus segment.
4. Clear the ECC error logging register of any false errors that occurred during the transition from parity mode to ECC mode.
5. Set the Parity Error Response bit for all functions of all devices to enable ECC error reporting. As this bit is set in each function, it begins providing full ECC protection.

Note that beginning with Step 2 above, the system operates without full error protection until Step 5 is complete.

Hot-plug systems add the further complication of needing to determine the ECC capability of a newly inserted card (which defaults to parity protection in Mode 1). One way to accomplish this is by disabling ECC error reporting on all functions of all other devices on the same bus segment by clearing their Parity Error Response bits, placing the source bridge back into parity mode, and then reading the PCI-X Capabilities List Item Version field of the newly inserted card. If the new card supports ECC, its ECC Mode bit is written to enable ECC generation but its Parity Enable Response bit is left cleared, and the enabling process above continues from Step 3. (Note that the SERR# Enable bit must remain cleared to prevent the device from misinterpreting ECC-protected address phases for transactions between other devices.) If the new card does not support ECC, the system must choose either to change the bus to parity protection, or to disable the slot with the new card and resume the enabling process above at Step 3. Note that the system operates without full data protection until Step 5 of the enabling process is again completed. Hot-plug systems that cannot tolerate operating without error protection for this length of time would quiesce all devices on the bus before disabling ECC checking, as they would if the bus were changing clock frequencies rather than error protection modes.

E.2. Use of PCI-X ECC in Memory Applications

E.2.1. Features for Memory ECC Applications

The ECC is also designed to make it suitable as a memory ECC if data is stored from AD[00] to AD[31] or AD[63] followed by ECC[0] through ECC[6] or ECC[7], broken into nibbles or bytes in the natural order. In some cases, the ECC signature used on the bus requires adjustments before being used as a memory ECC signature (see Section E.2.2). When used as a memory ECC, the ECC has the following additional characteristics:

- The "nibble protect" feature guards against failures of 4-bit wide memory chips.
- The ECC provides significant coverage against multiple-bit errors within a single data byte. Although it is not possible to protect against all of the mis-correct or mis-detect cases over a full byte, the ECC detects almost 78% of all such errors as uncorrectable.

- The address protection provided in the basic sub-ECC, as part of the phase protection for data phases, is easily extended to cover all address lines. The phase protection mechanism is described in Section 5.1.2.2.3.

E.2.2. Adjustments for Memory ECC Applications

The ECC protection method defined here is designed to provide a sound base for a memory ECC and to make conversions between the local bus ECC and the memory ECC fast and efficient. The nibble protection and the high byte-error detection rate of the ECC were specifically selected to make the ECC usable as a robust memory ECC. However, any ECC signature on the PCI bus may require adjustments before being used (directly) as a memory ECC signature, depending on the specific transaction, as follows:

- The bit-signatures used against data bits are the bit-signatures given in Table 5-1 and Table 5-2. The bus ECC signature can thus be used as a basis for creating the memory ECC signature, with a few adjustments.
- If any C/BE# signals have been included in the bus ECC signature, they must be removed before it is used as a memory ECC signature. For full width data phases, the C/BE# signals are either not included in the signature or are all logic 0 signals, which effectively means that the corresponding signatures were not included in the bus ECC signature. As a result, there is generally no adjustment to be made for full width data. For data that is not full width, a memory read-modify-write cycle with regeneration of the ECC signature is generally required.
- The data phase signature (as given in Table 5-4) used in the bus ECC signature may not be appropriate in the memory ECC signature. Even if, as suggested below, the memory ECC uses phase signatures, the lower address bits of the memory address may differ from the "phase address" used to generate the bus ECC signature. In particular, Split Completions use the address bits provided in the Split Completion address as the starting point for the "phase address" and this may not match the actual memory address used to store the data (e.g., for DWORD transactions these bits are always 0). The data phase signature (of the bus ECC signature) may be directly adjusted or it may be removed and (if the memory ECC uses phase signatures) the appropriate phase signature (for the memory address) inserted (via XOR) into the signature.
- It is desirable for a memory ECC to provide protection against addressing errors. The data phase signature used for the bus ECC only provides protection for address lines [4:3] and for phasing errors in burst memory transfers. It is suggested that a memory ECC based on the ECC defined in this section should include the data phase signature described in Table 5-4 to protect against such errors, and should, in addition, provide protection against errors in other address lines as described below.

The adjustments described above can either be applied as modifications (deltas) to the ECC signature received from the PCI bus (a C/BE# or inappropriate phase signature can be "removed" simply by re-including it since XOR is a self-canceling operation) or the memory ECC signature can be regenerated directly. For the latter method, it should be possible to obtain the signature for the data bits from the XOR trees used to check the bus ECC and a separate set of XOR trees should not be required.

The extension described here suggests a method for extending the phase protection coverage to provide protection for (all) address lines.

For each address line to be protected, select a unique signature with even parity (these signatures should not be AL or AH if the memory is 32 bits wide). The data phase protection signature completely protects address lines [4::3] and provides some protection for address line [5], and, for 32-bit memories, address line [2] is protected using AL and AH, so the selection of signatures should start with address line [5]. If these signatures are appropriately selected, the mechanism can protect against all errors in one, two, or three address lines, and the normal data phase protection signature also provides protection against cycle errors in burst memory transfers.

One method of generating such signatures is to use the RSV, RS2, RS3, and RS4 signatures (which are not used in communications between two PCI-X devices) and the AX and AY signatures (not used for data phases) in combination with the signatures for C/BE[3::0]# which would not be used in a memory, as follows:

- One of the AX, AY, RSV, or RS2 (or, if needed, RS3 or RS4) signatures.
- Either one or three of the C/BE[3::0]# signatures (see Table 5-1).

This allows up to 40 address bits to be covered with unique signatures. Note that a few of these combinations yield duplicated values and the duplications must be avoided.

The address protection mechanism is provided by including, in the ECC signature for the codeword, the data phase protection signature for address bits [6::3] combined with the assigned signature for each active (logic 1) bit in the address associated with the codeword. All of the extended signatures described above have even parity (like the AL and AH signatures) and do not alter the parity of the codeword. Because of the nibble protect characteristics of the C/BE[3::0]# signatures, this extension protects the codeword against all errors in one, two, or three of the included address lines and has a high probability of detecting most other address line errors. This approach can be used to protect up to over 40 address lines.

E.3. Converting Between ECC Widths

The following sections apply to cases in which ECC widths are being changed, such as PCI-X bridges, memory controllers, etc.

E.3.1. Combining ECC Signatures that Include Phase/Error Signatures

In some cases it may be desirable to construct an eight-bit ECC signature against a 64-bit data item from two seven-bit ECC signatures associated with the two halves of the data, for example when a bridge receives 32-bit data but forwards 64-bit data or when a memory controller receives 32-bit data but stores data in 64-bit words. In such cases the eight-bit ECC could be re-calculated directly from the data, but it is generally faster to combine the seven-bit ECC signatures.

When combining two seven-bit ECC signatures containing phase signatures (see Section 5.1.2.2.3) and possibly error signatures (see Section 8.7.1.1), the simple method described in Step d in Section 5.1.2.2.2 does not produce the correct result without some adjustments.

Seven-bit ECC signatures are only combined for data phases (address and attribute phases are never combined). Each of the seven-bit ECC signatures is expected to include a data phase protection signature as described in Section 5.1.2.2.3. When two 32-bit (or 36-bit) DWORD items are to be combined, they are the two halves of a naturally aligned 64-bit QWORD, and bits [6::3] of the phase address is the same for both items. The data phase protection signatures are thus the following, where "Aq" is the A5 through A0 signature associated with the [6::3] bits of the phase address:

- Aq ⊕ AL (the low 32-bit "width marker"—see Table 5-4) for the lower DWORD (which is [31::00] of the QWORD).
- Aq ⊕ AH (the high 32-bit "width marker"—see Table 5-4) for the upper DWORD (which is [63::32] of the QWORD).

If both seven-bit ECC signatures are valid (i.e., both form valid codewords when combined with the associated DWORDs), the signatures are (correctly) combined as follows:

- Combine the signatures using the method described in Step d in Section 5.1.2.2.2.
- Adjust the result by XORing in the ADJ adjustment in Table 5-4. The result is the basic eight-bit ECC signature without any data phase protection signature. The two Aq signatures combine to FFh because the phase signature for the upper DWORD has odd parity and is therefore effectively inverted in the result (just as the bit signatures are inverted in Table 5-2) and XORing a term with its inversion results in an FFh value, and ADJ is thus simply FFh ⊕ AL ⊕ AH.
- Modify the combined signature by including (using XOR) the data phase protection signature, which, in this case, is simply Aq.

For example, if the 64-bit data item is 0100 0000 0001 0000h and has a phase address of 28h (this covers only address bits [6::0]), the eight-bit signature would be C4h ⊕ 6Eh ⊕ 7Ch (AD[56], AD[16], and phase signature A5, respectively), which equals D6h. The seven-bit ECC signature for the upper part of the data (01000000h) would be 3Bh ⊕ 7Ch ⊕ 18h (AD[24], A5, and AH), which equals 5Fh. This value has even parity, so the nominal value for E7 when combining is 0 (the ADJ value inverts this). The seven-bit ECC signature for the lower half of the data is 6Eh ⊕ 7Ch ⊕ 24h (AD[16], A5, and AL), which equals 36h. Since the nominal E7 value is zero, the combination method results in a value of 36h ⊕ 5Fh ⊕ C3h ⊕ 7Ch (low, high, ADJ, and A5), which equals D6h again.

If one of the seven-bit ECC signatures indicates a single bit (correctable) error and the other is valid, the algorithm above results in an eight-bit ECC signature that correctly indicates the bit in error, with one exception: If the ECC signature associated with AD[2]=1 (the "upper" data) indicates an error in one of the ECC check bits (i.e., ECC[6::0]), the error signature (syndrome) will be inverted in the eight-bit ECC signature as described in the implementation note in Section 5.1.2.3 and in Table E-1.

If both seven-bit ECC signatures indicate a single bit error (or one indicates a single bit error and the other indicates PEL), the combination algorithm above results in an uncorrectable error with the correct syndrome for the bits in error.

If one of the seven-bit ECC signatures contains the PEL error signature and the other is valid, the combination algorithm above produces the correct result, inverting PEL into PEH if it is the ECC signature associated with AD[2]=1 that includes PEL.

If both seven-bit ECC signatures include the PEL signature, the combination algorithm above must be adjusted to use the PADJ adjustment (in Table 5-4) in place of ADJ. With this modification, the resulting eight-bit ECC signature correctly includes PED. (The two PEL signatures combine to FFh, and PADJ is simply ADJ ⊕ FFh ⊕ PED).

If one seven-bit ECC signature indicates an uncorrectable error and the other is valid, the combination algorithm preserves the error. If both seven-bit ECC signatures indicate errors (other than the cases described above), the eight-bit ECC signature reflects the combined error signature and may result in a mis-correction or a mis-detection.

The combination algorithm can be reversed to split an eight-bit ECC signature into two seven-bit ECC signatures. The seven-bit basic ECC signature for one part of the data (normally the "upper" part) must be calculated and used to "split" the eight-bit ECC signature. If the eight-bit ECC signature includes errors, the error must be "assigned" to one or both of the seven-bit ECC signatures. With the exception of PEL or PEH (which must be split into the appropriate PEL signature), uncorrectable errors should result in the inclusion of PEL in both seven-bit ECC signatures. Correctable errors (and PEL or PEH) can be assigned by reference to the E7 check bit, steering errors with E7 to the "upper" section and errors without E7 to the "lower" section.

E.3.2. Handling Combined Eight-Bit ECCs in Recovery Software

If seven-bit ECC signatures are combined to form an eight-bit ECC signature using the method described in Section 5.1.2.2.2 without (or before) correction, any error indicated in the seven-bit ECC signatures is carried forward into the eight-bit ECC signature.

In most cases, the resulting eight-bit syndrome (from a combination of two seven-bit ECC signatures where one or both indicate an error) correctly indicates the original error. However, there are a few exceptions and special considerations that require special attention in recovery software.

In simple cases involving just a single bit (nominally correctable) error in just one of the seven-bit ECC signatures (with the other seven-bit ECC signature correct), the eight-bit ECC signature/syndrome indicates the correct error (including conversion of PEL to PEH) with one exception: If the seven-bit ECC signature/syndrome for the "upper" codeword (would have) indicated an error in one of the ECC check bits in that "upper" codeword, the resulting eight-bit ECC syndrome is the inverse of the syndrome indicated in Table 5-5, and has the value indicated in Table E-1 below.

Table E-1: ECC Check Bit Syndromes in Combined Signatures

		ECC Check Bits (E7-E0)							
	code	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
inv ECC[7]/E7	7Fh		X	X	X	X	X	X	X
inv ECC[6]/E6	BFh	X		X	X	X	X	X	X
inv ECC[5]/E5	DFh	X	X		X	X	X	X	X
inv ECC[4]/E4	EFh	X	X	X		X	X	X	X
inv ECC[3]/E3	F7h	X	X	X	X		X	X	X
inv ECC[2]/E2	FBh	X	X	X	X	X		X	X
inv ECC[1]/E1	FDh	X	X	X	X	X	X		X
inv ECC[0]/E0	FEh	X	X	X	X	X	X	X	

In cases where both seven-bit ECC signatures indicated a correctable error, the combined eight-bit ECC indicates an uncorrectable error (an even parity syndrome). If one of the seven-bit ECC signatures is correct and the other indicates an uncorrectable error, the combined eight-bit ECC signature correctly reflects the nature of the error.

In cases where one of the seven-bit ECC signatures indicates an uncorrectable error and the other seven-bit ECC signature indicates some (correctable or uncorrectable) error, the syndromes are effectively combined. Depending on the specific errors involved, the resulting syndrome results in either a mis-detection or a mis-correction, or, in most cases, the syndrome still indicates an uncorrectable error.

E.4. Detecting Single-Pin Problems on the 16-Bit Bus

On either a 64- or 32-bit bus, a hardware problem (e.g., an open line or many solder shorts) on a single signal pin generally results in a frequent repetition of a single, correctable (i.e., single-bit error), syndrome. The syndrome indicates the specific signal pin on which the error is present and greatly assists in debugging of the problem, especially during manufacturing.

On the 16-bit bus, a single-pin problem also generally results in frequent errors, but the syndromes are a mix of correctable syndromes (for either of the data bits sent over the same pin) and uncorrectable syndromes. The seven-bit ECC is designed to provide a unique signature for each pair of signals, as described in Table 5-6. (See Table 2-22 for the assignment of signals to pins on the 16-bit bus.) The indicated syndromes can occur as the result of other double and multi-bit errors, but their occurrence is a strong indication of a problem on the associated signal pin.

Table E-2: ECC Syndromes for Single-Pin Double-Bit Errors on the 16-bit Bus

Bus Pin	Combined Signature	Latched Syndrome
AD[16]	27h+Chk	A7h
AD[17]	55h+Chk	D5h
AD[18]	06h+Chk	86h
AD[19]	78h+Chk	F8h
AD[20]	7Dh+Chk	FDh
AD[21]	1Dh+Chk	9Dh
AD[22]	14h+Chk	94h
AD[23]	22h+Chk	A2h
AD[24]	60h+Chk	E0h
AD[25]	39h+Chk	B9h
AD[26]	5Ah+Chk	DAh
AD[27]	7Bh+Chk	FBh
AD[28]	48h+Chk	C8h
AD[29]	59h+Chk	D9h
AD[30]	1Bh+Chk	9Bh
AD[31]	69h+Chk	E9h
ECC[2]	05h	05h
ECC[3]	0Ah	0Ah
ECC[4]	50h	50h
ECC[5]	20h+Chk	A0h
C/BE[2]#	44h+Chk	C4h
C/BE[3]#	65h+Chk	E5h

E.5. ECC Check/Correct Logic Example

Figure E-1 shows an example ECC correction pipeline for 32-bit data. The illustrated pipeline requires two clocks. During the first clock the data arrives and the expected ECC signature is generated. During the second clock, the received ECC signature is XORed with the expected ECC signature to develop the ECC syndrome and the syndrome is decoded to perform the correction, if any. For source-synchronous data, the ECC check bits arrive with the data. An implementation could simply delay the ECC check bits one clock and use the same pipeline or it could adjust the timing of the pipeline appropriately for source-synchronous data.

In practice, the ECC generation logic would also conditionally include the C/BE# signals in the expected signature XOR-trees (for address, attribute, and some data phases). The illustrated logic uses simple XOR-trees (effectively, an 17- or 18-wide XOR gate) to generate the expected ECC signature, with the XOR-trees derived from Table 5-1. Each one of the E0-E6 columns describes the associated XOR-tree. The generation logic is identical to the logic that would be used to generated the ECC signature for data sent to the bus and, in most implementations, this generation logic is shared by both the "send" and "receive" paths. The "Phase Info" is the phase signature described in Section 5.1.2.2.3. This signature

can be developed (pipelined) in advance of the arrival of the data from the PCI bus. An implementation could use any logic generating an equivalent result.

For 64-bit data, additional logic would be needed for E7 and the XOR-trees are wider. An implementation could use simple XOR-trees or it could use the alternate method described in Section 5.1.2.2.2 to generate the expected ECC signature. For the 16-bit bus, an additional XOR-tree is needed to generated the expected E16 value and the decode logic would need to be adjusted to verify E16.

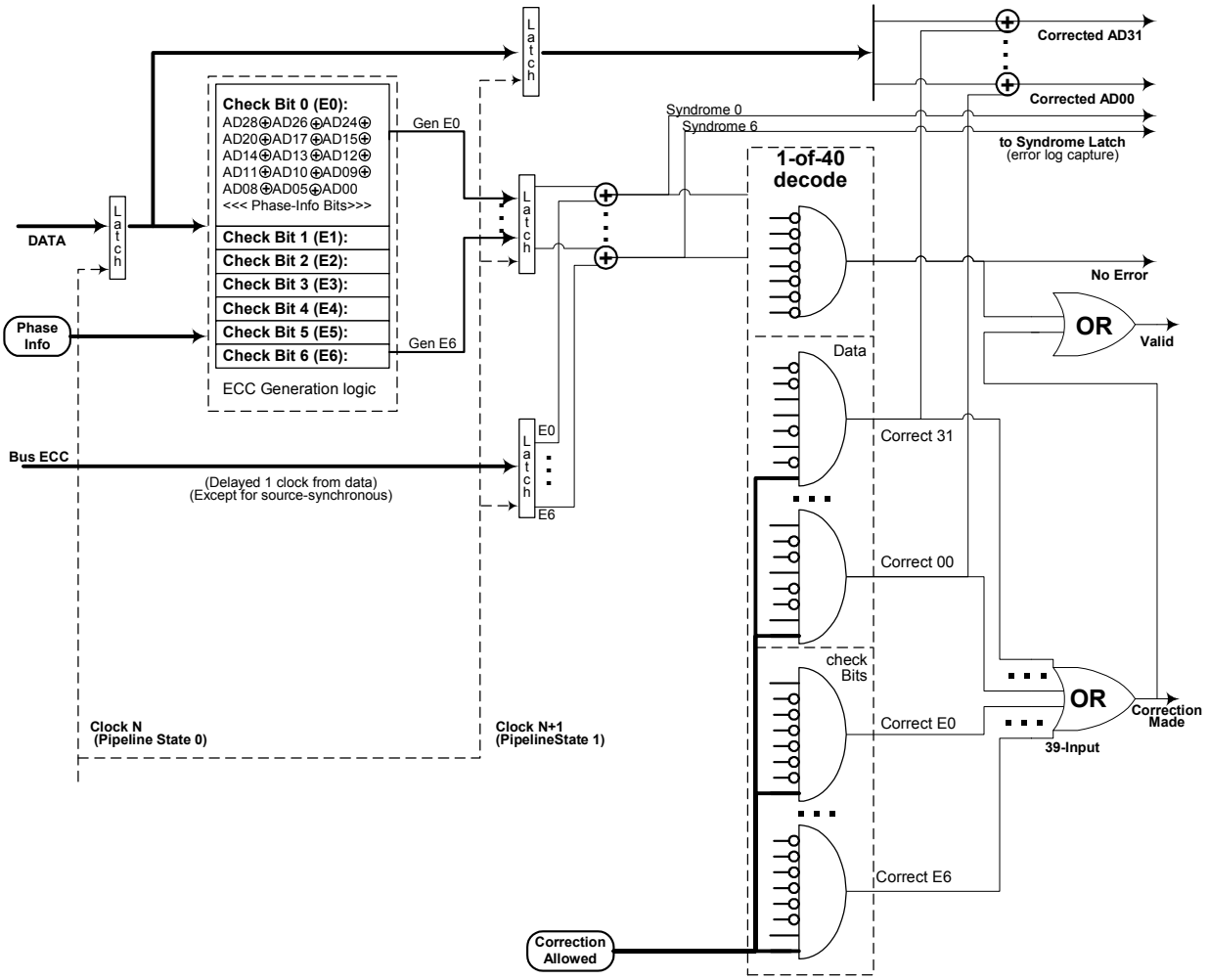

Figure E-1: ECC Check/Correct Logic Example