# PCI IP User's Guide

November 2010
IPUG18_09.2

# Table of Contents

# Introduction

Lattice's Peripheral Component Interconnect (PCI) Intellectual Property (IP) cores provide an ideal solution that meets the needs of today's high performance PCI applications. The PCI IP cores provide a customizable, 32-bit or 64-bit PCI Master and Target or Target only solution that is fully compliant with the *PCI Local Bus Specification, Revision 3.0* for speeds up to 66MHz. The PCI cores bridge the gap between the PCI Bus and specific design applications, providing an integrated PCI solution. These cores allow designers to focus on the application rather than on the PCI specification, resulting in a faster time-to-market.

PCI is a widely accepted bus standard that is used in many applications including telecommunications, embedded systems, high performance peripheral cards, and networking. The family of PCI IP core is one of the many in Lattice's portfolio of IP cores. For more information on these and other products, refer to the Lattice web site at: http://www.latticesemi.com/products/intellectualproperty/.

This document covers Target only, Master and Target, 64-bit, and 32-bit PCI IP cores implemented in a number of devices. Details of Master and 64-bit operation only apply to the appropriate cores. Pin assignments for specific variations of this core are described at the end of this document.

## Quick Facts

Table 1-1 through Table 1-8 give quick facts about the PCI IP core for LatticeEC™, LatticeECP™, LatticeECP2™, LatticeECP2M™, LatticeECP3™, LatticeXP™, LatticeXP2™, LatticeSC™, MachXO™, MachXO2™, and LatticeSCM™ devices.

***Table 1-1. PCI IP Core Quick Facts--PCI master/target 66MHz/64bit***

| | | **PCI IP configuration** | | | | | |
|---|---|---|---|---|---|---|---|
| | | **PCI master/target 66MHz 64bit** | | | | | |
| **Core Requirements** | FPGA Families Supported | LatticeEC LatticeECP | Lattice ECP2 Lattice ECP2M | LatticeXP | LatticeXP2 | LatticeECP3 | LatticeSC LatticeSCM |
| | Minimal Device Needed | LFEC10E-5F484C | LFE2-12E-6F484C | LFXP15C-5F388C | LFXP2-17E-6F484C | LFE3-35EA-7FN484CES | LFSC3GA15E-6F900C |
| **Resource Utilization** | Data Path Width | 64 | | | | | |
| | LUTs | 2500 | | | | | |
| | Registers | 900 | | | | | |
| **Design Tool Support** | Lattice Implementation | Lattice Diamond™ 1.0 or ispLEVER® 8.1 | | | | | |
| | Synthesis | Synopsys® Synplify™ Pro for Lattice D-2009.12L-1 | | | | | |
| | | Mentor Graphics® Precision™ RTL | | | | | |
| | Simulation | Aldec® Active-HDL™ 8.2 Lattice Edition | | | | | |
| | | Mentor Graphics ModelSim™ SE 6.3F | | | | | |

*Table 1-2. PCI IP Core Quick Facts--PCI master/target 66MHz/32bit*

| | | PCI IP configuration | | | | | |
|---|---|---|---|---|---|---|---|
| | | PCI master/target 66MHz 32bit | | | | | |
| **Core Requirements** | FPGA Families Supported | LatticeEC LatticeECP | Lattice ECP2 Lattice ECP2M | LatticeXP | LatticeXP2 | LatticeXP3 | LatticeSC LatticeSCM |
| | Minimal Device Needed | LFEC6E-5F256C | LFE2-6E-6F256C | LFXP6C-5F256C | LFXP2-5E-6FT256C | LFE3-17EA-7FN484CES | LFSC3GA15E-6F900C |
| **Resource Utilization** | Data Path Width | 32 | | | | | |
| | LUTs | 1600 | | | | | |
| | Registers | 700 | | | | | |
| **Design Tool Support** | Lattice Implementation | Diamond 1.0 or ispLEVER 8.1 | | | | | |
| | Synthesis | Synopsys Synplify Pro for Lattice D-2009.12L-1 | | | | | |
| | | Mentor Graphics Precision RTL | | | | | |
| | Simulation | Aldec Active-HDL 8.2 Lattice Edition | | | | | |
| | | Mentor Graphics ModelSim SE 6.3F | | | | | |

*Table 1-3. PCI IP Core Quick Facts--PCI master/target 33MHz/64bit*

| | | PCI IP configuration | | | | | |
|---|---|---|---|---|---|---|---|
| | | PCI master/target 33MHz 64bit | | | | | |
| **Core Requirements** | FPGA Families Supported | LatticeEC LatticeECP | Lattice ECP2 Lattice ECP2M | LatticeXP | LatticeXP2 | LatticeXP3 | LatticeSC LatticeSCM |
| | Minimal Device Needed | LFEC10E-5F484C | LFE2-12E-6F484C | LFXP15C-5F388C | LFXP2-17E-6F484C | LFE3-35EA-7FN484CES | LFSC3GA15 E-6F900C |
| **Resource Utilization** | Data Path Width | 64 | | | | | |
| | LUTs | 1400 | | | | | |
| | Registers | 800 | | | | | |
| **Design Tool Support** | Lattice Implementation | Diamond 1.0 or ispLEVER 8.1 | | | | | |
| | Synthesis | Synopsys Synplify Pro for Lattice D-2009.12L-1 | | | | | |
| | | Mentor Graphics Precision RTL | | | | | |
| | Simulation | Aldec Active-HDL 8.2 Lattice Edition | | | | | |
| | | Mentor Graphics ModelSim SE 6.3F | | | | | |

*Table 1-4. PCI IP Core Quick Facts--PCI master/target 33MHz/32bit*

| | | PCI IP configuration | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | PCI master/target 33MHz 32bit | | | | | | | |
| **Core Requirements** | FPGA Families Supported | MachXO | MachXO2 | LatticeEC LatticeECP | Lattice ECP2 Lattice ECP2M | LatticeXP | LatticeXP2 | LatticeXP3 | LatticeSC LatticeSCM |
| | Minimal Device Needed | LCMXO1200E-5FT256C | LCMXO-1200HC-6TG144CES | LFEC6E-5F256C | LFE2-6E-6F256C | LFXP6C-5F256C | LFXP2-5E-6FT256C | LFE3-17EA-7FN484CES | LFSC3GA15E-6F900C |
| **Resource Utilization** | Data Path Width | 32 | | | | | | | |
| | LUTs | 900 | | | | | | | |
| | Registers | 600 | | | | | | | |
| **Design Tool Support** | Lattice Implementation | Diamond 1.0 or ispLEVER 8.1 | | | | | | | |
| | Synthesis | Synopsys Synplify Pro for Lattice D-2009.12L-1 | | | | | | | |
| | | Mentor Graphics Precision RTL | | | | | | | |
| | Simulation | Aldec Active-HDL 8.2 Lattice Edition | | | | | | | |
| | | Mentor Graphics ModelSim SE 6.3F | | | | | | | |

*Table 1-5. PCI IP Core Quick Facts--PCI target 66MHz/64bit*

| | | PCI IP configuration | | | | | |
|---|---|---|---|---|---|---|---|
| | | PCI target 66MHz 64bit | | | | | |
| **Core Requirements** | FPGA Families Supported | LatticeEC LatticeECP | Lattice ECP2 Lattice ECP2M | LatticeXP | LatticeXP2 | LatticeXP3 | LatticeSC LatticeSCM |
| | Minimal Device Needed | LFEC6E-5F484C | LFE2-12E-6F484C | LFXP10C-5F388C | LFXP2-8E-6FT256C | LFE3-17EA-7FN484CES | LFSC3GA15E-6F900C |
| **Resource Utilization** | Data Path Width | 64 | | | | | |
| | LUTs | 1300 | | | | | |
| | Registers | 600 | | | | | |
| **Design Tool Support** | Lattice Implementation | Diamond 1.0 or ispLEVER 8.1 | | | | | |
| | Synthesis | Synopsys Synplify Pro for Lattice D-2009.12L-1 | | | | | |
| | | Mentor Graphics Precision RTL | | | | | |
| | Simulation | Aldec Active-HDL 8.2 Lattice Edition | | | | | |
| | | Mentor Graphics ModelSim SE 6.3F | | | | | |

*Table 1-6. PCI IP Core Quick Facts--PCI target 66MHz/32bit*

| | | PCI IP configuration | | | | | |
|---|---|---|---|---|---|---|---|
| | | PCI target 66MHz 32bit | | | | | |
| **Core Requirements** | FPGA Families Supported | LatticeEC LatticeECP | Lattice ECP2 Lattice ECP2M | LatticeXP | LatticeXP2 | LatticeXP3 | LatticeSC LatticeSCM |
| | Minimal Device Needed | LFEC3E-5Q208C | LFE2-6E-6F256C | LFXP3C-5Q208C | LFXP2-5E-6QN208C | LFE3-17EA-7FTN256CES | LFSC3GA15E-6F256C |
| **Resource Utilization** | Data Path Width | 32 | | | | | |
| | LUTs | 900 | | | | | |
| | Registers | 500 | | | | | |
| **Design Tool Support** | Lattice Implementation | Diamond 1.0 or ispLEVER 8.1 | | | | | |
| | Synthesis | Synopsys Synplify Pro for Lattice D-2009.12L-1 | | | | | |
| | | Mentor Graphics Precision RTL | | | | | |
| | Simulation | Aldec Active-HDL 8.2 Lattice Edition | | | | | |
| | | Mentor Graphics ModelSim SE 6.3F | | | | | |

*Table 1-7. PCI IP Core Quick Facts--PCI target 33MHz/64bit*

| | | PCI IP configuration | | | | | |
|---|---|---|---|---|---|---|---|
| | | PCI target 33MHz 64bit | | | | | |
| **Core Requirements** | FPGA Families Supported | LatticeEC LatticeECP | Lattice ECP2 Lattice ECP2M | LatticeXP | LatticeXP2 | LatticeXP3 | LatticeSC LatticeSCM |
| | Minimal Device Needed | LFEC6E-5F484C | LFE2-12E-6F484C | LFXP10C-5F388C | LFXP2-8E-6FT256C | LFE3-17EA-7FN484CES | LFSC3GA15E-6F900C |
| **Resource Utilization** | Data Path Width | 64 | | | | | |
| | LUTs | 800 | | | | | |
| | Registers | 600 | | | | | |
| **Design Tool Support** | Lattice Implementation | Diamond 1.0 or ispLEVER 8.1 | | | | | |
| | Synthesis | Synopsys Synplify Pro for Lattice D-2009.12L-1 | | | | | |
| | | Mentor Graphics Precision RTL | | | | | |
| | Simulation | Aldec Active-HDL 8.2 Lattice Edition | | | | | |
| | | Mentor Graphics ModelSim SE 6.3F | | | | | |

*Table 1-8. PCI IP Core Quick Facts--PCI target 33MHz/32bit*

| | | PCI IP configuration | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | PCI target 33MHz 32bit | | | | | | | |
| **Core Requirements** | FPGA Families Supported | MachXO | MachXO2 | LatticeEC LatticeECP | Lattice ECP2 Lattice ECP2M | LatticeXP | LatticeXP2 | LatticeXP3 | LatticeSC LatticeSCM |
| | Minimal Device Needed | LCMXO1200E-5FT256C | LCMXO-1200HC-6TG144CES | LFEC3E-5Q208C | LFE2-6E-6F256C | LFXP3C-5Q208C | LFXP2-5E-6QN208C | LFE3-17EA-7FTN256CES | LFSC3GA15E-6F256C |
| **Resource Utilization** | Data Path Width | 32 | | | | | | | |
| | LUTs | 600 | | | | | | | |
| | Registers | 500 | | | | | | | |
| **Design Tool Support** | Lattice Implementation | Diamond 1.0 or ispLEVER 8.1 | | | | | | | |
| | Synthesis | Synopsys Synplify Pro for Lattice D-2009.12L-1 | | | | | | | |
| | | Mentor Graphics Precision RTL | | | | | | | |
| | Simulation | Aldec Active-HDL 8.2 Lattice Edition | | | | | | | |
| | | Mentor Graphics ModelSim SE 6.3F | | | | | | | |

## Features

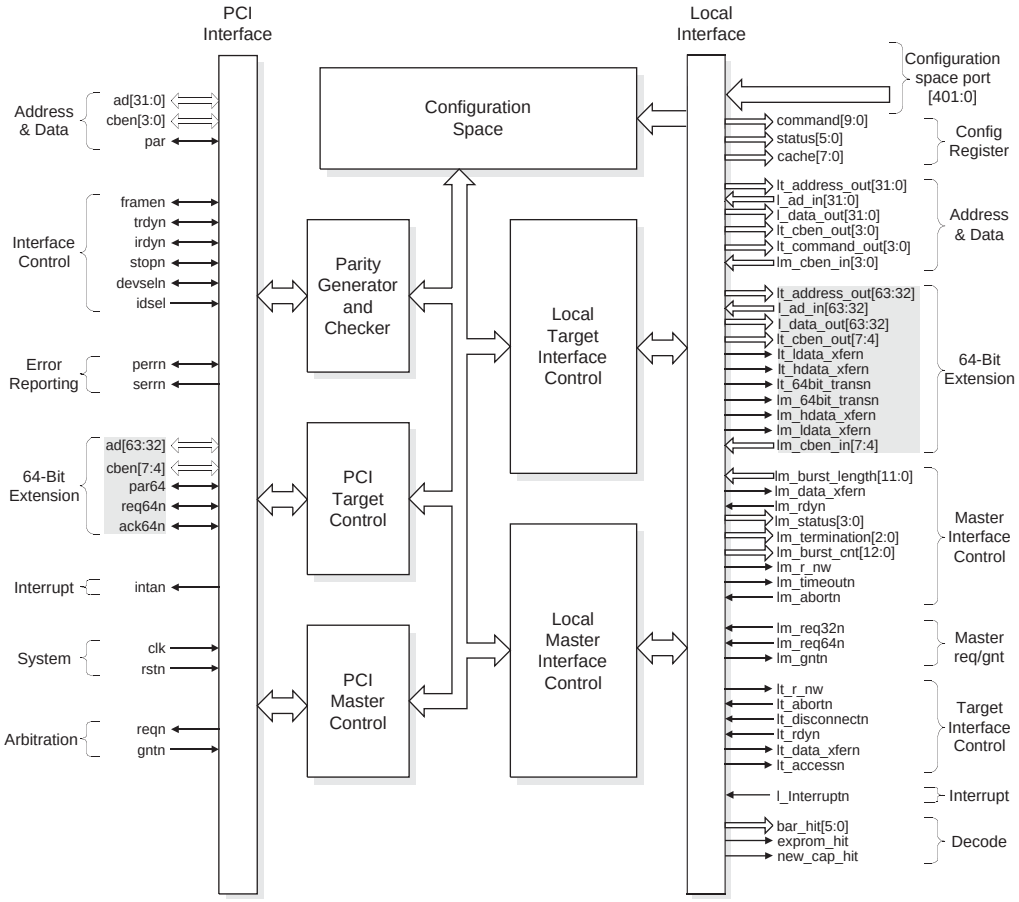
- Available as 32/64-bit PCI bus and 32/64-bit local bus
- PCI SIG Local Bus Specification, Revision 3.0 compliant
- 64-bit addressing support (dual address cycle)
- Capabilities list pointer support
- Parity error detection
- Up to six Base Address Registers (BARs)
- Fast back-to-back transaction support
- Supports zero wait state transactions
- Special cycle transaction support
- Customizable configuration space
- Up to 66MHz PCI
- Fully synchronous design

This chapter provides a functional description of the Lattice PCI IP core.

The PCI IP cores bridge the PCI bus to the back-end application. They decode transactions and pass PCI requests to the Local Interface. The back-end applications then send or receive the proper data associated with the PCI Interface via their Local Interface to respond to the PCI transactions. In the case of master versions the core executes PCI bus transactions based on back-end requests. Figure 2-1 illustrates the functional modules and internal bus structure used in the PCI IP core.

## Block Diagram

***Figure 2-1. PCI IP core Block Diagram***




Note: Signals in shaded boxes are used for 64-bit PCI Cores.

The PCI Master Target IP Core consists of multiple blocks, as shown in Figure 2-1. This section provides a detailed description of these blocks.

### PCI Master Control

The PCI Master Control interfaces with the PCI bus. It supports all of the address and command signals required to execute transactions on the PCI bus for both 32-bit and 64-bit PCI applications. A list of the supported PCI signals is available in the PCI Interface Signals section of this document. Once the Local Master Interface Control is granted the bus, it passes the transaction information to the PCI Master Control using the internal bus. The PCI Master Control then requests and executes the transaction on the PCI bus. The PCI IP cores support all of the

commands specified in the *PCI Local Bus Specification, Revision 3.0.* Table 2-1 lists the supported PCI commands.

***Table 2-1. PCI IP Core Command Support***

| cben[3:0] | Command | Support |
|---|---|---|
| 0000 | Interrupt Acknowledge | Yes |
| 0001 | Special Cycle | Yes |
| 0010 | I/O Read | Yes |
| 0011 | I/O Write | Yes |
| 0100 | Reserved | Ignored |
| 0101 | Reserved | Ignored |
| 0110 | Memory Read | Yes |
| 0111 | Memory Write | Yes |
| 1000 | Reserved | Ignored |
| 1001 | Reserved | Ignored |
| 1010 | Configuration Read | Yes |
| 1011 | Configuration Write | Yes |
| 1100 | Memory Read Multiple | Yes |
| 1101 | Dual Address Cycle | Yes |
| 1110 | Memory Read Line | Yes |
| 1111 | Memory Write and Invalidate | Yes |

The PCI Master control supports data transfer requirements for both high and low throughput back-end applications. It maintains up to the maximum 528 MBytes per second (MBps) burst data transfer rate when operating at 66MHz with a 64-bit data bus. The Advanced Master Transactions section of this document describes burst data transfers in further detail. For slower applications, single data phase transactions can also be easily implemented. The Basic PCI Master Read and Write Transactions section describes these basic transactions in detail.

## PCI Target Control

The PCI Target control interfaces with the PCI bus. It processes the address, data, command and control signals to transfer data to and from the PCI IP core for both 32-bit and 64-bit PCI applications. A list of the supported PCI signals is available in the PCI Interface Signals section. Once the PCI Target control detects a transaction, it passes the transaction information to the Local Interface control using the internal bus. It also responds to most Configuration Space accesses with no intervention from the Local Interface. The PCI IP core supports all of the commands specified in the *PCI Local Bus Specification, Revision 3.0*. Table 2-1 lists the supported PCI commands.

When designing for a particular target application, the back-end target design may not support all the commands listed in Table 2-1. As a result, the PCI IP core does not transfer data using those commands. For cases where the back-end target application does not support all the commands, it must issue the proper termination as described in the Target Termination section of this document.

The PCI Target control supports the data transfer requirements for both high and low throughput back-end applications. It can maintain a 528 MBps transfer rate during burst transactions when operating at 66MHz with a 64-bit data bus. The Advanced Target Transactions section describes the Burst transactions in further detail. For slower applications, single data phase transactions can also be easily implemented. The Basic PCI Target Read and Write Transactions section describes the these basic transactions in detail

## Local Master Interface Control

The Local Master Interface facilitates master transactions on the PCI Bus with the commands listed in Table 2-1. The Local Master Interface Control passes the local master transaction request from the user's application to the PCI Master Control which then executes the PCI bus transaction.

### Local Target Control

The Local Target Control responds to target transactions on the PCI bus. Fully decoded BAR select signals (bar_hit) and new capabilities select signal (new_cap_hit) are provided by the Local Target Control to indicate that the PCI IP core has been selected for a transaction. Registered address and command signals are available at the Local Interface from the Local Interface Control for the back-end application to properly handle the core's request. Additionally, the Local interface also supplies Configuration Space Register signals and a local interrupt request (l_interruptn) for users' applications. A full list of Local Interface signals and descriptions is available in the Local Interface Signals section.

### Configuration Space

The Configuration Space implements all the necessary Configuration Space registers required to support a single-function PCI IP core. It provides the first 64 bytes of header type 0, which is used for all device types other than PCI-to-PCI and CardBus bridges. The first 64 bytes of the predefined header region contain fields that uniquely identify the device and allow the device to be generically controlled. This predefined header portion of the Configuration Space is divided into two parts. The first 16 bytes of the header are defined the same way regardless of the type of device. The remaining bytes have different definitions depending on the functionality that the PCI IP core supports. These bytes include six Base Address Registers (BARs), the Capabilities Pointer (Cap Ptr), and the registers that control the interrupt capability. Refer to the Configuration Space Set-up section for additional information on the Configuration Space.

Accesses to the first 64 bytes of the Configuration Space are completed by the PCI IP core control with no intervention from the Local Target Interface control. Access beyond the first 64 bytes, such as the Capabilities List, is left to the Local Target Interface control. These transactions are described in the Advanced Configuration Accesses section.

### Parity Generator and Checker

Parity checking must occur on every PCI address and data cycle to be compliant with the *PCI Local Bus Specification, Revision 3.0.* The PCI IP core's Parity Generator and Checker module does all parity checking for the PCI device. The Parity Generator and Checker determines if the master is successful in addressing the desired target. It also verifies that data transfers occur correctly between the master and target devices. The address and byte enable signals are included in every calculation to ensure accuracy. Each address and data cycle that occurs on the PCI bus is checked for errors.

The parity check signals perrn and serrn are enabled or disabled using bit 6 and bit 8 of the PCI Command Register, which is part of the Configuration Space.

## Signal Descriptions

Pin Assignments for the evaluation configurations are shown "Pin Assignments For Lattice FPGAs" on page 161. Final selection of the pinouts is left to the designer to allow for maximum flexibility in the design. Pinouts are defined in the HDL source code, or as follows:.

- In Diamond, choose **View > Show Views > File List**, double-click the.lpf file, and edit the file to add pin location preferences.
- In ispLEVER, double-click **Edit Preference (ASCII)** in the Processes window, and edit the file in the Text Editor to add pin location preferences.

Refer to the Diamond or ispLEVER software help for additional information.

There are five types of signals defined in Table 2-2.

*Table 2-2. Signal Types*

| Signal Type | Description |
|---|---|
| in | **Input** is a standard input only signal. |
| out | **Output** is a standard output only signal. |
| t/s | **Tri-state** is a bidirectional, tri-state input/output pin. |
| s/t/s | **Sustained Tri-State** is an active low tri-state signal owned and driven by one agent at a time. |
| o/d | **Open Drain** allows multiple devices to share as a wire-OR. A pull-up is required to sustain the inactive state until another agent drives it and must be provided by the central resource. |

## PCI Interface Signals

The PCI Interface signals correspond to the PCI bus specification. Table 2-3 shows the input and output signals for the PCI IP core. These are the signals required by the PCI IP core to handle PCI bus side transactions. Table 2-3 describes each signal.

In addition to the signals required by the PCI IP core, there are some signals on the PCI Bus, referred to as "Additional Signals" in the PCI specifications, which must be handled appropriately to insure proper PCI IP core functions in a system. Refer to the relevant PCI specifications for a description of those Additional Signals (which are beyond the scope of this document). Examples of this type of signal are `M66EN` and `PRSNT[1:0]`.

*Table 2-3. PCI IP Core Signals*[1]

| Name | I/O | Polarity | Description |
|---|---|---|---|
| **PCI System** | | | |
| `clk` | in | — | The PCI system clock provides timing for all transactions. The clock frequency operates up to 66MHz. This clock is also used to provide timing to the Local Interface. |
| `rstn` | in | low | The asynchronous PCI system reset is used to set the PCI device to a starting known and stable state. |
| **PCI Address and Data** | | | |
| `ad[31:0]` | t/s | — | The multiplexed address and data bus. |
| `cben[3:0]` | t/s | — | Multiplexed command and byte enable signals. |
| `par` | t/s | — | The `par` signal generates even parity for `ad[31:0]` and `cben[3:0]` signals |
| **PCI Interface Control** | | | |
| `framen` | s/t/s | low | The `framen` signal is driven by the current master and used to indicate the start of cycle and the duration of the cycle. |
| `irdyn` | s/t/s | low | The initiator ready signal indicates that the current master is ready for the data phase. |
| `trdyn` | s/t/s | low | The target ready signal indicates that the current target is ready for the data phase. |
| `stopn` | s/t/s | low | The PCI IP core, as a target, drives this signal low requesting to stop the current transaction. |
| `idsel` | in | — | The initialization device select is used to select a target for configuration reads and writes. |
| `devseln` | s/t/s | low | Device select is actively driven by the PCI IP core to indicate that it is the target of the bus transaction. |

*Table 2-3. PCI IP Core Signals[1] (Continued)*

| Name | I/O | Polarity | Description |
|---|---|---|---|
| **PCI Error Reporting** | | | |
| perrn | s/t/s | low | Data parity error is used to report parity errors in the data phase. |
| serrn | o/d | low | System error is used to indicate catastrophic errors. |
| **PCI Interrupt** | | | |
| intan | o/d | low | Interrupt A is used to request an interrupt. |
| **PCI Bus Arbitration** | | | |
| reqn | out | low | Request for the use of PCI bus. |
| gntn | in | low | Grant the master's access to PCI bus. |
| **PCI 64-Bit Extension** | | | |
| ad[63:32] | t/s | — | The upper 32 bits of multiplexed address and data bus. |
| cben[7:4] | t/s | — | The upper, multiplexed command and byte enable signals for 64-bit applications. |
| par64 | t/s | — | The par64 signal generates even parity for ad[63:32] and cben[7:4] signals. |
| req64n | s/t/s | low | Used by the master to request a 64-bit data transaction. |
| ack64n | s/t/s | low | Signal used to indicate the acknowledgement of a request for 64-bit data transaction. |

1. Shaded rows apply to 64-bit applications.

## Local Interface Signals

The Local Interface provides all the necessary address and control signals to respond to and initiate transactions associated with the PCI bus. Command and status information are also available at the Local Interface, so the back-end application logic can essentially monitor the PCI bus. Table 2-4 contains the Local Interface signals that are divided into three different categories: Local Bus Signals, Local Target Bus signals and Local Master Bus signals.

The Local Bus Signals are shared between the Local Master Interface and Local Target Interface. These signals are typically denoted with an "l_". The Local Target Bus signals are used by the Local Target Interface and are denoted using "lt_". The Local Master Bus signals are used by the Local Master interface and are denoted using "lm_".

*Table 2-4. Local Interface Signals[1]*

| Name | I/O | Polarity | Description |
|---|---|---|---|
| **Local Address and Data** | | | |
| l_ad_in[31:0] | in | — | Local address/data input. The address input is used in Master Read/Write transactions, and the data input is used for master write/target read transactions |
| l_data_out[31:0] | out | — | Local Data output. Local side lower DWORD data output for a master read or a target write. |
| lt_address_out [31:0] | out | — | The local address bus for target read and write. This bus indicates the start address of the transaction. The bus, lt_address_out [31:0], is latched one clock after the framen signal is asserted on each transaction and remains unchanged until the next transaction. |
| lt_cben_out[3:0] | out | low | The local byte enables for target read and write. The lt_cben_out[3:0] determine which byte lanes of l_data_out[31:0] or l_ad_in[31:0] carry meaningful data. |
| lt_command_out[3:0] | out | — | The lt_command_out[3:0] latches the command information during the address phase of a PCI cycle. It indicates the PCI bus command for the current cycle (refer to Table 2-1). |
| lm_cben_in[3:0] | in | low | Local master command and byte enables. |

*Table 2-4. Local Interface Signals[1] (Continued)*

| Name | I/O | Polarity | Description |
|---|---|---|---|
| **Local 64-Bit Extension** | | | |
| l_ad_in[63:32] | in | — | Local address/data input. The address input is used in Master Read/Write transactions, and the data input is used for master write/target read transactions. |
| l_data_out[63:32] | out | — | Local Data output. Local side upper DWORD data output for a master read or a target write. |
| lt_address_out [63:32] | out | — | The local address bus for target read and write. This bus is valid only for 64bit address bar. The 64-bit combined signal lt_address_out [63:0] indicates the start address of the transaction. The high 32bit of the bus, lt_address_out[63:32], is latched two clock cycles after the framen signal is asserted on each transaction (only for dual address cycle) and remains unchanged until the next transaction. |
| lt_cben_out[7:4] | out | low | The local byte enables for 64-bit target read and write. The lt_cben_out[7:4] determine which byte lanes of l_data_out[63:32] or l_ad_in[63:32] carry meaningful data. |
| lt_ldata_xfern | out | low | This signal works same as lt_data_xfern. It applies to lower DWORD when local bus is 64bit. |
| lt_hdata_xfern | out | low | This signal works same as lt_data_xfern. It applies to upper DWORD when local bus is 64bit. |
| lt_64bit_transn | out | low | Signal to the local target that a 64-bit read or write transaction is underway on pci bus. |
| lm_ldata_xfern | out | low | This signal works same as lm_data_xfern. It applies to lower DWORD when local bus is 64bit. |
| lm_hdata_xfern | out | low | This signal works same as lm_data_xfern. It applies to upper DWORD when local bus is 64bit. |
| lm_64bit_transn | out | low | Signal to the local master that a 64-bit read or write transaction is underway on PCI bus. |
| lm_cben_in[7:4] | in | low | Local master byte enables. |
| **Local Interrupt** | | | |
| l_interruptn | in | low | The local side interrupt request indicates that the Local Interface is requesting an interrupt. This signal asserts the PCI side interrupt signal, intan, if interrupts are enabled in the Configuration Space. |
| **Config Register** | | | |
| cache[7:0] | out | — | The cache signal indicates the cache length in the cache registers defined in the Configuration Space |
| command[9:0] | out | — | Command register bits from the Configuration Space.<br>Bit 0 - I/O space enable, Command[0]<br>Bit 1 - Memory space enable, Command[1]<br>Bit 2 - Master enable, Command[2]<br>Bit 3 - Special cycles enable, Command[3]<br>Bit 4 - Memory write and invalidate enable, Command[4]<br>Bit 5 - VGA Palette Snoop, Command[5]<br>Bit 6 - Parity Error Response, Command[6]<br>Bit 7 - Reserved<br>Bit 8 - SERR# enable, Command[8]<br>Bit 9 - Fast back-to-back enable, Command[9] |
| status[5:0] | out | — | Status register bits from the Configuration Space.<br>Bit 0 - Master Data Parity Error, Status[8]<br>Bit 1 - Signaled Target Abort, Status[11]<br>Bit 2 - Received Target Abort, Status[12]<br>Bit 3 - Received Master Abort, Status[13]<br>Bit 4 - Signaled System Error with SERR#, Status[14]<br>Bit 5 - Detected Parity Error, Status[15] |

*Table 2-4. Local Interface Signals[1] (Continued)*

| Name | I/O | Polarity | Description |
|---|---|---|---|
| **Local Target Interface** | | | |
| lt_abortn | in | low | Local target abort request is used to request a target abort on the PCI bus. |
| lt_disconnectn | in | low | Local target disconnect (or retry) is used to request early termination of a bus transaction on the PCI bus. |
| lt_rdyn | in | low | Local target ready signal indicates that the Local Interface is ready to receive or send data. |
| lt_r_nw | out | — | Read/Write (read/not write) to signal whether the current transaction is a read or write. 1 = read, 0 = write |
| lt_accessn | out | low | Local target can access local interface if lt_accessn is active. Once lt_accessn active, local target needs to be ready for next process based on lt_command_out. lt_accessn is active during either of active bar_hit, exprom_hit or new_cap_hit. It is also active during special cycle command. |
| lt_data_xfern | out | low | This signal indicates local input data (l_ad_in) being read or local output data (l_data_out) being available at current clock cycle. When lt_data_xfern is active, if core reads data from l_ad_in, back-end can update l_ad_in for next data at next clock cycle. If core writes data on l_data_out, back-end can get valid data from l_data_out. It is only used when the local bus is 32 bits. |
| **Local Target Address Decode** | | | |
| bar_hit[5:0] | out | high | The bar_hit signal indicates that the master is requesting a transaction that falls within one of the Base Address register ranges. |
| new_cap_hit | out | high | New Capabilities List hit. new_cap_hit indicates that the master is requesting a Configuration Space register out of internal registers (00h-3fh), that is 40h-FFh., Although the hardware associated with the New Capabilities reside in the back-end logic, logically they are part of the PCI Configuration Space. |
| **Local Master req/gnt** | | | |
| lm_req32n | in | low | Local master 32-bit data transaction request. |
| lm_req64n | in | low | Local master 64-bit data transaction request. |
| lm_gntn | out | low | Signal to the local master that gntn is asserted. |
| **Local Master Interface Control** | | | |
| lm_rdyn | in | low | Local master is ready to receive data (read) or send data (write) |
| lm_burst_length [11:0] | in | — | Local master burst length determines the number of data phases in the transaction. For single data phase, it should be set to 1. lm_burst_length set to 0 means the burst length is 13'b1,0000,0000,0000. |
| lm_data_xfern[3] | out | low | This signal indicates local input data (l_ad_in) being read or local output data (l_data_out) available at current clock cycle. When lt_data_xfern is active, if core reads data from l_ad_in, back-end can update l_ad_in for next data at next clock cycle. If core writes data on l_data_out, back-end can get valid data from l_data_out. It is only used when the local bus is 32 bits. In a single data phase, it should be set to 1. lm_burst_length set by 0 means the length is 13'b1,0000,0000,0000. |
| lm_r_nw | out | — | (Read/Write) to signal whether the current transaction is a read or write. 1 = read, 0 = write |
| lm_timeoutn | out | — | Indicates that the transaction has timed out. |
| lm_abortn | in | — | Local master issues an abort to terminate a cycle that can not be completed. |

*Table 2-4. Local Interface Signals[1] (Continued)*

| Name | I/O | Polarity | Description |
|---|---|---|---|
| lm_status[3:0] | out | — | Indicate the master operation status.<br>0001 - Address loading<br>0010 - Bus transaction<br>0100 - Bus termination<br>1000 - Fast_back_to_back |
| lm_termination[2:0] | out | — | Indicate the master termination status.<br>000 - Normal termination<br>Normal termination occurs when the master finishes and completes the transaction normally. During a multi-data phase transfer, a condition can occur where the master's latency timer expires on the last data phase and master's gntn has been de-asserted. In this case, lm_timeoutn is asserted and the master also indicates this as Normal termination.<br><br>001 - Timeout termination<br>Timeout termination occurs when, during a multi-data phase transfer, the master's latency timer expired before the last data phase and the master's gntn is de-asserted on or before the last data phase. lm_timeoutn is also asserted in this case.<br><br>010 - No target response termination.<br>This is also known as Master Abort termination.<br><br>011 - Target abort termination<br>100 - Retry termination<br>101 - Disconnect data termination<br>110 - Grant abort termination<br>111 - Local master termination |
| lm_burst_cnt[12:0] | out | — | Local master burst transaction "down counter" value. When the local master requests a 32-bit transaction, the initial value of lm_burst_cnt is equal to lm_burst_length. When the local master requests a 64-bit transaction and lm_64bit_transn is active, the initial value of lm_burst_cnt is equal to lm_burst_length. When the local master requests a 64-bit transaction and lm_64bit_transn is inactive, the initial value of lm_burst_cnt is double of lm_burst_length. |

1. Shaded rows apply to 64-bit applications.
2. A Memory Write and Invalidate transaction is not governed by the Latency Timer except at cacheline boundaries. A master that initiates a transaction with the Memory Write and Invalidate command ignores the Latency Timer until a cacheline boundary. When the Latency Timer has expired (and gntn is deasserted), the core asserts lm_timeoutn. The backend must terminate the transaction at next cacheline boundary by asserting lm_abort.
3. During Master Read operation the signal lm_data_xfern always reflects valid data in the local data bus. But during Master Write operation, due to data prefetch ahead of the transactions on PCI bus, lm_data_xfern along with the lm_status reflects the data validity. If lm_status is 0100, (meaning a Bus Termination) ignore the lm_data_xfern assertion because the data being prefetched is not sent out on the PCI bus due to termination.

## PCI Configuration Space Setup

Determining the correct settings for the Configuration Space is an essential step in designing a PCI application, because the device may not function properly if the Configuration Space is not properly configured. The PCI IP core supports all of the required and some additional Configuration Space registers that apply to the PCI IP core (refer to *PCI Local Bus Specifications, Revision 3.0*, Chapter 6). Figure 2-2 shows the supported Configuration Space for the PCI IP core. This section describes the first 64 bytes of the Configuration Space in the PCI IP core and its customization method. For more information on the parameters used to customize the Configuration Space, refer to the Lattice PCI IP core Configuration Options section.

***Figure 2-2. PCI IP Core Configuration Space***

| | | | | |
|---|---|---|---|---|
| Device ID | | Vendor ID | | 00h |
| Status Register | | Command Register | | 04h |
| Class Code | | | Revision ID | 08h |
| BIST | Header Type | Latency Timer | Cache Line Size | 0Ch |
| Base Address 0 | | | | 10h |
| Base Address 1 | | | | 14h |
| Base Address 2 | | | | 18h |
| Base Address 3 | | | | 1Ch |
| Base Address 4 | | | | 20h |
| Base Address 5 | | | | 24h |
| Cardbus CIS Pointer | | | | 28h |
| Subsystem ID | | Subsystem Vendor ID | | 2Ch |
| Expansion ROM Base Address | | | | 30h |
| Reserved | | | Cap Ptr | 34h |
| Reserved | | | | 38h |
| MAX_LAT | MIN_GNT | Interrupt Pin | Interrupt Line | 3Ch |

Note: Shaded sections indicate reserved and unused sections in the configuration space. All unused and reserved registers return 0s.

**Vendor ID:** The Vendor ID is a 16-bit, read-only field used to identify the manufacturer of the product. The Vendor ID is set using the `VENDOR_ID` parameter. The Vendor ID is assigned by the PCI SIG to ensure uniqueness. Contact PCI SIG (www.pcisig.org) to obtain a unique Vendor ID.

**Device ID:** The Device ID is a 16-bit, read-only field that is defined by the manufacturer used to uniquely identify a particular product or model. The Device ID is set using the `DEVICE_ID` parameter. Its default value is 0000h.

**Revision ID:** The Revision ID is an 8-bit, read-only device-specific field that is set using the `REVISION_ID` parameter. This field is used by the manufacturer and should be viewed as an extension of the Device ID to distinguish between different functional versions of a PCI product.

**Class Code:** The Class Code is a 24-bit, read-only register and is used to identify the generic functionality of a device. The value of this register is determined by the `CLASS_CODE` parameter. The Class Code is broken up into three bytes. The upper byte holds the base class code; the middle byte holds the sub-class code. In addition, the lower byte holds the programming interface. The Class Code information is located in the *PCI Local Bus Specification, Revision 3.0*. The default setting for this register is FF0000h.

**Command Register:** The Command Register is a 16-bit read/write register that provides coarse control over the device. It is located at the lower 16 bits of address 04h in the Configuration Space. Using this register, the memory and I/O space can be disabled to allow only configuration accesses. This register also controls the parity error response and the `serrn` signal. Figure 2-3 and Table 2-5 illustrate the command register that is implemented in the PCI IP core.

***Figure 2-3. Command Register***

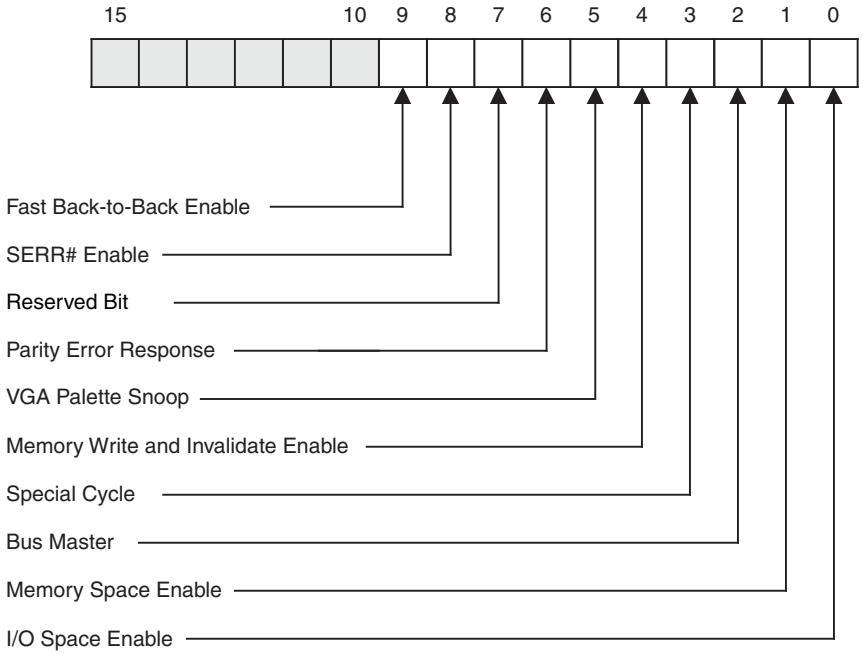



***Table 2-5. Command Register Description***

| Bit Location | Description |
|---|---|
| 0 | **I/O Space Enable** controls a device's response to I/O space accesses. I/O space accesses are enabled if the bit is set to a 1. After reset the I/O space enable bit is set to a 0. |
| 1 | **Memory Space Enable** controls a device's response to memory space accesses. Memory space accesses are enabled if the bit is set to a 1. After reset the memory space enable bit is set to a 0. |
| 2 | **Bus Master** enables the PCI IP core to act as a master on the PCI bus when this bit is set to 1. After reset the Bus Master enable bit is set to a 0. |
| 3 | **Special Cycle** controls a device's action on special cycle operations. Special cycle accesses are enabled if the bit is set to 1. After reset the bit is set to 0. |
| 4 | **Memory Write and Invalidate Enable** controls the PCI IP core's ability to execute the Memory Write and Invalidate cycle on the PCI bus. The Core, when required, will issue the Memory Write and Invalidate command if this bit is set to a 1. After reset this bit is set to a 0. |
| 5 | **VGA Palette Snoop** |
| 6 | **Parity Error Response** is used to control a device's response to parity errors. If the bit is 0, a parity error causes the Detected Parity Error status bit to be set in the status register but does not drive the perrn signal. After reset the bit is set to 0. This is the enable for parity error checking. However, even with the `perrn` signal disabled, the device is still required to generate parity. |
| 7 | **Reserved Bit** |
| 8 | **SERR Enable** is used to enable the `serrn` driver. To enable, this bit is set to a 1. After reset this bit is set to 0. |
| 9 | **Fast Back-to-Back Enable** allows the PCI IP core to execute fast back-to-back transactions to different devices. If the fast back-to-back enable is set to a 1, the Core executes fast back-to-back transactions. After reset this bit is set to 0. |
| 10-15 | **Reserved Bits** The returned value for these bits is 0 when this register is read. |

## Status Register

The Status Register is a 16-bit read/write register that provides information on the capabilities of the PCI IP core. It also reports the error status of the PCI IP core. The Status Register is located at the upper 16 bits of register location 04h. Writes to the Status Register from the PCI bus are slightly different, given that bits can be reset but not set. Writing a 1 to a bit in the status register resets it, but only if the current value of the bit is a 1. Writing a 0 to a bit has no effect. Figure 2-4 and Table 2-6 describe the Status Register that is implemented in the PCI IP core.

***Figure 2-4. Status Register***

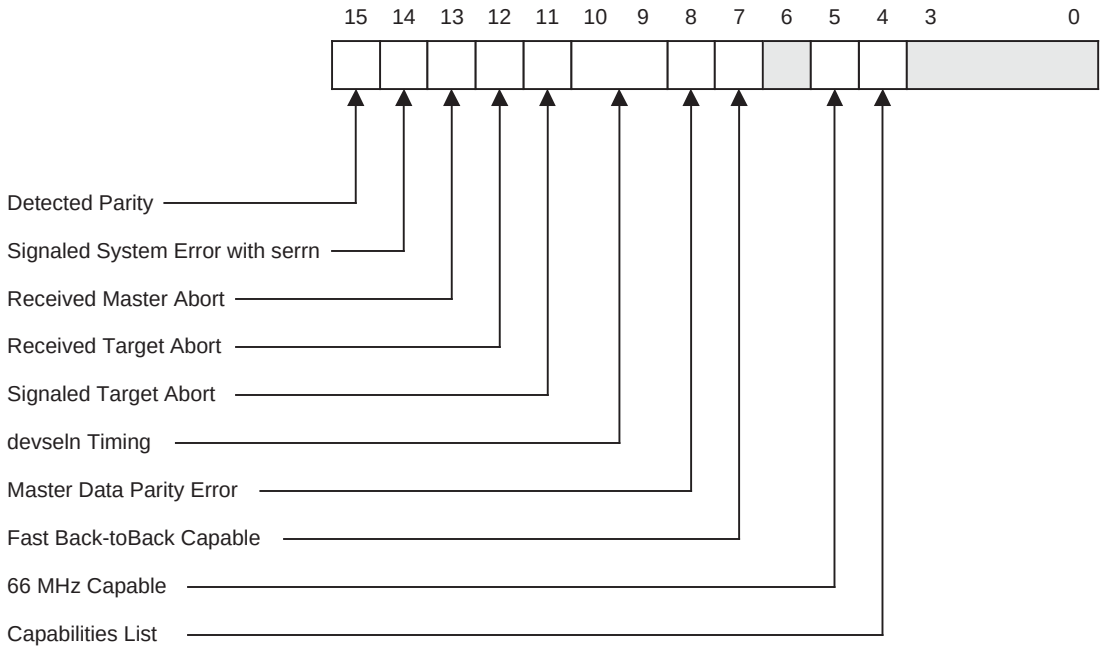



***Table 2-6. Status Register Descriptions***

| Bit Location | Description |
|---|---|
| 4 | **Capabilities List** is a read-only bit that indicates whether or not the device contains an address pointer to the start of the Capabilities list. The bit is set to a 1 to indicate that the Capabilities Pointer at location 34h is valid. After reset the value is set to a 0. The CAP_PTR_ENA parameter initializes this bit. |
| 5 | **66MHz Capable** is a read-only bit that is used to indicate that the device is capable of running at 66MHz. The bit is set to a 1 if the device is 66MHz capable. The PCI_66MHZ_CAP parameter initializes this bit. |
| 7 | **Fast Back-to-Back Capable** is a read-only bit that indicates if the device is capable of handling fast back-to-back transactions. The bit is set to a 1 if the device can accept these transactions. The FAST_B2B_CAP parameter initializes this bit. |
| 8 | **Master Data Parity Error** indicates that the bus master has detected a parity error during a transaction. A value of 1 means a parity error has occurred. After rest the bit is set to 0. |
| 9-10 | **DEVSEL Timing** bits indicate the slowest time for a device to assert the devseln signal for all accesses except the configuration accesses. The PCI IP core only supports the slow decode setting. The DEVSEL_TIMING parameter (bits 2 and 1) determines the DEVSEL timing.<br>00 - Fast (not supported)<br>01- Medium (not supported)<br>10 - Slow<br>11 - Reserved |
| 11 | **Signaled Target Abort** is set when the target terminates the cycle with a Target-Abort. Writing a 1 clears the Signaled Target Abort. |
| 12 | **Received Target Abort** is set to a 1 by the Core after it terminates a cycle with a target abort. |

*Table 2-6. Status Register Descriptions (Continued)*

| Bit Location | Description |
|---|---|
| 13 | **Received Master Abort** is set to a 1 by the Core after it terminates a cycle with a master abort with the exception of special cycles. |
| 14 | **Signaled System Error** with `serrn` is set when the device asserts serrn. Writing a one clears the bit. |
| 15 | **Detected Parity Error** is used to indicate a parity error even if the parity error handling is disabled. |
| 0,1,2,3,6 | **Reserved Bits** The returned value for each of these bits is 0 when this register is read. |

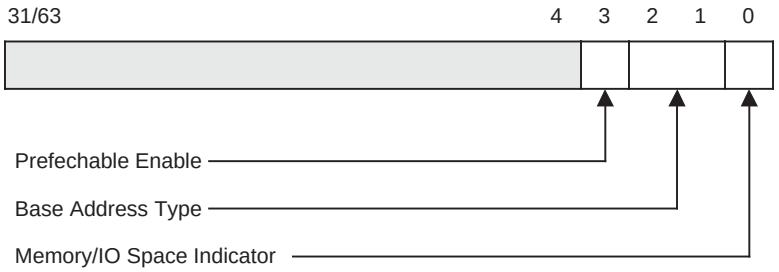
## Base Address Registers

The PCI IP core supports up to six Base Address Registers (BARs) for Master/Target and Target configurations. The BAR holds the base address for the PCI IP core, and it is used to point to the starting address of the PCI IP core in the system memory map. They are configured differently based on whether they are mapped in memory or I/O space. A memory location is addressed using 32 bits or 64 bits while I/O locations are limited to 32-bit addresses. The six BARs consist of 192 bits in the Configuration Space and are located in address locations 0x10 to 0x27.

## BAR Mapped to Memory Space

When selecting the amount of required memory for a BAR, the amount of memory is saved to the `BAR0-BAR5` parameters in its 2's complement form. Bits 0 through 3 of a memory BAR describes the attributes of the BAR and do not change. The minimum recommended amount of memory a BAR should request is 4Kbytes. Figure 2-5 and Table 2-7 describe the configuration of a BAR for memory space.

*Figure 2-5. Memory Base Address Register*




SERR Enable is used to enable the `serrn` driver. To enable, this bit is set to a 1. After reset this bit is set to 0.

*Table 2-7. Memory Base Address Register*

| Bit Location | Description |
|---|---|
| 0 | **Memory/I/O Space Indicator** indicates whether the base address is mapped to I/O or memory space. A 0 indicates mapping to the memory space. The value of this bit is set by bit 0 of the `BAR0-BAR5` parameters. |
| 1-2 | **Base Address Type** is used to determine whether the BAR is mapped into a 32-bit or 64-bit address space. These bits have the following meaning:<br>00 - located in 32-bit address space<br>01 - reserved<br>10 - located in 64-bit address space<br>11 - reserved |
| 3 | **Prefetchable Enable** is determined by bit 3. It is a read-only bit that indicates if the memory space is prefetchable. A value of 1 means the memory space is prefetchable. Bit 3 of the `BAR0-BAR5` parameters sets the value of this bit. |
| 4-31/63 | **Bits 4-31/63** are read/write to hold memory address and are initialized by the `BAR0-BAR5` parameters. |

## Bar Mapped to I/O Space

When selecting the amount of required I/O space for a BAR, the amount is saved to the BAR0-BAR5 parameters in its 2's complement form. Bits 0 and 1 of an I/O BAR describe the attributes of the BAR and do not change. Figure 2-6 and Table 2-8 describe the configuration of a BAR for I/O space.

***Figure 2-6. I/O Base Address Register***

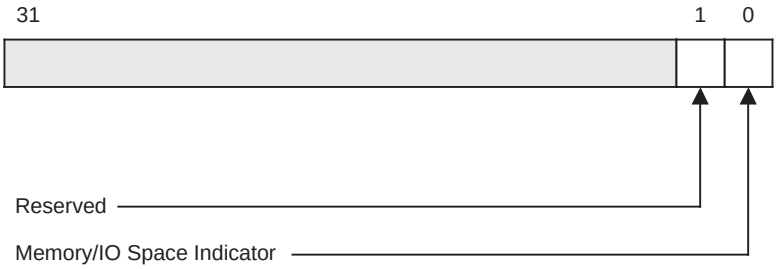



***Table 2-8. I/O Base Address Register***

| Bit Location | Description |
|---|---|
| 0 | **Memory/I/O space Indicator** indicates whether the base address is mapped to I/O or memory space. A 0 indicates mapping to the memory space. The value of this bit is set by bit 0 of the BAR0-BAR5 parameters. |
| 1 | **Bit 1** is reserved and hardwired to 0. This bit is read only. |
| 2-31 | **Bits 2-31** are read/write to hold the memory address and are initialized by the BAR0-BAR5 parameters. |

## Cache Line Size

The Cache Line Size register is an 8-bit read/write register, located at 0Ch. It specifies the Cache Line Size in Double Words (DWORDs). During a reset the register is set to 00h. This register is output to local interface as cache[7:0].

## Latency Timer

The Latency Timer register is an eight-bit read/write or read only register, located at byte address 0Dh. It specifies the Master Latency Timer value for a PCI Master on the PCI bus. During reset the register is set to 00h.

## CardBus CIS Pointer

The CardBus CIS Pointer is a read-only, 32-bit register at location 28h in the Configuration Space. The CIS_POINTER parameter determines the value of the register. For more information on the CardBus CIS Pointer, refer to the CardBus specification.

## Subsystem Vendor ID

The Subsystem Vendor ID is a 16-bit, read-only field and is used to further identify the manufacturer of the expansion board or subsystem. The SUBSYSTEM_VENDOR_ID parameter determines the value of the Subsystem Vendor ID register. The PCI SIG assigns the Vendor ID to ensure uniqueness. Contact PCI SIG (www.pcisig.org) to attain a unique Subsystem Vendor ID.

## Subsystem ID

The Subsystem ID is a 16-bit, read-only field and is used to further identify the particular device. This field is defined by the manufacturer and is used to uniquely identify products or models. The SUBSYSTEM_ID parameter determines the value of this register.

### Capabilities Pointer

The Capabilities Pointer indicates the starting location of the Capabilities List. It resides at address location 34h. The Capabilities Pointer consists of an 8-bit read-only register location. The capabilities pointer must be enabled by the CAP_PTR_ENA parameter. The CAP_POINTER parameter determines the value of this register.

### Min_Gnt

The Min_Gnt read-only register is an 8-bit field that is used to specify the length of time in microseconds for the Master to control the PCI bus. It resides in the upper 8 bits of address location 3Ch. The MIN_GRANT parameter determines the value of this register.

### Max_Lat

The Max_Lat read-only register is an 8-bit field that is used to specify the how often the PCI IP core the bus. It resides in the third byte of address location 3Ch. The MAX_LATENCY parameter determines the value of this register.

### Interrupt Line

The Interrupt Line register is set by the interrupt handling mechanism to define the interrupt routing. This is a read/write register is handled outside the operation of the PCI IP core. This register holds system interrupt routing information.

### Interrupt Pin

The Interrupt Pin register is used to indicate which of the four interrupts that the PCI IP core uses. Because the PCI IP core is a single function device, the only Interrupt Pin that can be selected is Interrupt A. If the interrupt is selected, the INTERRUPT_PIN parameter sets the register with a value of 01h. This eight-bit register is located at address location 3Dh.

### Reserved

All reserved registers are read-only. Write operations to reserved registers are completed normally, and the data is discarded. A 0 is returned after the read operations to reserved registers are completed normally.

## Lattice PCI IP core Configuration Options

Lattice PCI IP core allows an extensive definition of the PCI Configuration Space for optimum performance.

### IPexpress User-Controlled Configurations

The IPexpress user-configurable flow provides evaluation capability for any valid combination of parameters. Configurations can have a maximum of three BARs. To create a configuration with more than three BARs, contact Lattice.

The evaluation configurations of PCI IP core have a maximum of three BARs. To order a configuration with more than three BARs, contact Lattice.

***Table 2-9. IPexpress Parameters for PCI IP Core***

| Parameter Name | Range | Default(s) |
|---|---|---|
| Number of BARs | 1-6 | 3 |
| **Bus Definition Parameters** | | |
| PCI Data Bus Size | 32- or 64-bit | Note 1 |
| Local Master Data Bus Size | 32- or 64-bit | Note 2 |
| Local Target Data Bus Size | 32- or 64-bit | Note 2 |

***Table 2-9. IPexpress Parameters for PCI IP Core***

| Parameter Name | Range | Default(s) |
|---|---|---|
| Local Address Bus Width | 32- or 64-bit | 32-bit |

1. The value for PCI Data Bus size is set in each eval configuration as described in the appendices of the PCI IP core data sheet.
2. For 32-bit PCI Data Bus, only 32-bit Local Data Bus sizes are supported. For 64-bit PCI Data Bus, only 64-bit Local Data Bus sizes are supported.

## PCI Configuration Using Core Configuration Space Port

A set of signals called the Configuration Space Port is provided at the local bus side of the core to allow the user to define the PCI configuration space as required for the user's system. The names of these Core configuration input signals are all suffixed with _p.

Appropriate parameter values are to be assigned to the designated input signals of Core configuration space port to implement the desired PCI configuration space. Here are two examples to achieve this:

1. Directly assign parameter values to the input signals of Core configuration space port. The user needs to provide hard coded values to the Core's Configuration Space Port input signals in the core instantiation.

```
module pci_top();
  …
  customer_design   cus_design _inst(
              .xxxx(xxxx),
              .yyyy(yyyy),
              …….
              .zzzz(zzzz)
              ;


  pci_core     core_inst( .framen(framen),
              …
              .vdr_id_p(16'h1234),    //vendor_id = 16'h1234
              …
              .dev_tim_p(2'b10),       //devsel_timing = 2'b10  (slow)
              …
              );
endmodule
```

2. Typically, two Verilog files, para_cfg.v and PCI_params.v, can be used to load these parameters to Core's Configuration Space Port. These files are available in Lattice PCI IP release package.
   - Edit the PCI_params.v to set correct values to the parameters. Parameter names in PCI_params.v are all suffixed with _g. Alternatively use the PCI GUI provided with Lattice's software design tools to generate the PCI_params.v. Refer the note given below.
   - Instantiate para_cfg module and appropriately connect its ports to the Core configuration input signals of PCI IP core.

   para_cfg module will load the parameters, defined in PCI_params.v, into the Core's Configuration Space Port input signals.

```
module pci_top();
  …
  wire   [15:0]   vdr_id;
  wire   [ 1:0]    dev_tim;
  …
  …
```

```
    customer_design_instantiation(
                  .xxxx(xxxx),
                  .yyyy(yyyy),
                  …….
                  .zzzz(zzzz)
                  ;

    pci_core      core_inst(
                  . framen(framen),
                  …
                  . vdr_id_p (vdr_id),     //vendor_id = VENDOR_ID_g
                  …
                  . dev_tim_p (dev_tim),    //devsel_timing = DEVSEL_TIMING_g
                  …
                  );

    para_cfg       para_cfg_inst(.vdr_id_p (vdr_id),
                  …
                  . dev_tim_p (dev_tim),
                  …
                  );
endmodule
```

Table 2-10 shows the parameter signals and associated define values in PCI_params.v.

NOTE: There is a GUI provided with this PCI IP for the purpose of selecting the core's parameter values. Once installed into Lattice's Diamond or ispLEVER software design tools, the GUI can be accessed through the IPexpress™ tool. The GUI provides a range checking routine that ensures the selected values are within the core's valid range. If the user configures this PCI IP core outside of the GUI flow, it is the user's responsibility to ensure that the parameter values are within the valid ranges shown in Table 2-10. Parameters that are outside of the valid range will cause the PCI IP core to function improperly. The recommended flow is to follow example 2 above and use the PCI GUI to generate the params.v file. The name of this generated output file is fixed at "PCI_params.v". The user can find this file in the same directory where the <*modulename*>.lpc file is generated. Then, to prevent from overwriting the PCI_params.v file, the user should save a copy that is renamed to match the <*modulename*>.lpc file.

***Table 2-10. Customer Specific Parameters***

| Configuration Space Port Inputs | Corresponding Parameter Name in PCI_params.v | Range | Default | Description |
|---|---|---|---|---|
| vdr_id_p | VENDOR_ID_g | 0 - 0xFFFE | 0x0000 | Value of the Vendor ID field in the Configuration Space. This sets the lower 16 bits of address 00h. |
| dev_id_p | DEVICE_ID_g | 0 - 0xFFFF | 0x0000 | Value of the Device ID field in the Configuration Space. This sets the upper 16 bits of address 00h. |
| subs_vdr_id_p | SUB_VENDOR_ID_g | 0 - 0xFFFF | 0x0000 | Value for Subsystem Vendor ID field in the Configuration Space. The Subsystem Vendor ID is at the lower 16 bits of register location 2Ch. |
| subs_id_p | SUB_SYSTEM_ID_g | 0 - 0xFFFF | 0x0000 | Value for Subsystem ID field in the Configuration Space. The Subsystem ID is located in the upper 16 bits of address 2Ch. |

*Table 2-10. Customer Specific Parameters (Continued)*

| Configuration Space Port Inputs | Corresponding Parameter Name in PCI_params.v | Range | Default | Description |
|---|---|---|---|---|
| rev_id_p | REVISION_ID_g | 0 - 0xFF | 0x00 | Value for Revision ID field in the Configuration Space. This value correlates to the lower 8 bits of register 08h. |
| cls_code_p | CLASS_CODE_g | 0 - 0xFFFFFF | 0x000000 | Value for Class Code field in the Configuration Space. This is the value of the upper 24 bits of register 08h. Class Code is further subdivided into Base Class, Sub Class, and Interface values. Refer to the PCI local bus specification for valid Class codes. |
| dev_tim_p | DEVSEL_TIMING_g | 00 - 2'b10 | 2'b10 | Controls bits 9 and 10 in the status register, located at the upper 16 bits of address 04h. This parameter is used to define the decode speed of the PCI IP core.<br>00 - Fast (not supported)<br>01 - Medium (not supported)<br>10 - Slow<br>11 - Reserved |
| cap_list_ena_p | CAPABILITIES_LIST_ ENA_g | Enabled/Disabled | Enabled | Enable for the Capabilities Pointer. It is used to set the enable bit for the Capabilities List in the PCI Status register. This bit is used to indicate if the value of the Capabilities Pointer at location 34h is valid. This is bit 4 in the status register. |
| cap_ptr_p | CAPABILITIES_POINTER_g | 0x0 - 0xFF | 0x40 | Value for Capabilities Pointer field in the Configuration Space. This is an 8-bit value located at 34h. |
| cis_ptr_p | CIS_POINTER_g | 0 - 0xFFFFFFFF | 0x00000000 | Value for the Cardbus CIS Pointer field in the Configuration Space. This is a 32-bit value located at 28h in the Configuration Space Settings for the CIS Pointer are beyond the scope of this document. For more information on setting this register, refer to the CardBus specification. |
| fast_b2b_cap_p | FAST_B2B_CAP_g | Enabled/ Disabled | Enabled | Value for the Status field bit to enable fast back-to-back transfers. This is bit 7 of the status register. |
| irq_ack_ena_p | IRQ_ACK_ENA_g | Enabled/ Disabled | Enabled | Enable response to the Interrupt Acknowledge PCI command. |
| int_pin_p | INTERRUPT_PIN_g | 0x00 - 0x01 | 0x01 | Value for Interrupt Pin field in the Configuration Space. If set, it allows the local interrupt signal `l_interruptn` to appear on the PCI Interrupt `intan`. If the local interrupt is not used, it must be tied high. |
| hdw_lat_tmr_p | HARDWIRE_LATENCY_ TIMER_g | 0 - 0x10 | 0x00 | Value for read-only latency timer register |
| hdw_lat_tmr_ ena_p | HARDWIRE_LATENCY_ TIMER_ENA_g | Enabled/ Disabled | Disabled | Enable read only latency timer register. |
| min_gnt_p | MIN_GNT_g | 0 - 0xFF | 0x00 | Value for MIN_GRAND field in the configuration space. |
| max_lat_p | MAX_LAT_g | 0 - 0xFF | 0x00 | Value for MAX_LATENCY field in the configuration space. |

*Table 2-10. Customer Specific Parameters (Continued)*

| Configuration Space Port Inputs | Corresponding Parameter Name in PCI_params.v | Range | Default | Description |
|---|---|---|---|---|
| pci_66mhz_ cap_p | PCI_66MHZ_CAP_g | 33 or 66 | 66 | PCI value for the Status field bit to enable 66MHz. This is bit 5 in the status register. A 1 indicates that the PCI IP core is 66MHZ capable, and a 0 indicates that it is not. The default value is 1. |
| bar_64b_dat_ bus_p | BAR_64BIT_DATA_BUS_g | 6'b000000 - 6'b111111 | 6'b000000 | For 32-bit Local Data bus this parameter value is 6'b000000. For 64-bit Local Data bus this parameter value is 6'b111111. |
| bar0_p[1] | BAR0_g[1] | 0 - 0xFFFFFFFF | 0x00000000 | BAR0 configuration parameter (lower half of a 64-bit BAR). The lower four bits are used for BAR definition as indicated in the PCI specification. Rest of the bits indicate the memory or I/O size supported This BAR is located at 10h. If the bar is not used, it should be 32'h00000002. |
| bar1_ p[1] | BAR1_g[1] | 0 - 0xFFFFFFFF | 0x00000000 | BAR1 configuration parameter (upper half of a 64-bit BAR). The lower four bits are used for BAR definition as indicated in the PCI specification. Rest of the bits indicate the memory or I/O size supported This BAR is located at 14h. If the bar is not used, it should be 32'h00000002. |
| bar2_ p[1] | BAR2_g[1] | 0 - 0xFFFFFFFF | 0x00000000 | BAR2 configuration parameter (lower half of a 64-bit BAR). The lower four bits are used for BAR definition as indicated in the PCI specification. Rest of the bits indicate the memory or I/O size supported This BAR is located at 18h. If the bar is not used, it should be 32'h00000002. |
| bar3_ p[1] | BAR3_g[1] | 0 - 0xFFFFFFFF | 0x00000002 | BAR3 configuration parameter (upper half of a 64-bit BAR). The lower four bits are used for BAR definition as indicated in the PCI specification. Rest of the bits indicate the memory or I/O size supported This BAR is located at 1Ch. If the bar is not used, it should be 32'h00000002. |
| bar4_ p[1] | BAR4_g[1] | 0 - 0xFFFFFFFF | 0x00000002 | BAR4 configuration parameter (lower half of a 64-bit BAR). The lower four bits are used for BAR definition as indicated in the PCI specification. Rest of the bits indicate the memory or I/O size supported This BAR is located at 20h. If the bar is not used, it should be 32'h00000002. |
| bar5_ p[1] | BAR5_g[1] | 0 - 0xFFFFFFFF | 0x00000002 | BAR5 configuration parameter (upper half of a 64-bit BAR). The lower four bits are used for BAR definition as indicated in the PCI specification. Rest of the bits indicate the memory or I/O size supported This BAR is located at 24h. If the bar is not used, it should be 32'h00000002. |

*Table 2-10. Customer Specific Parameters (Continued)*

| Configuration Space Port Inputs | Corresponding Parameter Name in PCI_params.v | Range | Default | Description |
|---|---|---|---|---|

1. When using the second method to configure parameters, only the second column is used.
2. When a BAR is not used, its corresponding core configuration signal `bar_p` should be 32'h0000_0002. To achieve this:
   - BARx_g value in PCI_params.v should be 0x0000_0000.
   - The file para_cfg.v will translate value 0x0000_0000 to 0x0000_0002.

   In this case:
   - Content of that BAR register in PCI Configuration Space is 0x0000_0000

   The default values shown are the read back value (using PCI read command) of the enabled BAR after all 1's are written into that BAR (using PCI write command).

**Bar Configuration Details**

Memory Type:
Bit[0] BAR type = 0 (memory space indicator)
Bit[2:1] Base address type:
00 – Located in 32-bit address space
01 – Reserved
10 – Located in 64-bit address space
11 – Reserved
Bit[3] Prefetch 1= enable, 0=disable (This bit indicates whether or not the BAR can prefetch data from memory)
Bit[31:4] Memory size

I/O Type:
Bit[0] BAR type = 1 (I/O space indicator)
Bit[1] 0 – Reserved
Bit[31:2] Memory size

Bit[31:4/2] – memory or I/O size:

| | |
|---|---|
| 2Gbyte | 32'h8000_000_x |
| 1Gbyte | 32'hC000_000_x |
| 512Mbyte | 32'hE000_000_x |
| 256Mbyte | 32'hF000_000_x |
| 128Mbyte | 32'hF800_000_x |
| 64Mbyte | 32'hFC00_000_x |
| 32Mbyte | 32'hFE00_000_x |
| 16Mbyte | 32'hFF00_000_x |
| 8Mbyte | 32'hFF80_000_x |
| 4Mbyte | 32'hFFC0_000_x |
| 2Mbyte | 32'hFFE0_000_x |
| 1Mbyte | 32'hFFF0_000_x |
| 512kbyte | 32'hFFF8_000_x |
| 256Kbyte | 32'hFFFC_000_x |
| 128Kbyte | 32'hFFFE_000_x |
| 64Kbyte | 32'hFFFF_000_x |
| 32Kbyte | 32'hFFFF_800_x |
| 16Kbyte | 32'hFFFF_C00_x |
| 8Kbyte | 32'hFFFF_E00_x |
| 4Kbyte | 32'hFFFF_F00_x |
| 2Kbyte | 32'hFFFF_F80_x |
| 1Kbyte | 32'hFFFF_FC0_x |
| 512byte | 32'hFFFF_FE0_x |
| 256byte | 32'hFFFF_FF0_x |
| 128byte | 32'hFFFF_FF8_x |
| 64byte | 32'hFFFF_FFC_x |
| 32byte | 32'hFFFF_FFE_x |
| 16byte | 32'hFFFF_FFF_x |
| 8byte | 32'hFFFF_FFF_9 |
| 4byte | 32'hFFFF_FFF_D |
| unused | 32'h0000_000_2 |

Notes to the list above:
a. Only I/O BAR has 8-byte or 4-byte size.
b. For the value of "x" – please refer to the definition of Bit[3:0] as shown in the Memory BAR Configuration section of this document.
c. Not all BARs can be set for 2Gbytes. The total BAR space in one system should not be more than 4Gbytes for a 32-bit address.

# Local Bus Interface

## Target Operation

Initially, the local target is idle. A valid transaction in the PCI bus is indicated to the local bus side by the assertion of lt_accessn signal. At this time either the bar_hit, new_cap_hit or exprom_hit signal indicates whether a BAR or New Capabilities register is selected, and lt_command_out indicates the current PCI command. If the command is Special Cycle, then no BAR is selected, otherwise the selected BAR needs to prepare the next process.

For a Memory Read command, local target puts data on lt_ad_in and asserts lt_rdyn to indicate data on l_ad_in is valid. The core will read the data and assert lt_data_xfern after lt_rdyn is active. When the transaction is burst read, the core will continue to keep asserting lt_data_xfern at subsequent clocks and read data on l_ad_in if the local side does not insert wait cycle(s).

For a Memory Write command, local target asserts lt_rdyn to indicate that it is ready to receive data on l_data_out. The core will write data on l_data_out and assert lt_data_xfern to indicate valid data on l_data_out. Local target should read the data on l_data_out.

When the local target bus width is 64 bits, the signals lt_ldata_xfern and lt_hdata_xfern are used together instead of lt_data_xfern. For 32-bit data width, only lt_ldata_xfern is used. For 64-bit data width, lt_ldata_xfern and lt_hdata_xfern are used together. The signal lt_ldata_xfern applies to the lower 32-bit data, lt_hdata_xfern applies to the upper 32 bits of data.

A target transaction is ended when lt_accessn becomes inactive. At this time, bar_hit, new_cap_hit and exprom_hit are all deasserted.

When a 32-bit BAR is hit, only the following local bus signals are used:

- l_ad_in[31:0], l_data_out[31:0], lt_cben_out[3:0] and lt_ldata_xfern.

and the following signals are not used:

- l_ad_in[63:32], l_data_out[63:32], lt_cben_out[7:4] and lt_hdata_xfern.

## Master Operation

Local master starts a transaction request by asserting lm_req32n or lm_req64n when lm_status is in “Bus Termination” state. For a 32-bit transaction request, lm_req32n is asserted and lm_req64n is a “don't care”. For a 64-bit transaction, lm_req64n is asserted and lm_req32n is de-asserted. To minimize latency, the local master should issue the valid address, command and burst length on l_ad_in, lm_cben_in[3:0] and lm_burst_length respectively during the same clock cycle that lm_req32n or lm_req6n is asserted. Once PCI bus grants the bus, lm_gntn is asserted to indicate local master to continue with next process. Then local master works with lm_status. lm_req32n and lm_req64n should be deasserted right after lm_status is in “Address Loading” state, unless Fast Back-to-Back is intended. A normal transaction sequence of status starts from “Bus Termination” to “Address Loading” to “Bus Transaction” and ends with “Bus Termination”. During “Bus Transaction”, local master reads or writes data based on lm_data_xfern signal.

When the local master bus width is 64-bit, lm_ldata_xfern and lm_hdata_xfern are used instead of lm_data_xfern. For 32-bit data width BAR, only lm_ldata_xfern is used. For 64-bit data width BAR, lm_ldata_xfern and lm_hdata_xfern are used together. The signal lm_ldata_xfern applies to the lower 32 bits of data, lm_hdata_xfern applies to the upper 32 bits of data.

## Basic PCI Master Read and Write Transactions

Read and write transactions to memory and I/O space are used to transfer data on the PCI bus. The basic read and write transactions use the following PCI commands:

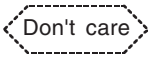
- I/O Read
- I/O Write
- Memory Read
- Memory Write
- Configuration Read
- Configuration Write

To simplify the integration of the PCI IP core, the basic master transactions are described based on different bus con-figurations supported with this PCI IP core. Although the fundamentals of the basic master transactions are the same, different bus configurations require slightly different local bus signaling. Refer to the following sections for more information on the basic bus master transactions with specific PCI IP core configurations:

- 32-bit PCI Master with a 32-Bit Local Bus
- 64 bit PCI Master with a 64-Bit Local Bus
- 32-bit PCI Master with a 64-Bit Local Bus

Refer to the advanced bus master transactions in the Advanced Master Transactions section for more information on properly handling wait state insertion and early termination of bus transactions by the PCI IP core.

**Waveform Legend**

| Symbol | Description |
|---|---|
|  | Driven signal |
|  | Driven bus signals or driven PCI parity signal (par and par64). |
|  | Floated PCI signals. If the signal is high, it is high maintained by system pull-up resistor. If the signal is in the middle of level place, it is tri-state. |
|  | Don't care local signal. For input signal, the core doesn't read it. For output signal, it is an invalid value. |
|  | 1. Local interface: Don't care bus signals. For input signals, the core doesn't read it. For output signal, they are invalid value. 2. PCI interface: the signal can be any value. |
|  | PCI signal turnaround. The core releases PCI bus control and changes output enable from ENABLE to DISABLE. |

Note: The clock number in the waveform is for the clock period, that is, after the current rising clock edge.

### 32-bit PCI Master with a 32-bit Local Bus

This section discusses read and write transactions executed by the PCI IP core operating as a master, configured with a 32-bit PCI bus and a 32-bit local bus. Because 32-bit I/O and memory transactions are alike, they are discussed together.

Figure 2-7 illustrates a basic 32-bit read transaction. Table 2-11 gives a clock-by-clock description of the basic 32-bit transaction shown in Figure 2-7. Understanding the latency between the PCI bus and the IP core's Local Master Interface is important for a read transaction. The clock number in the waveforms is for the clock period, that is, after the current rising clock edge.

***Figure 2-7. 32-bit Master Single Read Transaction with a 32-Bit Local Interface***

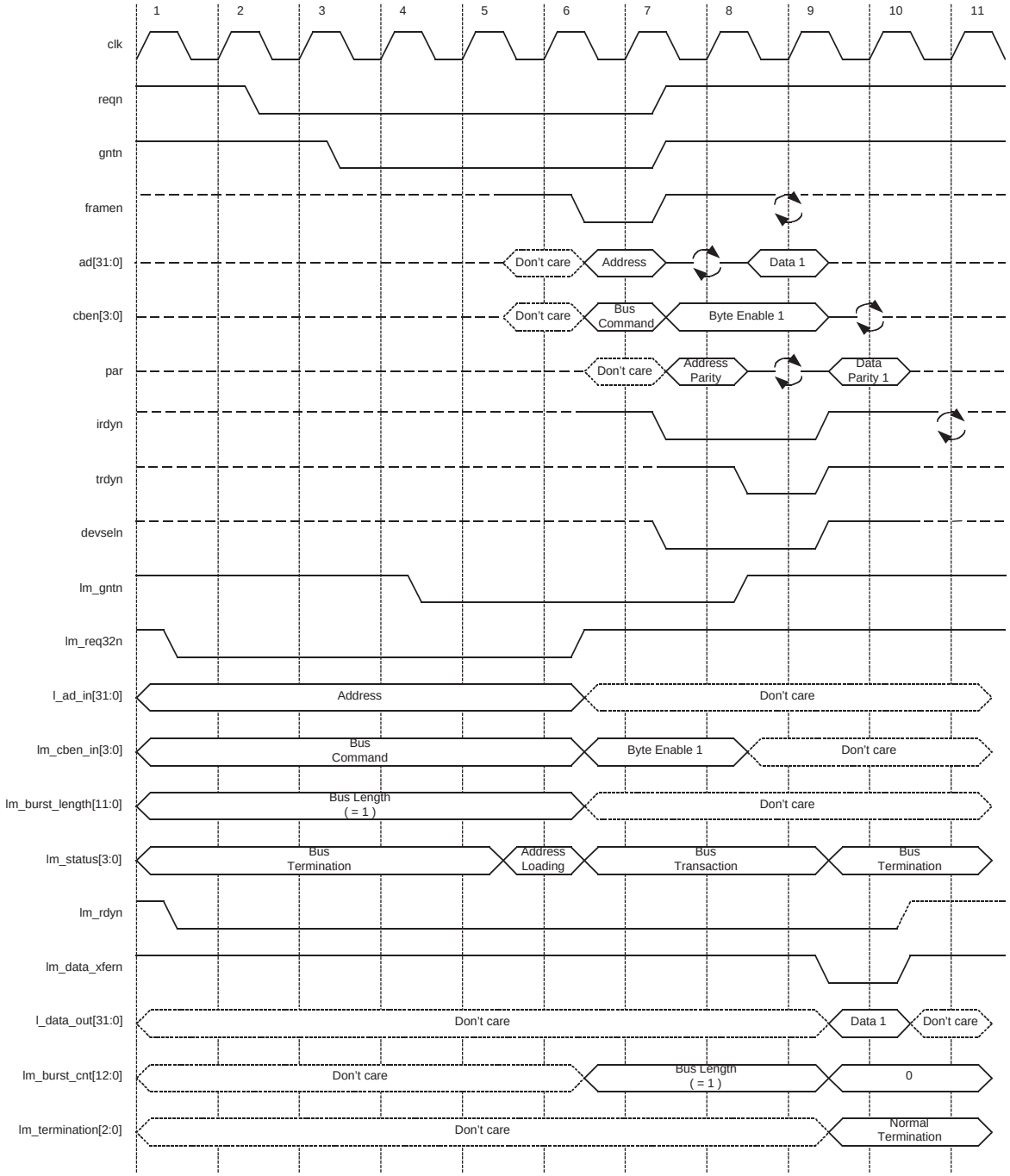


***Table 2-11. 32-bit Master Single Read Transaction with a 32-Bit Local Interface***

| CLK | Phase | Description |
|---|---|---|
| 1 | Idle | The lm_req32n signal is asserted by the master application logic on the Local Master interface for the 32-bit data transaction request. The Local Master interface drives the PCI starting address, the bus command, and the burst transaction length during the same clock cycle on l_ad_in, lm_cben_in and lm_burst_length, respectively. |
| 2 | Idle | The Core's Local Master Interface detects the asserted lm_req32n and asserts reqn to request the use of PCI bus. |
| 3 | Idle | gntn is asserted to grant the Core access to the PCI bus. Core is now the PCI master. |
| 4 | Idle | Since gntn is asserted and the current bus is idle, the Core starts the bus transactions. The Core asserts lm_gntn to inform the local master that the bus request is granted. |
| 5 | Idle | If both lm_req32n and gntn were asserted on the previous cycle, lm_status[3:0] is changed to 'Address Loading' to indicate the starting address, the bus command, and the burst length are being latched. |
| 6 | Address | The Core asserts framen to start transaction and the local master de-asserts lm_req32n when the previous lm_status[3:0] was 'Address Loading' and if it doesn't want to request fast back-to-back PCI bus transaction.<br>Since lm_status[3:0] was 'Address Loading' on the previous cycle. The Core drives the PCI starting address on ad[31:0] and the PCI command on cben[3:0]. On the same cycle, it outputs lm_status[3:0] as 'Bus Transaction' to indicate the beginning of the address/data phases.<br>lm_burst_cnt gets the value of the burst length.<br>Because lm_rdyn was asserted on the previous cycle and the next cycle is the first data phase, the local master provides the byte enables on lm_cben_in[3:0]. Asserting lm_rdyn also means the local master is ready to read data. If it is not ready to read data, it keeps lm_rdyn de-asserted until it is ready. |
| 7 | Turn around | The Core de-asserts reqn when framen was asserted and lm_req32n was de-asserted on the previous cycle.<br>The Core tri-states the ad[31:0] lines and drives the byte enables (Byte Enable 1). Since lm_rdyn was asserted on the previous cycle, it asserts irdyn to indicate it is ready to read data. Because this is a single data phase transaction, the master de-asserts framen simultaneously.<br>The target asserts devseln to claim the transaction. |
| 8 | Data 1 | The Core de-asserts lm_gntn to follow gntn.<br>The target asserts trdyn and puts Data 1 on ad[31:0].<br>If the local master is ready to read the first DWORD, lm_rdyn remains asserted.<br>Since both irdyn and trdyn are asserted, the first data phase is completed on this cycle. |
| 9 | Turn around | The Core relinquishes control of framen and cben. It de-asserts irdyn and changes the status of lm_status[3:0] into 'Bus Termination' with lm_termination as 'Normal Termination' because both trdyn and irdyn were asserted during the last cycle.<br>Since the previous data phase was completed, the Core transfers Data 1 on l_data_out[31:0] and decreases the lm_burst_cnt to zero.<br>The target relinquishes control of ad[31:0]. It de-asserts devseln and trdyn.<br>If both trdyn and lm_rdyn were asserted on the previous cycle, the Core asserts lm_data_xfern to the local master to signify Data 1 is available on l_data_out[31:0]. With lm_data_xfern asserted, the local master safely reads Data 1. |
| 10 | Idle | The master relinquishes control of irdyn and de-asserts lm_data_xfern, and the local master de-asserts lm_rdyn. |

Figure 2-8 illustrates a basic 32-bit write transaction. Table 2-12 gives a clock-by-clock description of the 32-bit write transaction.

***Figure 2-8. 32-bit Master Single Write Transaction with a 32-bit Local Interface***

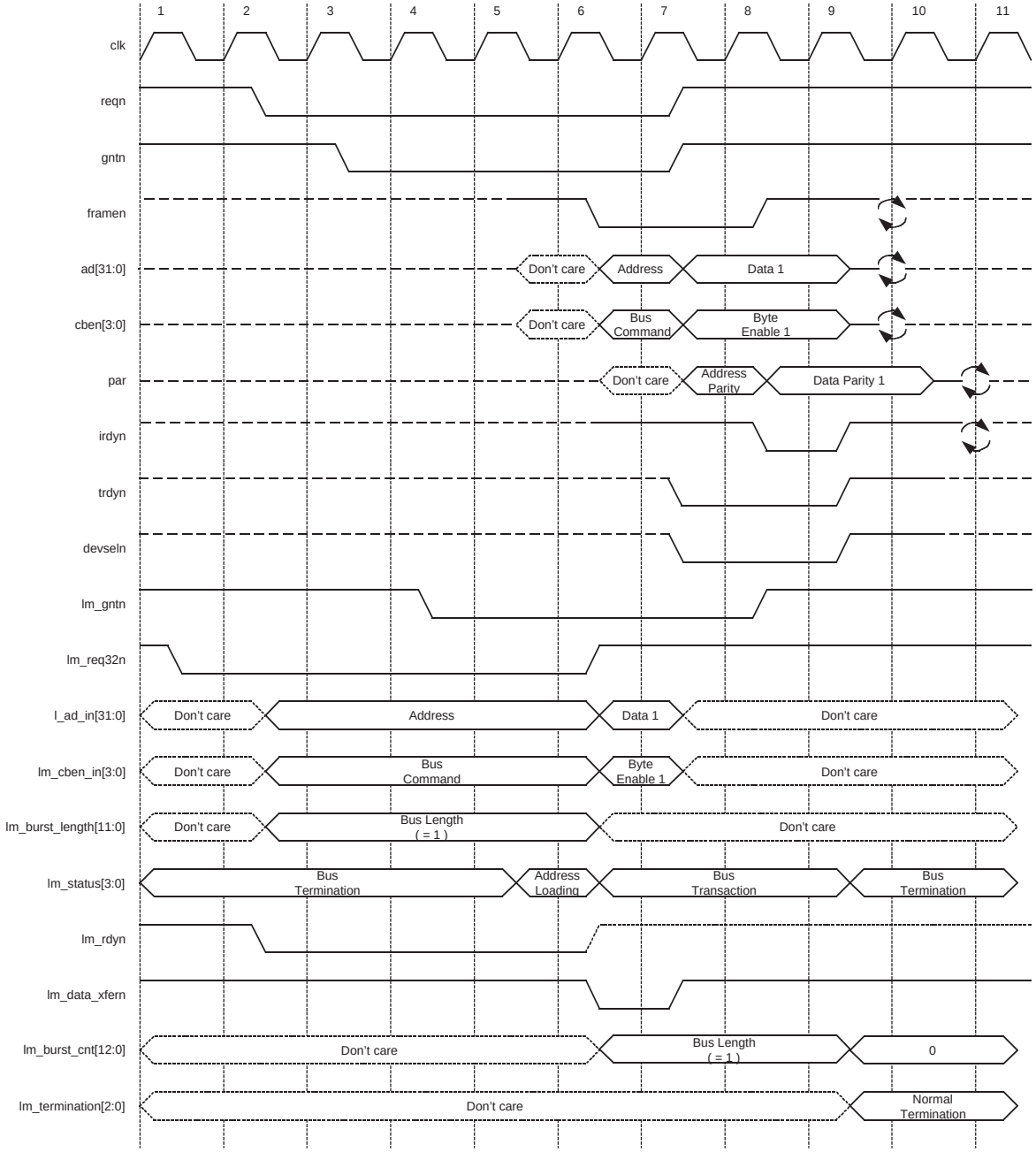


*Table 2-12. 32-bit Master Single Write Transaction with a 32-bit Local Interface*

| CLK | Phase | Description |
|---|---|---|
| 1 | Idle | The lm_req32n signal is asserted by the master application logic on the Local Master interface for the 32-bit data transaction request. The Local Master interface drives the PCI starting address, the bus command, and the burst transaction length during the same clock cycle on l_ad_in, lm_cben_in and lm_burst_length, respectively. |
| 2 | Idle | The Core's Local Master Interface detects the asserted lm_req32n and asserts reqn to request the use of PCI bus. |
| 3 | Idle | gntn is asserted to grant the Core access to the PCI bus. Core is now the PCI master. |
| 4 | Idle | Since gntn is asserted and the current bus is idle, the Core is going to start the bus transactions. The Core asserts lm_gntn to inform the local master that the bus request is granted. |
| 5 | Idle | If both lm_req32n and gntn were asserted on the previous cycle, lm_status[3:0] is changed to 'Address Loading' to indicate the starting address, the bus command and the burst length are being latched. |
| 6 | Address | The local master de-asserts lm_req32n when the previous lm_status[3:0] was 'Address Loading' and if it doesn't want to request another PCI bus transaction.<br>The Core asserts framen to initiate the 32-bit write transaction when gntn was asserted and lm_status[3:0] was 'Address Loading' on the previous cycle. It also drives the PCI starting address on ad[31:0] and the PCI command on cben[3:0]. On the same cycle, it outputs lm_status[3:0] as 'Bus Transaction' to indicate the beginning of the address/data phases.<br>lm_burst_cnt gets the value of the burst length.<br>Because lm_rdyn was asserted on the previous cycle and the next cycle is the first data phase, the local master provides Data 1 on l_ad_in[31:0] and the byte enables on lm_cben_in[3:0]. And the Core asserts lm_data_xfern to the local master to signify these data and byte enables are being read and will be transferred to the PCI bus.<br>Asserting lm_rdyn means the local master is ready to write data. If it is not, it keeps lm_rdyn de-asserted until it is ready. |
| 7 | Wait | If the target completes the fast decode and is ready to receive 32-bit data, it asserts devseln and trdyn.<br>The Core de-asserts reqn when framen was asserted and lm_req32n was de-asserted on the previous cycle.<br>With lm_data_xfern asserted on the previous cycle that was the address phase, the local master increments the address counter while the Core transfers Data 1 and the byte enables to ad[31:0] and cben[3:0]. |
| 8 | Data 1 | The Core de-asserts lm_gntn to follow gntn. Since the transaction only has one data, the Core asserts irdyn and de-asserts framen, Data 1 and the byte enables are kept on the PCI bus, the first data phase is completed. |
| 9 | Turn around | The Core relinquishes control of framen, ad and cben. It de-asserts irdyn, decreases lm_burst_cnt to zero and changes lm_status[3:0] into 'Bus Termination' with lm_termination as 'Normal Termination' because both trdyn and irdyn were asserted last cycle. The target de-asserts devseln, ackn and trdyn. |
| 10 | Idle | The Core relinquishes control of irdyn and par. |

## 64-Bit PCI Master with a 64-Bit Local Bus

This section discusses read and write transactions for a PCI IP core configured with a 64-bit PCI bus and a 64-bit local bus. The PCI Specification requires all 64-bit PCI master devices to execute both 64-bit and 32-bit transactions. The 32-bit transactions for the 32-bit Core, described in the previous section, are similar to the 32-bit transactions for the 64-bit PCI IP core configuration.

The 64-bit memory read transaction is similar to the 32-bit memory read transaction with the exception of additional PCI signals required for 64-bit signaling. Figure 2-9 and Table 2-13 illustrate a basic 64-bit read transaction.

***Figure 2-9. 64-bit Master Single Read Transaction with a 64-bit Local Interface***

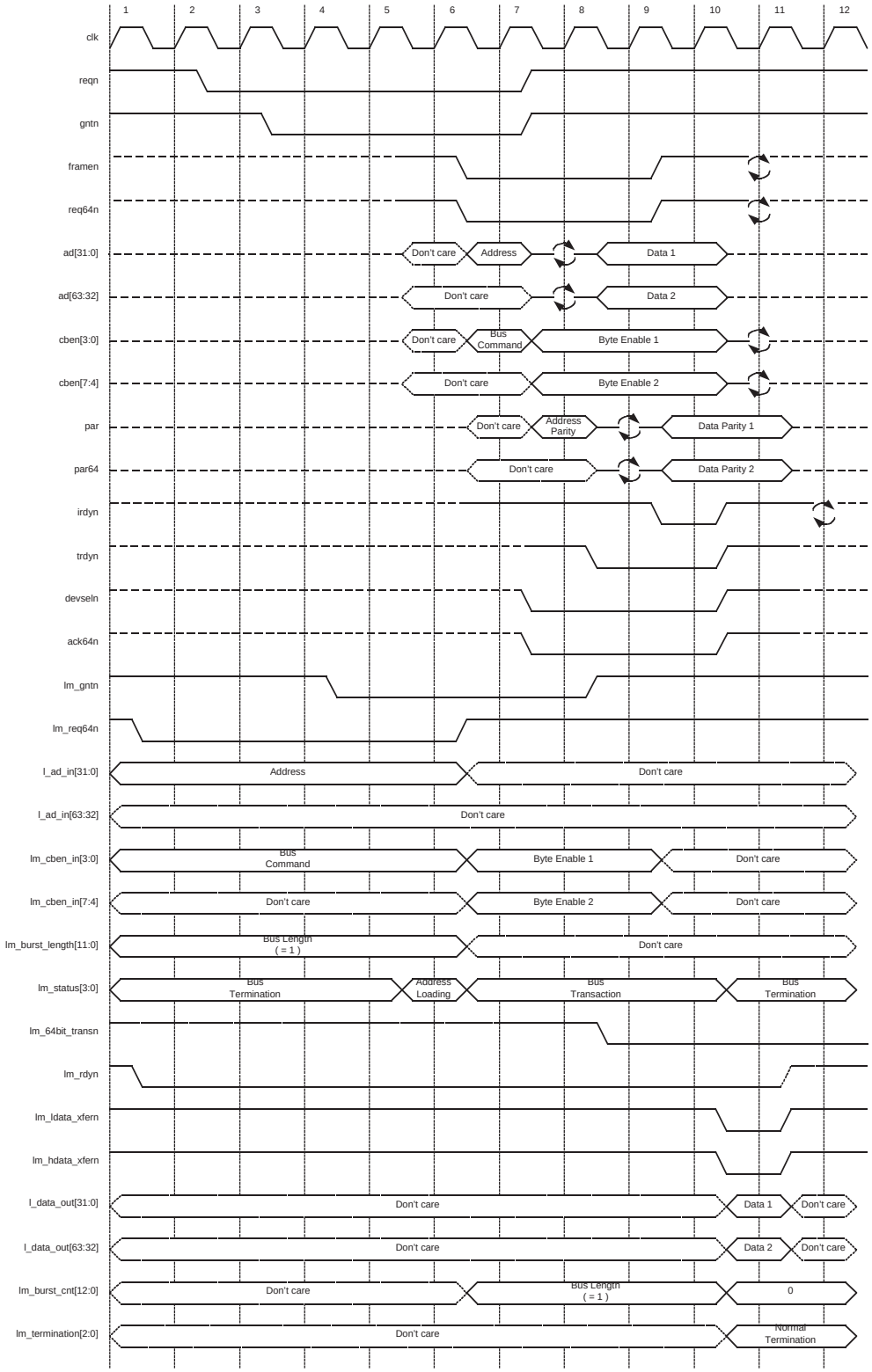


***Table 2-13. 64-bit Master Single Read Transaction with a 64-bit Local Interface***

| CLK | Phase | Description |
|---|---|---|
| 1 | Idle | The local master asserts lm_req64n for the 64-bit data transaction request. It also puts the PCI starting address, the bus command, and the burst length during the same clock cycle on l_ad_in, lm_cben_in, and lm_burst_length, respectively. |
| 2 | Idle | The Core's Local Master Interface detects the asserted lm_req64n and asserts reqn to request the use of PCI bus. |
| 3 | Idle | gntn is asserted to grant the Core access to the PCI bus. Core is now the PCI master. |
| 4 | Idle | Since gntn is asserted and the current bus is idle, the Core is going to start the bus transactions. The master asserts lm_gntn to inform the local master that the bus request is granted. |
| 5 | Idle | If both lm_req64n and lm_gntn are asserted on the previous cycle, lm_status[3:0] is changed to 'Address Loading' to indicate the starting address, the bus command and the burst length are latched. |
| 6 | Address | The local master de-asserts lm_req64n when the previous lm_status[3:0] was 'Address Loading' and if it doesn't want to request another PCI bus transaction.<br>The Core asserts framen and req64n to initiate the 64-bit read transaction when gntn was still asserted and lm_status[3:0] was 'Address Loading' on the previous cycle. It also drives the PCI starting address on ad[31:0] and the PCI command on cben[3:0]. On the same cycle, it outputs lm_status[3:0] as 'Bus Transaction' to indicate the beginning of the address/data phases.<br>lm_burst_cnt gets the value of the burst length.<br>Because lm_rdyn was asserted on the previous cycle and the next cycle is the first data phase, the local master provides the byte enables on lm_cben_in[7:0]. Asserting lm_rdyn also means the local master is ready to read data for the single data transaction. If it is not ready to read data, it keeps lm_rdyn de-asserted until it is ready. |
| 7 | Turn around | The Core de-asserts reqn when framen was asserted but lm_req64n was de-asserted on the previous cycle. The target asserts devseln and ack64n to indicate it acknowledges the 64-bit transaction. The Core tri-states the ad[63:0] lines and drives the byte enables (Byte Enable 1 and 2). |
| 8 | Wait | The Core asserts lm_64bit_transn to indicate the current data transaction is 64 bits wide. It de-asserts lm_gntn to follow gntn.<br>The target asserts trdyn and puts Data 1 and 2 on ad[63:0].<br>If the local master is ready to read the 64-bit word (QWORD), it keeps lm_rdyn asserted. |
| 9 | Data 1 and 2 | Since lm_rdyn was asserted on the previous cycle, it asserts irdyn to indicate it is ready to read data. Since both irdyn and trdyn are asserted, the first data phase is completed on this cycle. |
| 10 | Turn around | Since the previous data phase was completed, the Core transfers Data 1 and 2 on l_data_out[63:0] and decreases the lm_burst_cnt to zero.<br>The Core relinquishes control of framen, req64n and cben. It de-asserts irdyn and changes lm_status[3:0] into 'Bus Termination' with lm_termination as 'Normal Termination' because both trdyn and irdyn were asserted last cycle.<br>The target relinquishes control of ad[63:0]. It de-asserts devseln, ack64n and trdyn.<br>If both trdyn and lm_rdyn were asserted on the previous cycle, the Core asserts lm_ldata_xfern and lm_hdata_xfern to the local master to signify Data 1 and 2 are available on l_data_out[63:0]. With lm_ldata_xfern and lm_hdata_xfern asserted, the local master safely reads Data 1 and 2. |
| 11 | Idle | The Core relinquishes control of irdyn and de-asserts lm_ldata_xfern and lm_hdata_xfern, and the local master de-asserts lm_rdyn since all of the burst data have been read. |

The 64-bit memory write transaction is similar to the 32-bit target write transaction with additional PCI signals required for 64-bit signaling. Figure 2-10 and Table 2-14 show a basic 64-bit write transaction.

***Figure 2-10. 64-bit Master Single Write Transaction with a 64-bit Local Interface***

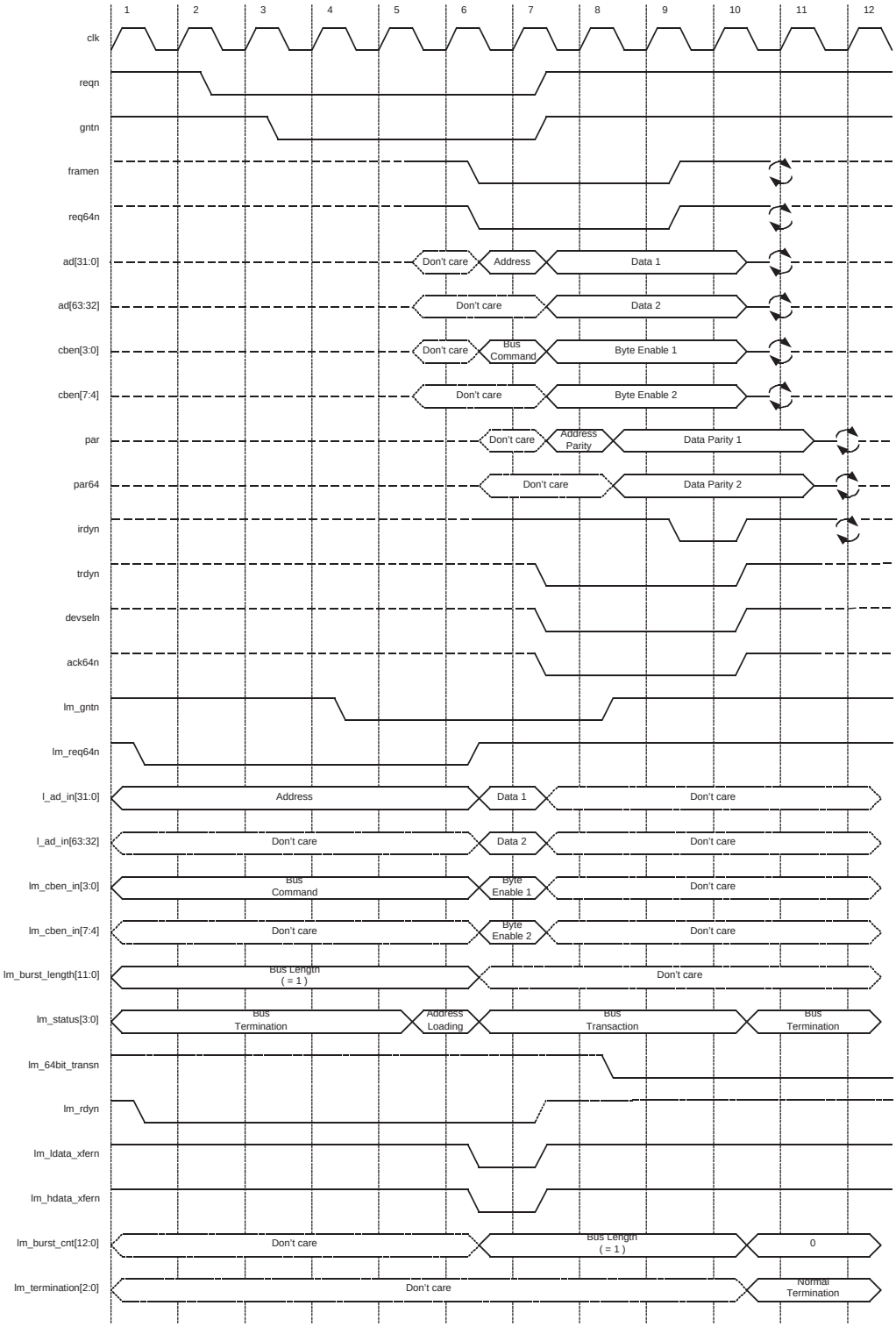


***Table 2-14. 64-bit Master Single Write Transaction with a 64-bit Local Interface***

| CLK | Phase | Description |
|---|---|---|
| 1 | Idle | The local master asserts lm_req64n for the 64-bit data transaction request. It also issues the PCI starting address, the bus command and the burst length on l_ad_in, lm_cben_in and lm_burst_length respectively on the same clock cycle. |
| 2 | Idle | The Core's Local Master Interface detects the asserted lm_req64n and asserts reqn to request the use of PCI bus. |
| 3 | Idle | gntn is asserted to grant the Core access to the PCI bus. Core is now the PCI master. |
| 4 | Idle | Since gntn is asserted and the current bus is idle, the Core is going to start the bus transactions. The Core asserts lm_gntn to inform the local master that the bus request is granted. |
| 5 | Idle | If both lm_req64n and lm_gntn were asserted on the previous cycle, lm_status[3:0] is changed to 'Address Loading' to indicate the starting address, the bus command and the burst length are being latched. |
| 6 | Address | The local master de-asserts lm_req64n when the previous lm_status[3:0] was 'Address Loading' and if it doesn't want to request another PCI bus transaction.<br>The Core asserts framen and req64n to initiate the 64-bit write transaction when gntn was asserted and lm_status[3:0] was 'Address Loading' on the previous cycle. It also drives the PCI starting address on ad[31:0] and the PCI command on cben[3:0]. On the same cycle, it outputs lm_status[3:0] as 'Bus Transaction' to indicate the beginning of the address/data phases.<br>lm_burst_cnt gets the value of the burst length.<br>Because lm_rdyn was asserted on the previous cycle and the next cycle is the first data phase, the local master provides Data 1 and Data 2 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0]. And the Core asserts lm_ldata_xfern and lm_hdata_xfern to the local master to signify these data and byte enables are being read and will be transferred to the PCI bus.<br>Asserting lm_rdyn means the local master is ready to write data. If it is not, it keeps lm_rdyn de-asserted until it is ready. |
| 7 | Wait | The Core de-asserts reqn when framen was asserted and lm_req64n was de-asserted on the previous cycle.<br>If the target completes the fast decode and is ready to receive 64-bit data, it asserts devseln, ack64n and trdyn.<br>With lm_ldata_xfern and lm_hdata_xfern asserted on the previous cycle that was the address phase, the local master increments the address counter while the Core transfers Data 1 and Data 2 and their byte enables to ad[63:0] and cben[7:0].<br>Because this is the first write data phase and devseln is just asserted, the master keeps framen asserted and irdyn de-asserted to judge 64-bit or 32-bit transaction. |
| 8 | Wait | Since ack64n and trdyn are asserted on the previous cycle. The Core asserts lm_64bit_transn to indicate the current data transaction is 64-bits wide. It de-asserts lm_gntn to follow gntn. |
| 9 | Data 1 and 2 | With lm_rdyn asserted in the previous cycle, the Core asserts irdyn and deasserts framen for a single cycle transaction. Since both irdyn and trdyn are asserted, the data phase is completed on the cycle. |
| 10 | Turn around | The Core relinquishes control of framen, req64n, ad and cben. It de-asserts irdyn, decreases 'lm_burst_cnt' to zero and changes lm_status[3:0] into 'Bus Termination' with lm_termination as 'Normal Termination' because both trdyn and irdyn were asserted last cycle. The target de-asserts devseln, ack64n and trdyn. |
| 11 | Idle | The Core relinquishes control of irdyn, par and par64. |

## 32-bit PCI Master with a 64-Bit Local Bus

This section discusses read and write transactions executed by the PCI IP core configured with a 32-bit PCI bus and a 64-bit local bus. The 32-bit PCI master transactions, described in the 32-Bit PCI Master and 32-Bit Local Bus section, are similar to these master transactions; however; the data is handled differently at the Local Master Interface. Two 32-bit PCI data phases are required to transfer 64 bits of data to the Local Master Interface.

The Local Master Interface control latches the complete QWORD and routes the proper DWORD to the PCI data bus. The `lm_ldata_xfern` and `lm_hdata_xfern` signals specify which DWORD is transferred.

If the starting address is aligned with QWORD, the first DWORD is assumed to be the lower DWORD of a QWORD and is placed on the PCI data bus. Otherwise, the upper DWORD is placed on the PCI data bus.

The 64-bit memory read transaction is similar to the 32-bit target read transaction with additional PCI signals required for 64-bit signaling. Figure 2-11 and Table 2-15 illustrate a basic 64-bit read transaction.

***Figure 2-11. 32-bit Master Single Read Transaction with a 64-bit Local Interface***

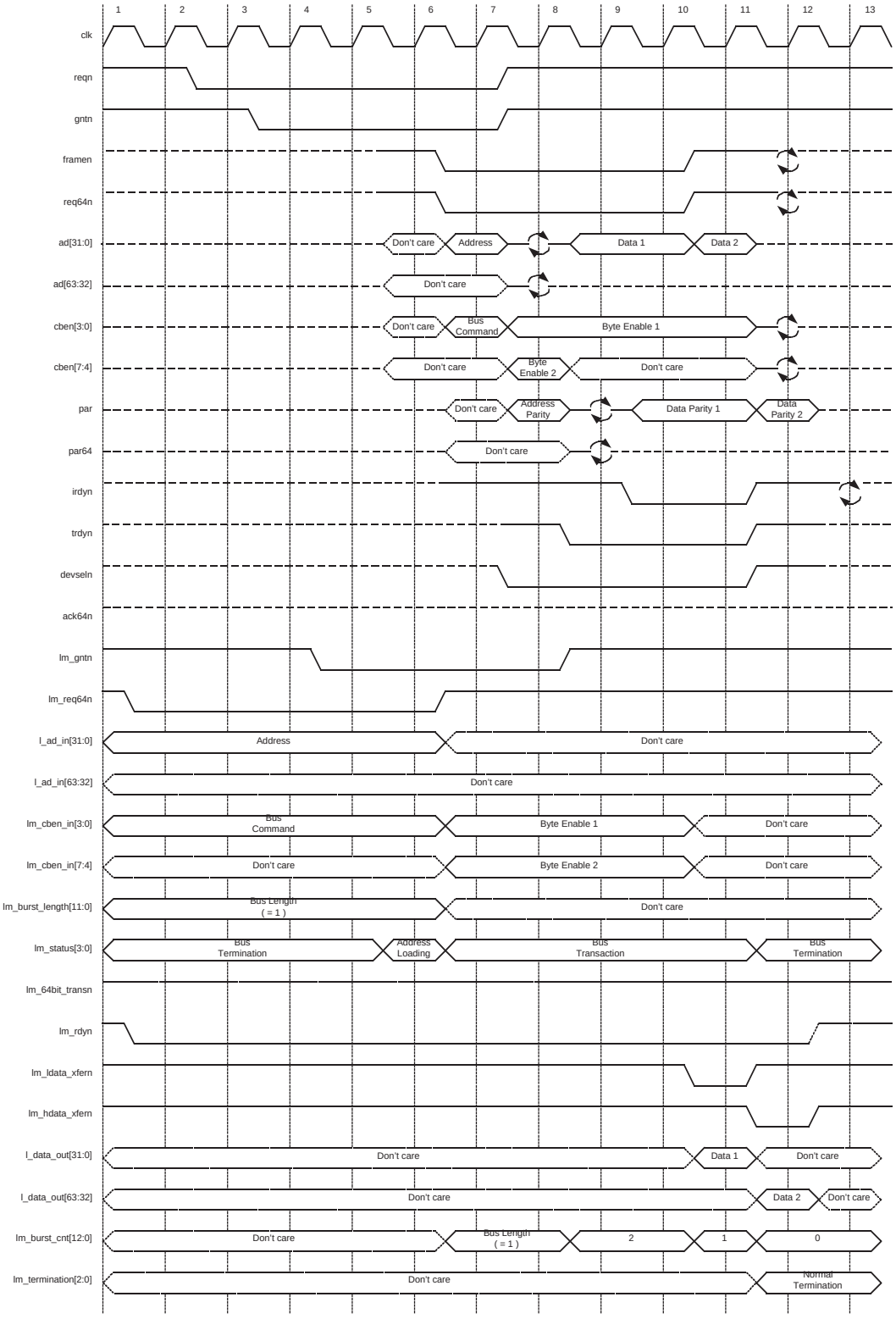


***Table 2-15. 32-bit Master Single Read Transaction with a 64-bit Local Interface***

| CLK | Phase | Description |
|---|---|---|
| 1 | Idle | The local master asserts lm_req64n for the 64-bit data transaction request. It also issues the PCI starting address, the bus command and the burst length on l_ad_in, lm_cben_in and lm_burst_length respectively on the same clock cycle. |
| 2 | Idle | The Core's Local Master Interface detects the asserted lm_req64n and asserts reqn to request the use of PCI bus. |
| 3 | Idle | gntn is asserted to grant the Core access to the PCI bus. Core is now the PCI master. |
| 4 | Idle | Since gntn is asserted and the current bus is idle, the Core is going to start the bus transactions. The Core asserts lm_gntn to inform the local master that the bus request is granted. |
| 5 | Idle | If both lm_req64n and lm_gntn were asserted on the previous cycle, lm_status[3:0] is changed to 'Address Loading' to indicate the starting address, the bus command and the burst length are being latched. |
| 6 | Address | The local master de-asserts lm_req64n when the previous lm_status[3:0] was 'Address Loading' and if it doesn't want to request another PCI bus transaction.<br>The Core asserts framen and req64n initiate the 64-bit read transaction when gntn was still asserted and lm_status[3:0] was 'Address Loading' on the previous cycle. It also drives the PCI starting address on ad[31:0] and the PCI command on cben[3:0]. On the same cycle, it outputs lm_status[3:0] as 'Bus Transaction' to indicate the beginning of the address/data phases.<br>lm_burst_cnt gets the value of the burst length.<br>Because lm_rdyn was asserted on the previous cycle and the next cycle is the first data phase, the local master provides the byte enables on lm_cben_in[7:0]. Asserting lm_rdyn also means the local master is ready to read data. If it is not ready to read data, it keeps lm_rdyn de-asserted until it is ready. |
| 7 | Turn around | The Core de-asserts reqn when framen was asserted but lm_req64n was de-asserted on the previous cycle.<br>The Core tri-states the ad[63:0] lines and drives the byte enables (Byte Enable 1 and 2). Since lm_rdyn was asserted on the previous cycle, it asserts irdyn to indicate it is ready to read data.<br>The target asserts devseln. Since the target is 32 bits, it doesn't assert ack64n as devseln. |
| 8 | Wait | The Core de-asserts lm_gntn to follow gntn. The target asserts trdyn and puts Data 1 ad[31:0].<br>Since the Core detects the PCI bus transaction is 32 bits, it de-asserts lm_64bit_transn and changes lm_burst_cnt to two. The transaction is changed from single cycle to burst cycle.<br>If the local master is ready to read the first DWORD, it keeps lm_rdyn asserted. |
| 9 | Data 1 | The Core asserts irdyn and keeps framen asserted to signify the burst continues. Since both irdyn and trdyn are asserted, the first data phase is completed on this cycle. |
| 10 | Data 2 | Since the previous data phase was completed, the Core transfers Data 1 on l_data_out[31:0] and decreases the lm_burst_cnt to one.<br>If both trdyn and lm_rdyn were asserted on the previous cycle, the Core asserts lm_ldata_xfern to the local master to signify Data is available on l_data_out[31:0]. With lm_ldata_xfern asserted, the local master can safely read Data 1 and increment the address counter.<br>If the local master keeps lm_rdyn asserted on the previous cycle, the Core keeps irdyn asserted.<br>If the target is still ready to provide data, it keeps trdyn asserted and drives the next DWORD (Data 2) on ad[31:0].<br>If the local master is ready to read the next DWORD, it keeps lm_rdyn asserted.<br>The Core keeps irdyn and de-asserts framen for the last data cycle.<br>Since both irdyn and trdyn are asserted, the second data phase is completed on this cycle. |

*Table 2-15. 32-bit Master Single Read Transaction with a 64-bit Local Interface (Continued)*

| CLK | Phase | Description |
|---|---|---|
| 11 | Turn around | Since the previous data phase was completed, the Core transfers Data 2 on l_data_out[63:32] and decreases the lm_burst_cnt to zero.<br>The Core relinquishes control of framen, req64n and cben. It de-asserts irdyn and changes lm_status[3:0] into 'Bus Termination' with lm_termination as 'Normal Termination' because both trdyn and irdyn were asserted last cycle.<br>The target relinquishes control of ad[31:0]. It de-asserts devseln and trdyn.<br>If both trdyn and lm_rdyn were asserted on the previous cycle, the Core asserts lm_hdata_xfern to the local master to signify Data 2 is available on l_data_out[63:32]. With lm_hdata_xfern asserted, the local master can safely read Data 2. |
| 12 | Idle | The Core relinquishes control of irdyn and de-asserts lm_hdata_xfern, and the local master de-asserts lm_rdyn since all of the burst data have been read. |

The 64-bit memory write transaction is similar to the 32-bit write transaction with additional PCI signals required for 64-bit signaling. Figure 2-12 and Table 2-16 show a basic 64-bit write transaction.

***Figure 2-12. 32-bit Master Single Write Transaction with a 64-bit Local Interface***

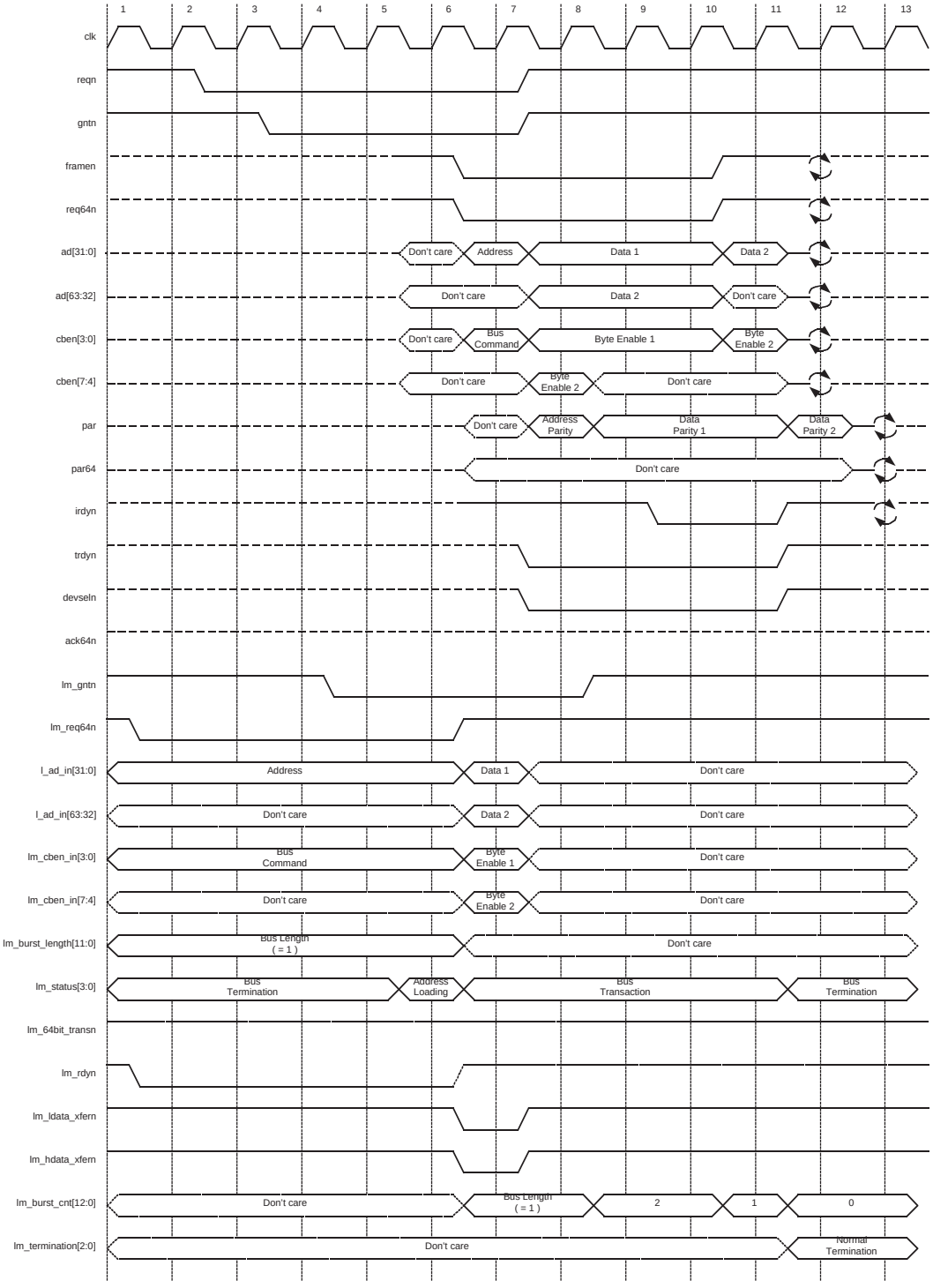


*Table 2-16. 32-bit Master Single Write Transaction with a 64-bit Local Interface*

| CLK | Phase | Description |
|---|---|---|
| 1 | Idle | The local master asserts lm_req64n for the master 64-bit data transaction request. It also issues the PCI starting address, the bus command and the burst length on l_ad_in, lm_cben_in and lm_burst_length respectively on the same clock cycle. |
| 2 | Idle | The Core's Local Master Interface detects the asserted lm_req64n and asserts reqn to request the use of PCI bus. |
| 3 | Idle | gntn is asserted to grant the Core access to the PCI bus. Core is now the PCI master. |
| 4 | Idle | Since gntn is asserted and the current bus is idle, the Core is going to start the bus transactions. The Core asserts lm_gntn to inform the local master that the bus request is granted. |
| 5 | Idle | If both lm_req64n and gntn were asserted on the previous cycle, lm_status[3:0] is changed to 'Address Loading' to indicate the starting address, the bus command and the burst length are being latched. |
| 6 | Address | The local master de-asserts lm_req64n when the previous lm_status[3:0] was 'Address Loading' and if it doesn't want to request another PCI bus transaction.<br>The Core asserts framen and req64n to initiate the 64-bit write transaction when gntn was asserted and lm_status[3:0] was 'Address Loading' on the previous cycle. It also drives the PCI starting address on ad[31:0] and the PCI command on cben[3:0]. On the same cycle, it outputs lm_status[3:0] as 'Bus Transaction' to indicate the beginning of the address/data phases.<br>lm_burst_cnt gets the value of the burst length.<br>Because lm_rdyn was asserted on the previous cycle and the next cycle is the first data phase, the local master should provide Data 1 and Data 2 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0]. The Core asserts lm_ldata_xfern and lm_hdata_xfern to the local master to signify these data and byte enables are being read and will be transferred to the PCI bus.<br>Asserting lm_rdyn means the local master is ready to write data. If it is not, it should keep lm_rdyn de-asserted until it is ready. |
| 7 | Wait | If the target completes the fast decode and is ready to receive 32-bit data, it asserts devseln and trdyn. But it doesn't assert ack64n.<br>The Core de-asserts reqn when framen was asserted and lm_req64n was de-asserted on the previous cycle.<br>With lm_data_xfern and lm_hdata_xfern asserted on the previous cycle that was the address phase, the Core transfers Data 1, Data 2 and the byte enables to ad[63:0] and cben[7:0]. |
| 8 | Wait | The Core de-asserts lm_gntn to follow gntn. Since the Core detects the PCI bus transaction width is 32 bits. It de-asserts lm_64bit_transn and changes lm_burst_cnt to two. The transaction is changed from a single cycle to a burst cycle. |
| 9 | Data 1 | With both devseln and lm_rdyn asserted previous cycle, the Core asserts irdyn, and it prepares for the 32-bit write burst.<br>Because the Core performs the burst transactions, it keeps framen asserted.<br>Since both irdyn and trdyn are asserted, the first data phase is completed on this cycle. |
| 10 | Data 2 | Since the previous data phase was completed, the Core decreases lm_burst_cnt.<br>Since Data 1 PCI bus was read by the target, the Core transfers Data 2 and the byte enables to ad[31:0] and cben[3:0].<br>With lm_rdyn asserted previous cycle, the Core keeps irdyn asserted. The Core de-asserts framen for the last data cycle.<br>Since both irdyn and trdyn are asserted, the second data phase is completed on this cycle. |
| 11 | Turn around | The Core relinquishes control of framen, req64n, ad and cben. It de-asserts irdyn, decreases lm_burst_cnt to zero and changes lm_status[3:0] into 'Bus Termination' with lm_termination as 'Normal Termination' because both trdyn and irdyn were asserted last cycle. The target de-asserts devseln and trdyn. |
| 10 | Idle | The Core relinquishes control of irdyn par and par64. |

### Configuration Read and Write Transactions

When operating as a PCI master, the PCI IP core supports Configuration cycles to CSR addresses 00h to FFh. The Local Master Interface has full control of these types of accesses, which are similar to the memory transactions described in earlier sections. The PCI IP core only supports 32-bit, single data phase transactions to the configuration registers.

During a configuration access, the PCI master drives an address/data pin that is connected to the `idsel` signal for all of the PCI target devices. For more information on the binding of the address/data signals to the `idsel` signal, refer to the *PCI Local Bus Specification, Revision 3.0*.

### PCI Master I/O Read and Write Transactions

The PCI IP core's application executes I/O space transactions. Transactions to I/O address space are similar to the basic memory transactions discussed in the Basic PCI Master Read and Write Transactions section.

By definition, read and write transactions to I/O space are executed using 32-bit PCI transactions only. Driving all 32 bits of the address and byte enables (`cben[3:0]`) is required.

## Advanced Master Transactions

Most PCI applications require more than basic read and write transactions. For these applications, the PCI IP core offers advanced features to handle the more difficult aspects of the PCI bus. The advanced features are used to provide the PCI application with more flexibility and improve the overall PCI system performance.

### Wait States

Care must be taken when processing wait states to be compliant with the *PCI Local Bus Specification, Revision 3.0*. Once a PCI master or a PCI target signals that it is ready to send or receive data, it must complete the current PCI data phase. For example, if the PCI IP core, as a target, is ready to write data and the PCI master inserts wait states, the PCI IP core must wait to write the data until the master is ready again. Additionally, if the PCI IP core asserts `trdyn` for a data phase, it cannot insert any wait states until the next data phase. Coincident master and target wait state insertion is also a possibility. Refer to the PCI Specification for more information regarding coincident wait state insertion.

Two types of wait states occur on the PCI bus: master wait state insertion and target wait state insertion. When the PCI master inserts wait states, the PCI IP core must hold off data until the PCI master is ready to complete the data phase. The PCI IP core inserts the second type of wait states. The back-end application controls the PCI IP core's wait state insertion via the Local Master Interface.

Figure 2-13 and Table 2-17 illustrate master-inserted and target-inserted wait states for read transactions. The figure illustrates the correlation between the PCI Interface and the Local Master Interface. The table gives a clock-by-clock description of each event in the figure.

***Figure 2-13. 32-bit Master Read Transaction with Local Wait State***

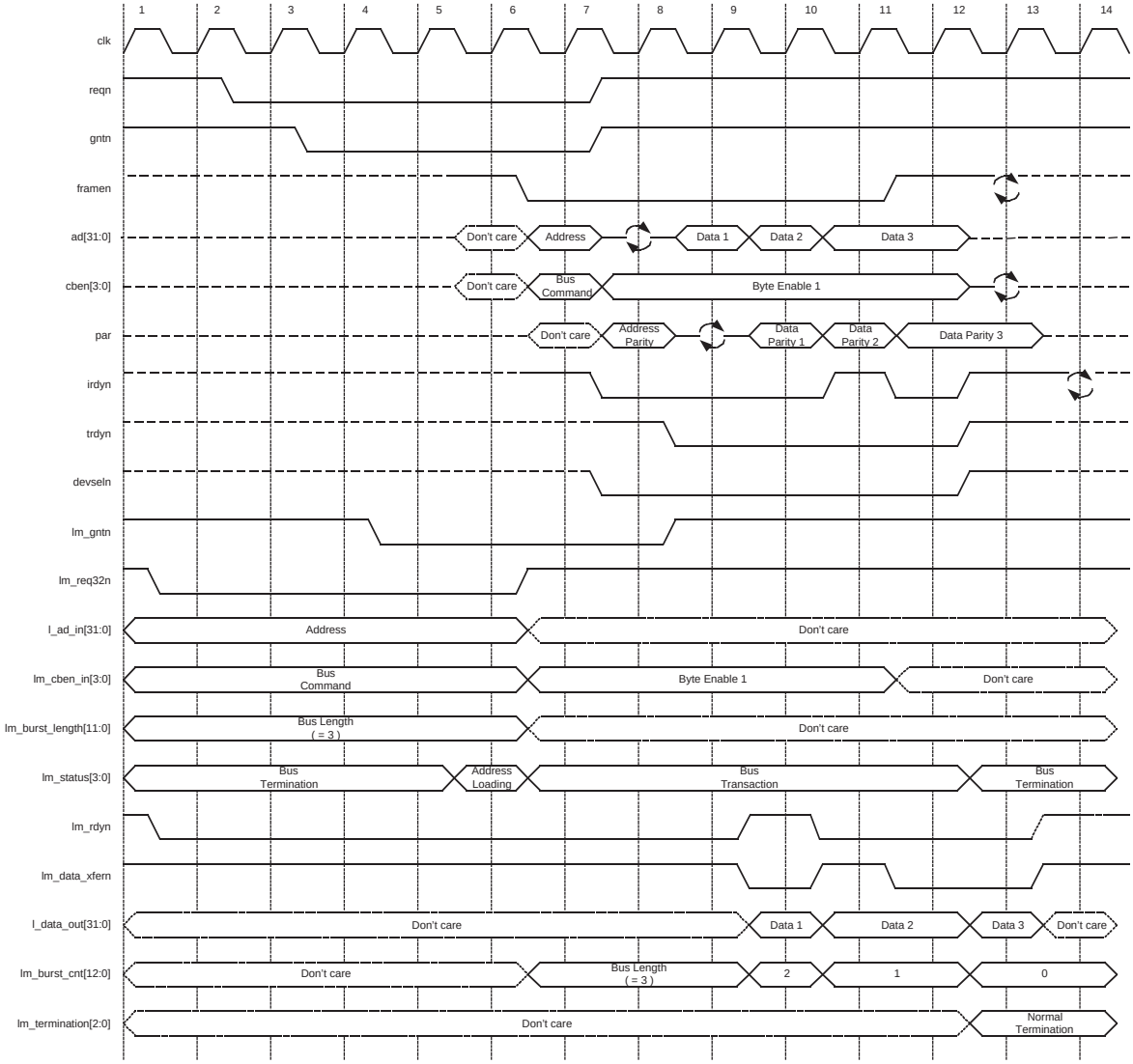



***Table 2-17. 32-bit Master Read Transaction with Local Wait State***

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Idle | The `lm_req32n` signal is asserted on the Local Master interface, by the local master to request a 32-bit data transaction. The local master issues the PCI starting address, the bus command and the burst length during the same clock cycle, to `l_ad_in`, `lm_cben_in`, and `lm_burst_length`, respectively. |
| 2 | Idle | The master detects the asserted `lm_req32n` and asserts `reqn` to request the use of PCI bus. |
| 3 | Idle | `gntn` is asserted to grant the Core access to the PCI bus. Core is now the PCI master. |
| 4 | Idle | Since `gntn` is asserted and the current bus is idle, the Core starts the bus transactions. The Core asserts `lm_gntn` to inform the local master that the bus request is granted. |
| 5 | Idle | If both `lm_req32n` and `gntn` were asserted on the previous cycle, `lm_status[3:0]` is changed to 'Address Loading' to indicate the starting address, the bus command, and the burst length are being latched. |

*Table 2-17. 32-bit Master Read Transaction with Local Wait State (Continued)*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 6 | Address | The Core asserts framen to start transaction and the local master de-asserts lm_req32n when the previous lm_status[3:0] was 'Address Loading' and if it doesn't want to request another PCI bus transaction.<br>lm_status[3:0] was 'Address Loading' on the previous cycle. It also drives the PCI starting address on ad[31:0] and the PCI command on cben[3:0]. On the same cycle, it outputs lm_status[3:0] as 'Bus Transaction' to indicate the beginning of the address/data phases.<br>lm_burst_cnt gets the value of the burst length.<br>Because lm_rdyn was asserted on the previous cycle and the next cycle is the first data phase, the local master provides the byte enables on lm_cben_in[3:0]. Asserting lm_rdyn also means the local master is ready to read data. If it is not ready to read data, it keep lm_rdyn de-asserted until it is ready. |
| 7 | Turn around | The Core de-asserts reqn when framen was asserted and lm_req32n was de-asserted on the previous cycle.<br>The Core tri-states the ad[31:0] lines and drives the byte enables (Byte Enable 1). Since lm_rdyn was asserted on the previous cycle, it asserts irdyn to indicate it is ready to read data. Because this is not the last cycle transaction, the Core keeps framen.<br>The target asserts devseln to response the command. |
| 8 | Data 1 | The Core de-asserts lm_gntn to follow gntn.<br>With the trdyn asserted, Data 1 is driven on to ad[31:0]. If the PCI IP core is ready to receive data, irdyn remains asserted and it keeps the Byte Enables on cben[3:0].<br>The Local Master interface is ready to receive data, so it keeps lm_rdyn. |
| 9 | Data 2 | If the PCI IP core is ready to receive data, it asserts irdyn and keeps the byte enables on cben[3:0]. The target device drives Data 2 on the PCI bus.<br>Since the previous data phase was completed, the master transfers Data 1 on l_data_out[31:0] and decreases lm_burst_cnt to two. The Core asserts lm_data_xfern if lm_rdyn was asserted on the previous cycle.<br>The local master interface is not ready to receive next data, so it de-asserts lm_rdyn. |
| 10 | Master Wait | Since lm_rdyn was de-asserted on the previous cycle, the Core de-asserts irdyn to signify the Core is inserting a wait state. The target device drives Data 3 on the PCI bus.<br>Since the previous data phase was completed, the Core transfers Data 2 on l_data_out[31:0] and decreases lm_burst_cnt to one. The Core de-asserts lm_data_xfern if lm_rdyn wasn't asserted on the previous cycle.<br>The local master asserts lm_rdyn for being ready to receive data. |
| 11 | Data 3 | The Core asserts irdyn and de-asserts framen for the last data phase. The target keeps devseln, trdyn and Data 3 on PCI bus. The Core asserts lm_data_xfern if lm_rdyn was asserted on the previous cycle. The local master asserts lm_rdyn for being ready to receive data. |
| 12 | Turn around | The Core de-asserts idyn, the target de-asserts both devseln and trdyn. The Core relinquishes control of framen, ad[31:0], and cben[3:0]. Since the previous data phase was completed, the Core transfers Data 1 on l_data_out[31:0] and decreases lm_burst_cnt to zero. The Core asserts lm_data_xfern if lm_rdyn was asserted on the previous cycle.<br>The Core changes lm_status[3:0] into the 'Bus Termination' state with lm_termination as 'Normal Termination' because both trdyn and irdyn were asserted last cycle. |
| 13 | Idle | The Core relinquishes control of irdyn. |

Figure 2-14 and Table 2-18 show master-inserted and target-inserted wait states occurring on write transactions. The figure illustrates the correlation of the PCI interface to the Local Master Interface. The table gives a clock-by-clock description of each event in the figure.

***Figure 2-14. 32-bit Master Write Transaction with Local Wait State***

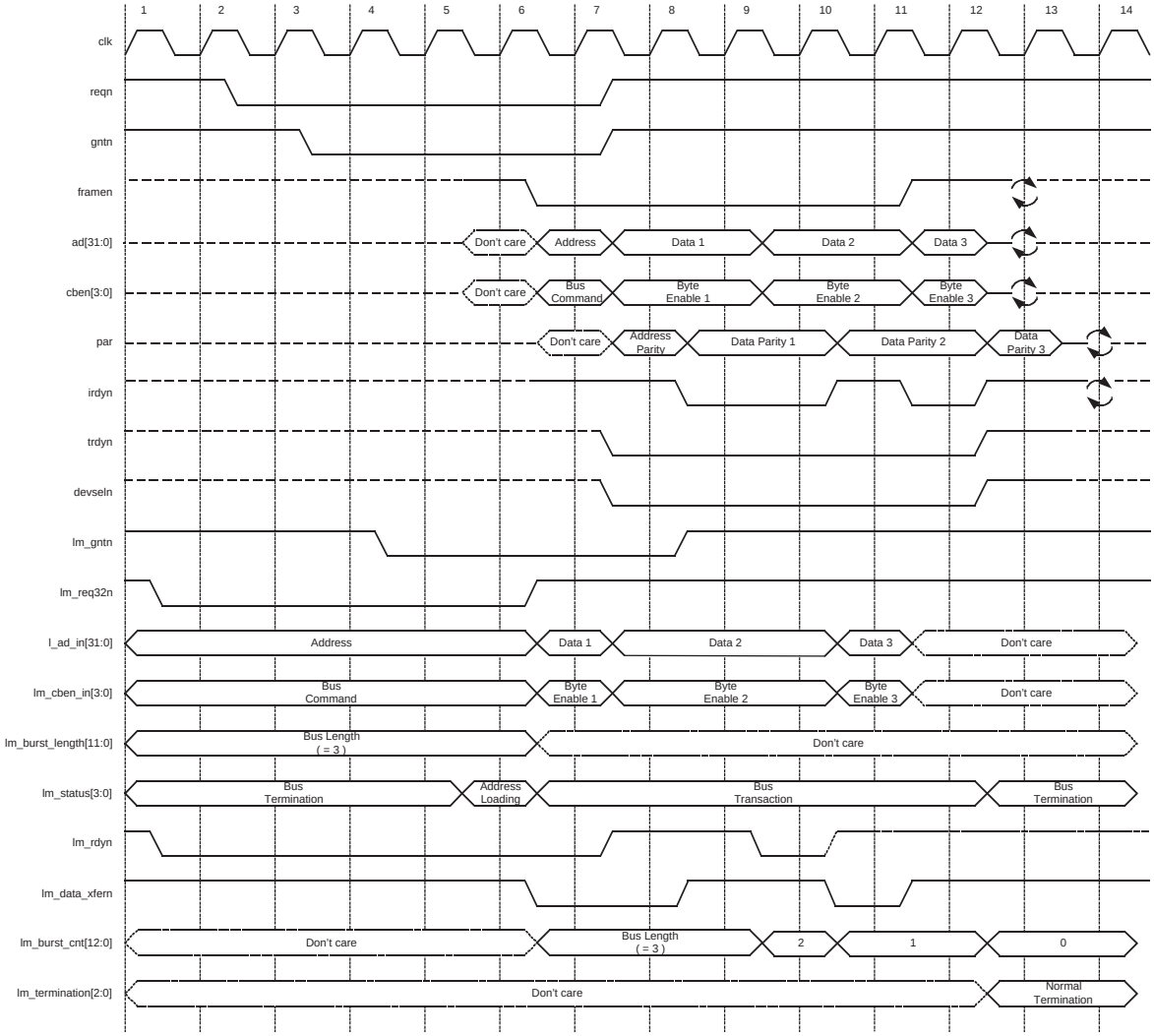



***Table 2-18. 32-bit Master Write Transaction with Local Wait State***

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Idle | The `lm_req32n` signal is asserted on the Local Master interface by the local master to request for 32-bit data transaction. The local master issues the PCI starting address, the bus command, and the burst length during the same clock cycle on `l_ad_in`, `lm_cben_in` and `lm_burst_length`, respectively. |
| 2 | Idle | The Core's Local Master Interface detects the asserted `lm_req32n` and asserts `reqn` to request the use of PCI bus. |
| 3 | Idle | `gntn` is asserted to grant the Core access to the PCI bus. Core is now the PCI master. |
| 4 | Idle | Since `gntn` is asserted and the current bus is idle, the Core is going to start the bus transactions. The Core asserts `lm_gntn` to inform the local master that the bus request is granted. |

*Table 2-18. 32-bit Master Write Transaction with Local Wait State (Continued)*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 5 | Idle | If both lm_req32n and gntn were asserted on the previous cycle, lm_status[3:0] is changed to 'Address Loading' to indicate the starting address, the bus command and the burst length are being latched. |
| 6 | Address | The local master de-asserts lm_req32n when the previous lm_status[3:0] was 'Address Loading' and if it doesn't want to request another PCI bus transaction.<br>The Core asserts framen to initiate the 32-bit write transaction when gntn was asserted and lm_status[3:0] was 'Address Loading' on the previous cycle. It also drives the PCI starting address on ad[31:0] and the PCI command on cben[3:0]. On the same cycle, it outputs lm_status[3:0] as 'Bus Transaction' to indicate the beginning of the address/data phases.<br>lm_burst_cnt gets the value of the burst length.<br>Because lm_rdyn was asserted on the previous cycle, the local master provides Data 1 on l_ad_in[31:0] and byte enable 1 on lm_cben_in[3:0]. And the Core asserts lm_data_xfern to the local master to signify these data and byte enables are being read and will be transferred to the PCI bus.<br>Asserting lm_rdyn means the local master is ready to write data. If it is not, it keeps lm_rdyn de-asserted until it is ready. |
| 7 | Wait | If the target completes the fast decode and is ready to receive 32-bit data, it asserts devseln and trdyn.<br>The Core de-asserts reqn when framen was asserted and lm_req32n was de-asserted on the previous cycle.<br>With lm_data_xfern asserted on the previous cycle that was the address phase, the local master increments the address counter while the Core transfers Data 1 and the byte enables to ad[31:0] and cben[3:0].<br>Because lm_rdyn was asserted on the previous cycle, the local master provides Data 2 on l_ad_in[31:0] and byte enable 2 on lm_cben_in[3:0]. And the Core asserts lm_data_xfern to the local master to signify these data and byte enables are being read and will be transferred to the PCI bus. Data 2 will be buffered and put on PCI bus after Data 1 phase finished.<br>The local master de-asserts lm_rdyn to inform the Core it isn't ready for Data 3. |
| 8 | Data 1 | The Core de-asserts lm_gntn to follow gntn. It asserts irdyn. Framen, Data 1 and the byte enable 1 are kept on the PCI bus. Since the irdyn and trdyn are asserted, the first data phase is completed. The Core de-asserts lm_data_xfern if lm_rdyn was de-asserted on the previous cycle. |
| 9 | Data 2 | The Core asserted irdyn if it has gotten Data 2 from local master interface. It transfers Data 2 and byte enable 2 on ad[31:0] and cben[3:0] respectively.<br>The target keeps devseln and trdyn. Data 2 phase is completed.<br>Since the previous data phase was completed, the Core decreases lm_burst_cnt to two.<br>Lm_rdyn is asserted to ready for Data3. |
| 10 | Wait | Since the Core has not gotten Data 3 from local master interface. It de-asserts irdyn to inform a wait cycle.<br>Because lm_rdyn was asserted on the previous cycle, the local master provides Data 3 on l_ad_in[31:0] and byte enable 3 on lm_cben_in[3:0]. And the Core asserts lm_data_xfern to local master to signify these data and byte enables are being read and will be transferred to the PCI bus.<br>Since the previous data phase was completed, the Core decreases lm_burst_cnt to one. |
| 11 | data 3 | The Core asserts irdyn and de-asserts framen to inform the last data phase. It transfer Data 3 and byte enable The third data phase is completed. |
| 12 | Turn around | The Core relinquishes control of framen, ad and cben. It de-asserts irdyn, decreases lm_burst_cnt to zero and changes lm_status[3:0] into 'Bus Termination' with lm_termination as 'Normal Termination' because both trdyn and irdyn were asserted last cycle. |
| 13 | Idle | The Core relinquishes control of irdyn, par. |

## Burst Read and Write Master Transactions

Burst read and write transactions to memory addresses are used to achieve the high throughput associated with the PCI bus. The PCI IP core supports the zero-wait state and burst data transfers for the following commands:

- Memory Read
- Memory Write
- Memory Read Multiple
- Dual Address Cycle
- Memory Read Line
- Memory Write and Invalidate

The burst data transfers are described based on the different PCI and Local bus configurations supported by the PCI IP core. Although the fundamentals of burst data transfers are similar for all PCI IP core configurations, different bus configurations require slightly different Local Master Interface signaling. The PCI IP core does not execute burst cycles for Configuration Space or I/O space accesses. Refer to the following sections for more information on bursting with specific PCI IP core configurations:

- 32-Bit PCI Master and a 32-Bit Local Bus
- 64-Bit PCI Master with a 64-Bit Local Bus
- 32-Bit PCI Master with a 64-Bit Local Bus

**32-Bit PCI Master and a 32-bit Local Bus**

The following section discusses read and write burst data transfers for a PCI IP core configured with a 32-bit PCI bus and a 32-bit Local bus. Figure 2-15 and Table 2-19 show a 32-bit burst data transfer during a read transaction. The figure illustrates how the PCI interface correlates to the Local Master Interface. The table gives a clock-by-clock description of each event that occurs in the figure.

***Figure 2-15. 32-bit Master Burst Read Transaction with a 32-bit Local Interface***

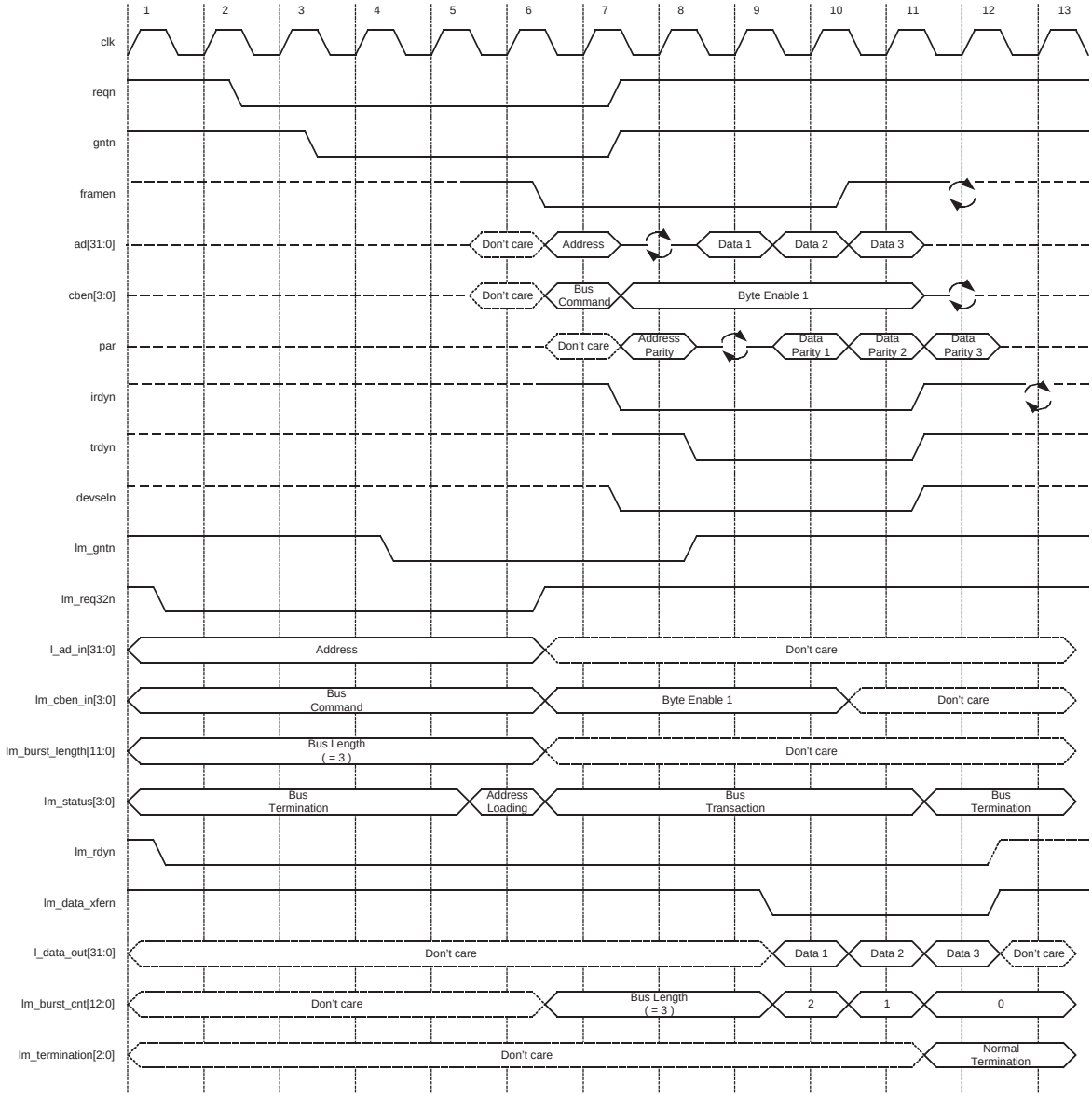



***Table 2-19. 32-bit Master Burst Read Transaction with a 32-bit Local Interface***

| CLK | Phase | Description |
|---|---|---|
| 1 | Idle | The `lm_req32n` signal is asserted by the local master to request a 32-bit data transaction. The local master issues the PCI starting address, the bus command, and the burst length during the same clock cycle to `l_ad_in`, `lm_cben_in`, and `lm_burst_length`, respectively. |
| 2 | Idle | The Core's Local Master Interface detects the asserted `lm_req32n` and asserts `reqn` to request the use of PCI bus. |
| 3 | Idle | `gntn` is asserted to grant the Core access to the PCI bus. Core is now the PCI master. |
| 4 | Idle | Since `gntn` is asserted and the current bus is idle, the Core starts the bus transactions. The master asserts `lm_gntn` to inform the local master that the bus request is granted. |
| 5 | Idle | If both `lm_req32n` and `gntn` were asserted on the previous cycle, `lm_status[3:0]` is changed to 'Address Loading' to indicate the starting address, the bus command, and the burst length are being latched. |

*Table 2-19. 32-bit Master Burst Read Transaction with a 32-bit Local Interface (Continued)*

| CLK | Phase | Description |
|---|---|---|
| 6 | Address | The Core asserts framen to start transaction and the local master de-asserts lm_req32n when the previous lm_status[3:0] was 'Address Loading' and if it doesn't want to request another PCI bus transaction.<br>lm_status[3:0] was 'Address Loading' on the previous cycle. It also drives the PCI starting address on ad[31:0] and the PCI command on cben[3:0]. On the same cycle, it outputs lm_status[3:0] as 'Bus Transaction' to indicate the beginning of the address/data phases.<br>lm_burst_cnt gets the value of the burst length.<br>Because lm_rdyn was asserted on the previous cycle and the next cycle is the first data phase, the local master provides the byte enables on lm_cben_in[3:0]. Asserting lm_rdyn also means the local master is ready to read data. If it is not ready to read data, it keeps lm_rdyn de-asserted until it is ready. |
| 7 | Turn around | The Core de-asserts reqn when framen was asserted but lm_req32n was de-asserted on the previous cycle.<br>The Core tri-states the ad[31:0] lines and drives the byte enables (Byte Enable 1). Since lm_rdyn was asserted on the previous cycle, it asserts irdyn to indicate it is ready to read data.<br>Because the Core performs the burst transactions, it keeps framen asserted. |
| 8 | Data 1 | It de-asserts lm_gntn to follow gntn.<br>The target asserts trdyn and puts Data 1 on ad[31:0].<br>With lm_rdyn asserted on the previous cycle, the Core keeps irdyn asserted.<br>The Core keeps framen asserted to the target to signify the burst continues.<br>If the local master is ready to read the first DWORD, it keeps lm_rdyn asserted.<br>Since both irdyn and trdyn are asserted, the first data phase is completed on this cycle. |
| 9 | Data 2 | Since the previous data phase was completed, the Core transfers Data 2 on l_data_out[31:0] and decreases the lm_burst_cnt.<br>If both trdyn and lm_rdyn were asserted on the previous cycle, the Core asserts lm_data_xfern the local master to signify Data 1 are available on l_data_out[31:0]. With lm_data_xfern asserted, the local master can safely read Data 1 and increment the address counter.<br>If the local master keeps lm_rdyn asserted on the previous cycle, the Core keeps irdyn asserted.<br>The Core keeps framen asserted to the target to signify the burst continues.<br>If the target is still ready to provide data, it keeps trdyn asserted and drives the next DWORD (Data 2) on ad[31:0].<br>If the local master is ready to read the next DWORD, it keeps lm_rdyn asserted.<br>Since both irdyn and trdyn are asserted, the second data phase is completed on this cycle. |
| 10 | Data 3 | Since the previous data phase was completed, the Core transfers Data 3 on l_data_out[31:0] and decreases the lm_burst_cnt.<br>If both trdyn and lm_rdyn were asserted on the previous cycle, the Core asserts lm_data_xfern to the local master to signify Data 2 are available on l_data_out[31:0]. With lm_data_xfern asserted, the local master can safely read Data 2 and increment the address counter.<br>With lm_rdyn asserted on the previous cycle, the Core keeps irdyn asserted.<br>Because the current transaction is the last, the master de-asserts framen to signal the end of the burst.<br>If the target is still ready to provide data, it keeps trdyn asserted and drives the next DWORD (Data 3) on ad[31:0]. If the local master is ready to read the next DWORD, it keeps lm_rdyn asserted.<br>Since both irdyn and trdyn are asserted, the third data phase is completed on this cycle. |

*Table 2-19. 32-bit Master Burst Read Transaction with a 32-bit Local Interface (Continued)*

| CLK | Phase | Description |
|---|---|---|
| 11 | Turn around | Since the previous data phase was completed, the Core transfers Data 3 on l_data_out[31:0] and decreases the lm_burst_cnt to zero.<br>The Core relinquishes control of framen and cben. It de-asserts irdyn and changes lm_status[3:0] into 'Bus Termination' with lm_termination as 'Normal Termination' because both trdyn and irdyn were asserted last cycle.<br>The target relinquishes control of ad[31:0]. It de-asserts devseln and trdyn.<br>If both trdyn and lm_rdyn were asserted on the previous cycle, the Core asserts lm_data_xfern to the local master to signify Data 3 are available on l_data_out[31:0]. With lm_data_xfern asserted, the local master can safely read Data 3. |
| 12 | Idle | The Core relinquishes control of irdyn and de-asserts lm_data_xfern, and the local master de-asserts lm_rdyn since all of the burst data have been read. |

Figure 2-16 and Table 2-20 show an example of a 32-bit burst data transfer during a write transaction with the assumption that the device select timing is set to slow and wait states are not inserted. The figure illustrates how the PCI interface correlates to the Local Master Interface. The table gives a clock-by-clock description of each event that occurs in the figure.

***Figure 2-16. 32-bit Master Burst Write Transaction with a 32-bit Local Interface***

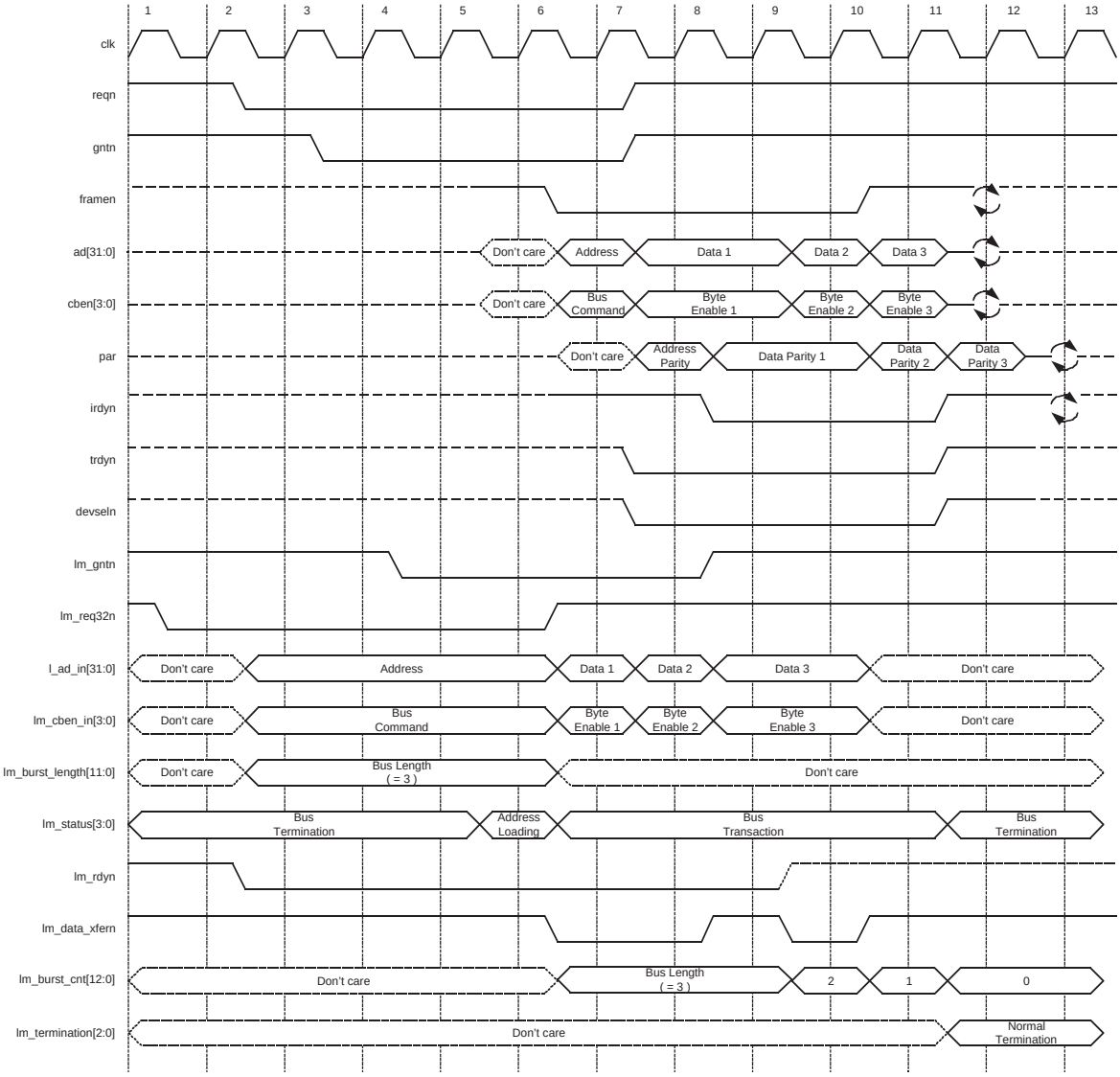



***Table 2-20. 32-bit Master Burst Write Transaction with a 32-Bit Local Interface***

| CLK | Phase | Description |
|---|---|---|
| 1 | Idle | The `lm_req32n` signal is asserted by the local master to request a 32-bit data transaction. The local master issues the PCI starting address, the bus command, and the burst length during the same clock cycle on `l_ad_in`, `lm_cben_in` and `lm_burst_length`, respectively. |
| 2 | Idle | The Core's Local Master Interface detects the asserted `lm_req32n` and asserts `reqn` to request the use of PCI bus. |
| 3 | Idle | `gntn` is asserted to grant the Core access to the PCI bus. Core is now the PCI master. |
| 4 | Idle | Since `gntn` is asserted and the current bus is idle, the Core is going to start the bus transactions. The Core asserts `lm_gntn` to inform the local master that the bus request is granted. |
| 5 | Idle | If both `lm_req32n` and gntn were asserted on the previous cycle, `lm_status[3:0]` is changed to 'Address Loading' to indicate the starting address, the bus command and the burst length are being latched. |

*Table 2-20. 32-bit Master Burst Write Transaction with a 32-Bit Local Interface (Continued)*

| CLK | Phase | Description |
|---|---|---|
| 6 | Address | The local master de-asserts lm_req32n when the previous lm_status[3:0] was 'Address Loading' and if it doesn't want to request another PCI bus transaction.<br>The Core asserts framen to initiate the 32-bit write transaction when gntn was asserted and lm_status[3:0] was 'Address Loading' on the previous cycle. It also drives the PCI starting address on ad[31:0] and the PCI command on cben[3:0]. On the same cycle, it outputs lm_status[3:0] as 'Bus Transaction' to indicate the beginning of the address/data phases.<br>lm_burst_cnt gets the value of the burst length.<br>Because lm_rdyn was asserted on the previous cycle and the next cycle is the first data phase, the local master should provide Data 1 on l_ad_in[31:0] and the byte enables on lm_cben_in[3:0]. And the Core asserts lm_data_xfern to the local master to signify these data and byte enables are being read and will be transferred to the PCI bus.<br>Asserting lm_rdyn means the local master is ready to write data. If it is not, it should keep lm_rdyn de-asserted until it is ready. |
| 7 | Wait | The Core de-asserts reqn after the assertion of framen.<br>lm_data_xfern is asserted to signify Data 2 on l_ad_in[31:0] and the byte enables on lm_cben_in[3:0] are being read and will be transferred to the PCI bus.<br>If the local master is ready to provide the next DWORD, it keeps lm_rdyn asserted. |
| 8 | Data 1 | The Core keeps framen asserted and asserts irdyn. It also de-asserts lm_data_xfern to the local master to signify Data 3 on l_ad_in[31:0] is not read.<br>If the local master is ready to provide the next DWORD, it keeps lm_rdyn asserted.<br>Because the Core performs the burst transactions, it keeps framen asserted.<br>Since both irdyn and trdyn are asserted, the first data phase is completed on this cycle. |
| 9 | Data 2 | Since the previous data phase was completed, the Core decreases lm_burst_cnt.<br>Since Data 1 on PCI bus were read by the target, the Core transfers Data 2 and their byte enables to ad[31:0] and cben[3:0].<br>With lm_rdyn asserted previous cycle, the Core keeps irdyn asserted.<br>Because both lm_rdyn and trdyn were asserted on the previous cycle, the Core asserts lm_data_xfern to the local master to signify Data 3 on l_ad_in[31:0] and the byte enables on lm_cben_in[3:0] are being read and will be transferred to the PCI bus.<br>Because Data 3 are the last data, the local master de-asserts lm_rdyn.<br>Since both irdyn and trdyn are asserted, the second data phase is completed on this cycle. |
| 10 | Data 3 | Since the previous data phase was completed, the Core decreases lm_burst_cnt.<br>Since Data 2 on the PCI bus were read, the Core transfers Data 3 and their byte enables to ad[31:0] and cben[3:0].<br>Because the current transaction is the last, the Core de-asserts framen to signal the end of the burst, also it de-asserts lm_data_xfern.<br>Since both irdyn and trdyn are asserted, the third data phase is completed on this cycle. |
| 11 | Turn around | The Core relinquishes control of framen, ad and cben. It de-asserts irdyn, decreases lm_burst_cnt to zero and changes lm_status[3:0] into 'Bus Termination' with lm_termination as 'Normal Termination' because both trdyn and irdyn were asserted last cycle. The target de-asserts devseln and trdyn. |
| 12 | Idle | The Core relinquishes control of irdyn and par. |

### 64-bit PCI Master with a 64-bit Local Bus

The following discusses read and write burst transactions for the PCI IP core configured with a 64-bit PCI bus and a 64-bit Local bus. Figure 2-17 and Table 2-21 illustrate a 64-bit burst write transaction. The figure shows how the PCI Interface correlates to the Local Master Interface. The table gives a clock-by-clock description of each event that occurs in the figure.

The 32-bit burst transaction is similar to a 32-bit burst transaction for the 64-bit PCI IP core configuration. When the 64-bit target core responds to a 32-bit burst transaction, the upper 32 bits of the data bus are ignored.

***Figure 2-17. 64-bit Master Burst Read Transaction with a 64-bit Local Interface***

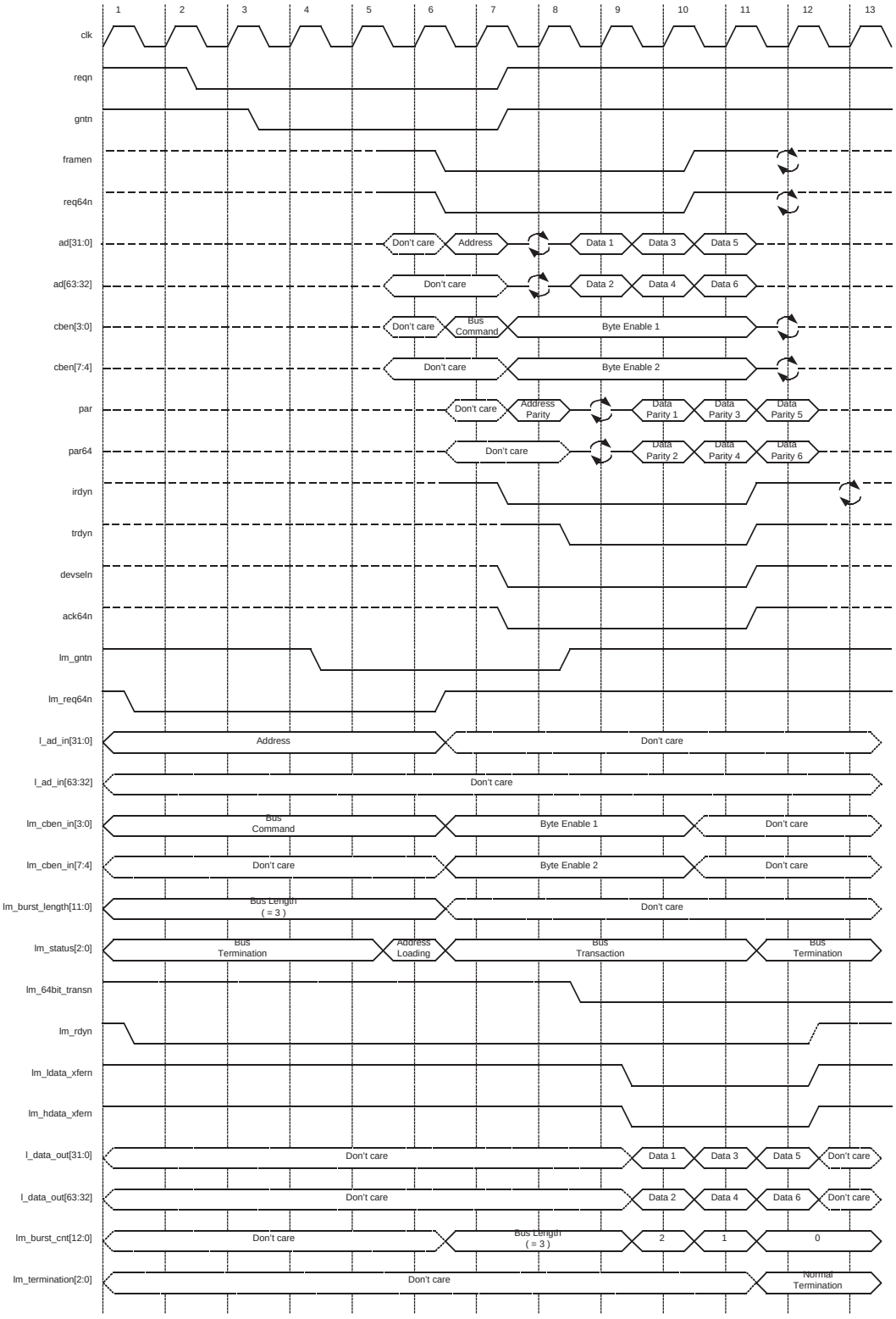


*Table 2-21. 64-bit Master Burst Read Transaction with a 64-bit Local Interface*

| CLK | Phase | Description |
|---|---|---|
| 1 | Idle | The local master asserts lm_req64n to request 64-bit wide data transaction. It also issues the PCI starting address, the bus command and the burst length will be available on l_ad_in, lm_cben_in and lm_burst_length respectively on the same clock cycle. |
| 2 | Idle | The Core's Local Master Interface detects the asserted lm_req64n and asserts reqn to request the use of PCI bus. |
| 3 | Idle | gntn is asserted to grant the Core access to the PCI bus. Core is now the PCI master. |
| 4 | Idle | Since gntn is asserted and the current bus is idle, the Core is going to start the bus transactions. The Core asserts lm_gntn to inform the local master that the bus request is granted. |
| 5 | Idle | If both lm_req64n and lm_gntn were asserted on the previous cycle, lm_status[3:0] is changed to 'Address Loading' to indicate the starting address, the bus command and the burst length are being latched. |
| 6 | Address | The local master de-asserts lm_req64n when the previous lm_status[3:0] was 'Address Loading' and if it doesn't want to request another PCI bus transaction.<br>The Core asserts framen and req64n to initiate the 64-bit read transaction when gntn was still asserted and lm_status[3:0] was 'Address Loading' on the previous cycle. It also drives the PCI starting address on ad[31:0] and the PCI command on cben[3:0]. On the same cycle, it outputs lm_status[3:0] as 'Bus Transaction' to indicate the beginning of the address/data phases.<br>lm_burst_cnt gets the value of the burst length.<br>Because lm_rdyn was asserted on the previous cycle and the next cycle is the first data phase, the local master provides the byte enables on lm_cben_in[7:0]. Asserting lm_rdyn also means the local master is ready to read data. If it is not ready to read data, it keeps lm_rdyn de-asserted until it is ready. |
| 7 | Turn around | The Core de-asserts reqn when framen was asserted but lm_req64n was de-asserted on the previous cycle.<br>The target asserts devseln and ack64n to indicate it acknowledges the 64-bit transaction. The Core tri-states the ad[63:0] lines and drives the byte enables (Byte Enable 1 and 2). Since lm_rdyn was asserted on the previous cycle, it asserts irdyn to indicate it is ready to read data.<br>Because the Core performs burst data transfer, it keeps framen asserted. |
| 8 | Data 1 and 2 | The Core asserts lm_64bit_transn to indicate the current data transaction is 64 bits wide. It de-asserts lm_gntn to follow gntn.<br>The target asserts trdyn and puts Data 1 and 2 on ad[63:0].<br>With lm_rdyn asserted on the previous cycle, the Core keeps irdyn asserted.<br>The Core keeps framen asserted to the target to signify the burst continues. It de-asserts lm_gntn to follow gntn.<br>If the local master is ready to read the first QWORD, it keeps lm_rdyn asserted.<br>Since both irdyn and trdyn are asserted, the first data phase is completed on this cycle. |
| 9 | Data 3 and 4 | Since the previous data phase was completed, the Core transfers Data 1 and 2 on l_data_out[63:0] and decreases the lm_burst_cnt.<br>If both trdyn and lm_rdyn were asserted on the previous cycle, the Core asserts lm_ldata_xfern and lm_hdata_xfern to the local master to signify Data 1 and 2 are available on l_data_out[63:0]. With lm_ldata_xfern and lm_hdata_xfern asserted, the local master can safely read Data 1 and 2 and increment the address counter.<br>If the local master keeps lm_rdyn asserted on the previous cycle, the Core keeps irdyn asserted.<br>The Core keeps framen asserted to the target to signify the burst continues.<br>If the target is still ready to provide data, it keeps trdyn asserted and drives the next QWORD (Data 3 and 4) on ad[63:0].<br>If the local master is ready to read the next QWORD, it keeps lm_rdyn asserted.<br>Since both irdyn and trdyn are asserted, the second data phase is completed on this cycle. |

*Table 2-21. 64-bit Master Burst Read Transaction with a 64-bit Local Interface (Continued)*

| CLK | Phase | Description |
|---|---|---|
| 10 | Data 5 and 6 | Since the previous data phase was completed, the Core transfers Data 3 and 4 on l_data_out[63:0] and decreases the lm_burst_cnt.<br>If both trdyn and lm_rdyn were asserted on the previous cycle, the Core asserts lm_ldata_xfern and lm_hdata_xfern to the local master to signify Data 3 and 4 are available on l_data_out[63:0]. With lm_ldata_xfern and lm_hdata_xfern asserted, the local master can safely read Data 3 and 4 and increment the address counter.<br>With lm_rdyn asserted on the previous cycle, the Core keeps irdyn asserted.<br>Because the current transaction is the last, the Core de-asserts framen and req64n to signal the end of the burst.<br>If the target is still ready to provide data, it keeps trdyn asserted and drives the next QWORD (Data 5 and 6) on ad[63:0].<br>If the local master is ready to read the next QWORD, it keeps lm_rdyn asserted.<br>Since both irdyn and trdyn are asserted, the third data phase is completed on this cycle. |
| 11 | Turn around | Since the previous data phase was completed, the Core transfers Data 5 and 6 on l_data_out[63:0] and decreases the lm_burst_cnt to zero.<br>The Core relinquishes control of framen, req64n and cben. It de-asserts irdyn and changes lm_status[3:0] into 'Bus Termination' with lm_termination as 'Normal Termination' because both trdyn and irdyn were asserted last cycle.<br>The target relinquishes control of ad[63:0]. It de-asserts devseln, ack64n and trdyn.<br>If both trdyn and lm_rdyn were asserted on the previous cycle, the Core asserts lm_ldata_xfern and lm_hdata_xfern to the local master to signify Data 5 and 6 are available on l_data_out[63:0]. With lm_ldata_xfern and lm_hdata_xfern asserted, the local master can safely read Data 5 and 6. |
| 12 | Idle | The Core relinquishes control of irdyn and de-asserts lm_ldata_xfern and lm_hdata_xfern, and the local master de-asserts lm_rdyn since all of the burst data have been read. |

Figure 2-18 and Table 2-22 illustrate a 64-bit burst write transaction. The figure shows how the PCI interface correlates to the Local Master Interface. The table gives a clock-by-clock description of each event that occurs in the figure.

***Figure 2-18. 64-bit Master Burst Write Transaction with a 64-bit Local Interface***

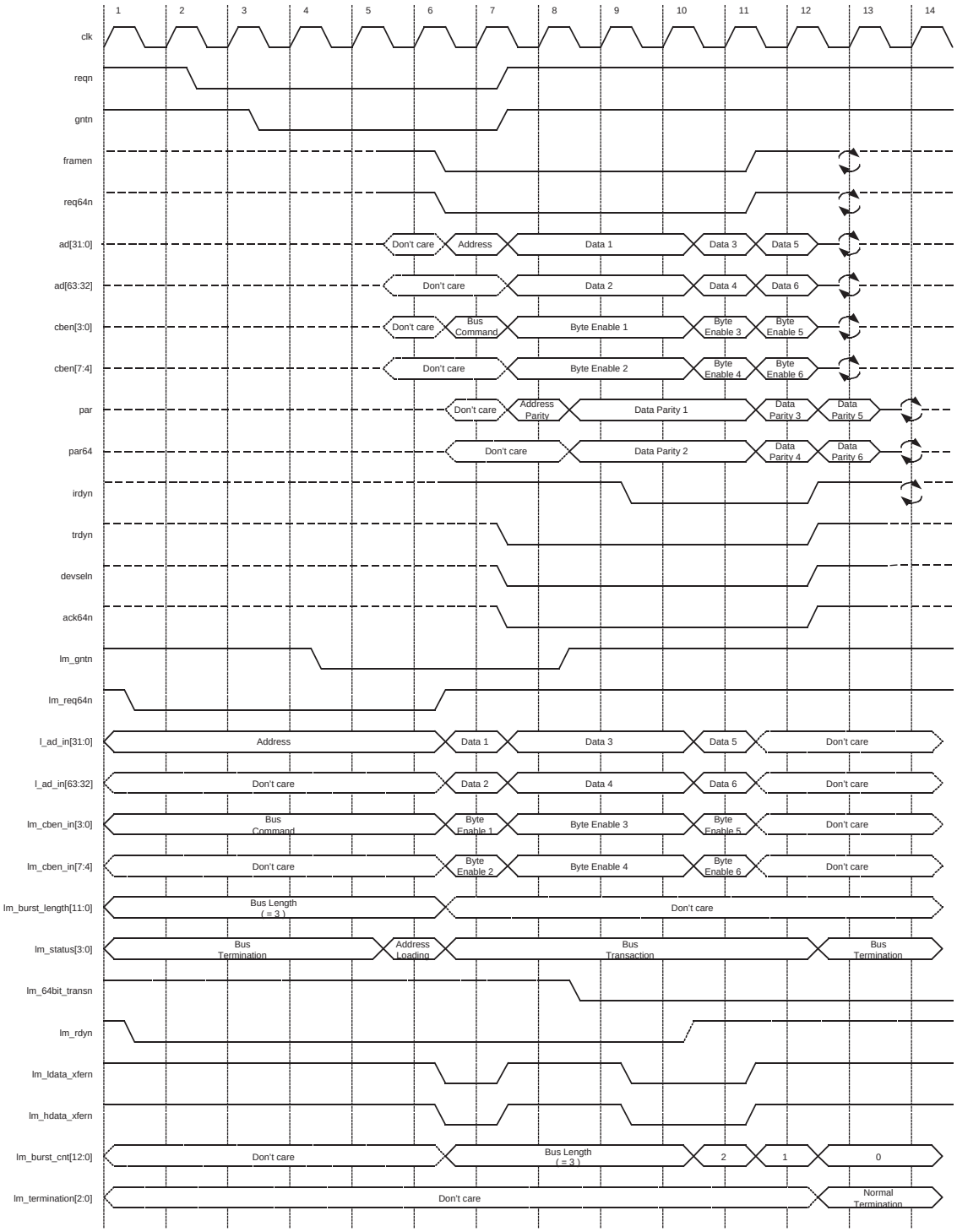


*Table 2-22. 64-bit Master Burst Write Transaction with a 64-bit Local Interface*

| CLK | Phase | Description |
|---|---|---|
| 1 | Idle | The local master asserts lm_req64n to request for 64-bit wide data transaction. It also issues the PCI starting address, the bus command and the burst length on l_ad_in, lm_cben_in and lm_burst_length respectively on the same clock cycle. |
| 2 | Idle | The Core's Local Master Interface detects the asserted lm_req64n and asserts reqn to request the use of PCI bus. |
| 3 | Idle | gntn is asserted to grant the Core access to the PCI bus. Core is now the PCI master. |
| 4 | Idle | Since gntn is asserted and the current bus is idle, the Core is going to start the bus transactions. The Core asserts lm_gntn to inform the local master that the bus request is granted. |
| 5 | Idle | If both lm_req64n and lm_gntn were asserted on the previous cycle, lm_status[3:0] is changed to 'Address Loading' to indicate the starting address, the bus command and the burst length are being latched. |
| 6 | Address | The local master de-asserts lm_req64n when the previous lm_status[3:0] was 'Address Loading' and if it doesn't want to request another PCI bus transaction.<br>The Core asserts framen and req64n to initiate the 64-bit write transaction when gntn was asserted and lm_status[3:0] was 'Address Loading' on the previous cycle. It also drives the PCI starting address on ad[31:0] and the PCI command on cben[3:0]. On the same cycle, it outputs lm_status[3:0] as 'Bus Transaction' to indicate the beginning of the address/data phases.<br>lm_burst_cnt gets the value of the burst length.<br>Because lm_rdyn was asserted on the previous cycle and the next cycle is the first data phase, the local master provides Data 1 and Data 2 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0]. And the Core asserts lm_ldata_xfern and lm_hdata_xfern to the local master to signify these data and byte enables are being read and will be transferred to the PCI bus.<br>Asserting lm_rdyn means the local master is ready to write data. If it is not, it keeps lm_rdyn de-asserted until it is ready. |
| 7 | Wait | The Core de-asserts reqn when framen was asserted but lm_req64n was de-asserted on the previous cycle.<br>If the target completes the fast decode and is ready to receive 64-bit data, it asserts devseln, ack64n and trdyn.<br>With lm_ldata_xfern and lm_hdata_xfern asserted on the previous cycle that was the address phase, the local master should increment the address counter while the Core transfers Data 1 and Data 2 and their byte enables to ad[63:0] and cben[7:0].<br>With lm_rdyn asserted on the previous cycle, the local master provides Data 3 and Data 4 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0]. Because this is the first write data phase and devseln is just asserted, the Core keeps framen asserted and irdyn de-asserted to judge 64-bit or 32-bit transactions. It also de-asserts lm_ldata_xfern and lm_hdata_xfern to the local master to signify that Data 3 and Data 4 on l_ad_in[63:0] are not read.<br>If the local master is ready to provide the next QWORD, it keeps lm_rdyn asserted.<br>Since irdyn is not asserted, the first data phase is not completed. |
| 8 | Wait | Since lm_ldata_xfern and lm_hdata_xfern were not asserted on the previous cycle, the local master keeps Data 3 and Data 4 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0].<br>Because the Core needs one more cycle to decide 64-bit or 32-bit transaction, it keeps framen asserted and irdyn de-asserted. It also keeps lm_ldata_xfern and lm_hdata_xfern de-asserted to the local master to signify Data 3 and Data 4 on l_ad_in[63:0] are not read.<br>The Core asserts lm_64bit_transn to indicate the current data transaction is 64 bits wide. It de-asserts lm_gntn to follow gntn.<br>If the local master is ready to provide the next QWORD, it keeps lm_rdyn asserted.<br>Since irdyn is not asserted, the first data phase is not completed. |

*Table 2-22. 64-bit Master Burst Write Transaction with a 64-bit Local Interface (Continued)*

| CLK | Phase | Description |
|---|---|---|
| 9 | Data 1 and 2 | Since lm_ldata_xfern and lm_hdata_xfern were not asserted on the previous cycle, the local master keeps Data 3 and Data 4 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0].<br>With both devseln and lm_rdyn asserted in the previous cycle, the Core asserts irdyn, and it prepares for the 64-bit write burst. So it asserts lm_ldata_xfern and lm_hdata_xfern to the local master to signify Data 3 and Data 4 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0] are being read and will be transferred to the PCI bus.<br>The Core keeps framen asserted and asserts irdyn. It also keeps lm_ldata_xfern and lm_hdata_xfern de-asserted to the local master to signify Data 3 and Data 4 on l_ad_in[63:0] are not read.<br>If the local master is ready to provide the next QWORD, it keeps lm_rdyn asserted.<br>Because the Core performs the burst transactions, it keeps framen asserted.<br>Since both irdyn and trdyn are asserted, the first data phase is completed on this cycle. |
| 10 | Data 3 and 4 | Since the previous data phase was completed, the Core decreases 'lm_burst_cnt'.<br>With lm_ldata_xfern and lm_hdata_xfern asserted on the previous cycle, the local master should increment the address counter.<br>Since Data 1 and Data 2 on the PCI bus were read by the target, the Core transfers Data 3 and Data 4 and their byte enables to ad[63:0] and cben[7:0].<br>With lm_rdyn asserted previous cycle, the Core keeps irdyn asserted.<br>With lm_ldata_xfern and lm_hdata_xfern asserted on the previous cycle, the local master provides Data 5 and Data 6 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0].<br>Because both lm_rdyn and trdyn were asserted on the previous cycle, the Core asserts lm_ldata_xfern and lm_hdata_xfern to the local master to signify Data 5 and Data 6 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0] are being read and will be transferred to the PCI bus. Because Data 5 and Data 6 are the last data, the local master de-asserts lm_rdyn. The Core keeps framen asserted to signify the burst continues. Since both irdyn and trdyn are asserted, the second data phase is completed on this cycle. |
| 11 | Data 5 and 6 | Since the previous data phase was completed, the Core decreases 'lm_burst_cnt'.<br>Since Data 3 and Data 4 on PCI bus were read, the Core transfers Data 5 and Data 6 and their byte enables to ad[63:0] and cben[7:0].<br>With lm_rdyn asserted previous cycle, the Core keeps irdyn asserted.<br>Because the current transaction is the last, the Core de-asserts framen and req64n to signal the end of the burst, also it de-asserts lm_ldata_xfern and lm_hdata_xfern.<br>Since both irdyn and trdyn are asserted, the third data phase is completed on this cycle. |
| 12 | Turn around | The Core relinquishes control of framen, req64n, ad and cben. It de-asserts irdyn, decreases 'lm_burst_cnt' to zero and changes lm_status[3:0] into 'Bus Termination' with lm_termination as 'Normal Termination' because both trdyn and irdyn were asserted last cycle. The target de-asserts devseln, ack64n and trdyn. |
| 13 | Idle | The Core relinquishes control of irdyn, par and par64. |

**32-bit PCI Master with a 64-Bit Local Bus**

The following discusses read and write transactions for a PCI IP core configured with a 32-bit PCI bus and a 64-bit local bus. Two PCI data phases are required when writing or reading 64-bit data via the Local Master Interface.

The 32-bit PCI transaction, as described in the 32-Bit PCI Master and 32-Bit Local Bus section, is similar to these transactions; however, 32-bit data on a 64-bit data path is handled differently at the Local Master Interface. When the 64-bit target core responds to a 32-bit transaction, the upper 32 bits of the Local data bus should be ignored or return 0's.

With a 64-bit back-end, the address counter needs to increment only by a QWORD (eight bytes). As a result, the local back-end control latches the complete QWORD and routes the proper DWORD to the PCI data bus. The lm_ldata_xfern and lm_hdata_xfern signals specify which DWORD is transferred.

If the starting address is QWORD aligned, the first DWORD is assumed to be the lower DWORD of a QWORD. Otherwise, it is the upper DWORD. If the starting address is not QWORD aligned, it must be DWORD aligned.

Figure 2-19 and Table 2-23 illustrate a burst transaction to a 32-bit PCI IP core with a 64-bit Local Master Interface. The figure illustrates how the PCI interface correlates to the Local Master Interface. The table gives a clock-by-clock description of each event in the figure.

***Figure 2-19. 32-bit Master Burst Read Transaction with a 64 bit Local Interface***

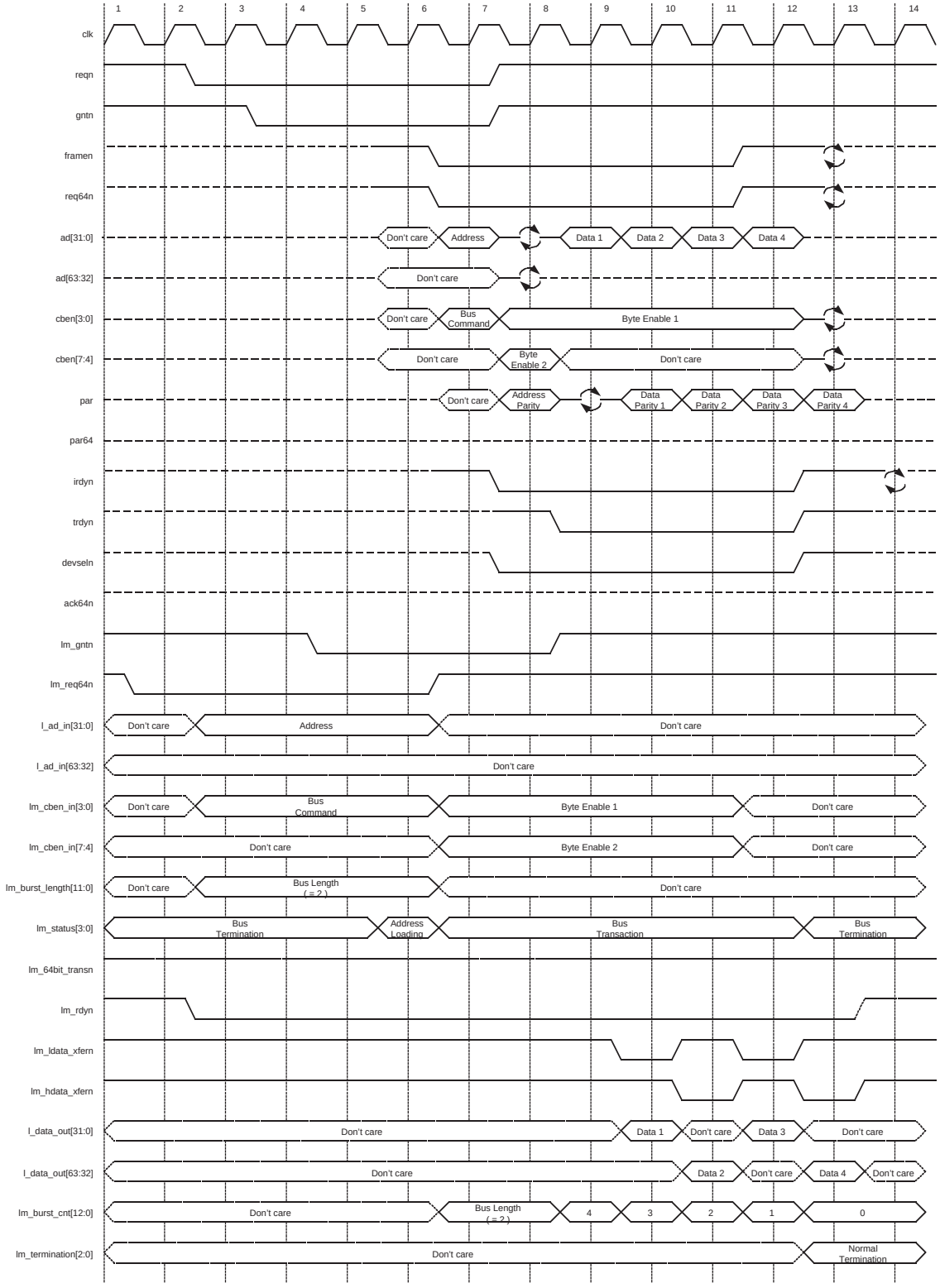


*Table 2-23. 32-bit Master Burst Read Transaction with a 64-Bit Local Interface*

| CLK | Phase | Description |
|---|---|---|
| 1 | Idle | The lm_req32n signal is asserted by the local master to request 32-bit data transaction. The local master issues the PCI starting address, the bus command, and the burst length during the same clock cycle to l_ad_in, lm_cben_in, and lm_burst_length, respectively. |
| 2 | Idle | The Core's Local Master Interface detects the asserted lm_req32n and asserts reqn to request the use of PCI bus. |
| 3 | Idle | gntn is asserted to grant the Core access to the PCI bus. Core is now the PCI master. |
| 4 | Idle | Since gntn is asserted and the current bus is idle, the Core starts the bus transactions. The Core asserts lm_gntn to inform the local master that the bus request is granted. |
| 5 | Idle | If both lm_req32n and gntn were asserted on the previous cycle, lm_status[3:0] is changed to 'Address Loading' to indicate the starting address, the bus command, and the burst length are being latched. |
| 6 | Address | The Core asserts framen to start transaction and the local master de-asserts lm_req32n when the previous lm_status[3:0] was 'Address Loading' and if it doesn't want to request another PCI bus transaction.<br>lm_status[3:0] was 'Address Loading' on the previous cycle. It also drives the PCI starting address on ad[31:0] and the PCI command on cben[3:0]. On the same cycle, it outputs lm_status[3:0] as 'Bus Transaction' to indicate the beginning of the address/data phases.<br>lm_burst_cnt gets the value of the burst length.<br>Because lm_rdyn was asserted on the previous cycle and the next cycle is the first data phase, the local master provides the byte enables on lm_cben_in[3:0]. Asserting lm_rdyn also means the local master is ready to read data. If it is not ready to read data, it keep lm_rdyn de-asserted until it is ready. |
| 7 | Turn around | The Core de-asserts reqn when framen was asserted but lm_req64n was de-asserted on the previous cycle.<br>The target only asserts devseln to indicate it doesn't acknowledge the 64-bit transaction.<br>The Core tri-states the ad[31:0] lines and drives the byte enables (Byte Enable 1). Since lm_rdyn was asserted on the previous cycle, it asserts irdyn to indicate it is ready to read data. Because the master performs the burst transactions, it keeps framen asserted. |
| 8 | Data 1 | The Core de-asserts lm_64bit_transn to indicate the current data transaction is 32-bit wide. It de-asserts lm_gntn to follow gntn.<br>The target asserts trdyn and puts Data 1 on ad[31:0].<br>With lm_rdyn asserted on the previous cycle, the Core keeps irdyn asserted.<br>The Core keeps framen asserted to the target to signify the burst continues.<br>If the local master is ready to read the first DWORD, it keeps lm_rdyn asserted.<br>Since both irdyn and trdyn are asserted, the first data phase is completed on this cycle. |
| 9 | Data 2 | Since the previous data phase was completed, the Core transfers Data 1 on l_data_out[31:0] and decreases the lm_burst_cnt.<br>If both trdyn and lm_rdyn were asserted on the previous cycle, the Core asserts lm_ldata_xfern and de-asserts lm_hdata_xfern to the local master to signify Data 1 are available on l_data_out[31:0]. With lm_ldata_xfern asserted, the local master can safely read Data 1 and increment the address counter.<br>If the local master keeps lm_rdyn asserted on the previous cycle, the Core keeps irdyn asserted.<br>The Core keeps framen asserted to the target to signify the burst continues.<br>If the target is still ready to provide data, it keeps trdyn asserted and drives the next DWORD (Data 2) on ad[31:0].<br>If the local master is ready to read the next DWORD, it keeps lm_rdyn asserted. Since both irdyn and trdyn are asserted, the second data phase is completed on this cycle. |

*Table 2-23. 32-bit Master Burst Read Transaction with a 64-Bit Local Interface (Continued)*

| CLK | Phase | Description |
|---|---|---|
| 10 | Data 3 | Since the previous data phase was completed, the Core transfers Data 2 on l_data_out[63:32] and decreases the lm_burst_cnt.<br>If both trdyn and lm_rdyn were asserted on the previous cycle, the Core de-asserts lm_ldata_xfern and asserts lm_hdata_xfern to the local master to signify Data 2 are available on l_data_out[63:32]. With lm_hdata_xfern asserted, the local master can safely read Data 2 and increment the address counter.<br>With lm_rdyn asserted on the previous cycle, the Core keeps irdyn asserted. If the target is still ready to provide data, it keeps trdyn asserted and drives the next DWORD (Data 3) on ad[31:0].<br>If the local master is ready to read the next DWORD, it keeps lm_rdyn asserted.<br>Since both irdyn and trdyn are asserted, the third data phase is completed on this cycle. |
| 11 | Data 4 | Since the previous data phase was completed, the Core transfers Data 3 on l_data_out[31:0] and decreases the lm_burst_cnt.<br>If both trdyn and lm_rdyn were asserted on the previous cycle, the Core asserts lm_ldata_xfern and de-asserts lm_hdata_xfern to the local master to signify Data 3 are available on l_data_out[31:0]. With lm_ldata_xfern asserted, the local master can safely read Data 3 and increment the address counter.<br>With lm_rdyn asserted on the previous cycle, the Core keeps irdyn asserted.<br>Because the current transaction is the last, the Core de-asserts framen and req64n to signal the end of the burst. If the target is still ready to provide data, it keeps trdyn asserted and drives the next DWORD (Data 4) on ad[31:0].<br>If the local master is ready to read the next DWORD, it keeps lm_rdyn asserted.<br>Since both irdyn and trdyn are asserted, the fourth data phase is completed on this cycle. |
| 12 | Turn around | Since the previous data phase was completed, the Core transfers Data 4 on l_data_out[63:32] and decreases the lm_burst_cnt to zero.<br>The Core relinquishes control of framen, req64n and cben. It de-asserts irdyn and changes lm_status[3:0] into 'Bus Termination' with lm_termination as 'Normal Termination' because both trdyn and irdyn were asserted last cycle.<br>The target relinquishes control of ad[31:0]. It de-asserts devseln and trdyn.<br>If both trdyn and lm_rdyn were asserted on the previous cycle, the Core de-asserts lm_ldata_xfern and asserts lm_hdata_xfern to the local master to signify Data 4 are available on l_data_out[63:32]. With lm_hdata_xfern asserted, the local master can safely read Data 4. |
| 12 | Idle | The Core relinquishes control of irdyn and de-asserts lm_ldata_xfern and lm_hdata_xfern, and the local master de-asserts lm_rdyn since all of the burst data have been read. |

Figure 2-20 and Table 2-24 illustrate a burst transaction for a 32-bit PCI IP core with a 64-bit Local Interface. The figure shows how the PCI interface correlates to the Local Interface. The table gives a clock-by-clock description of each event illustrated in the figure.

***Figure 2-20. 32-bit Master Burst Write Transaction With a 64-bit Local Interface***

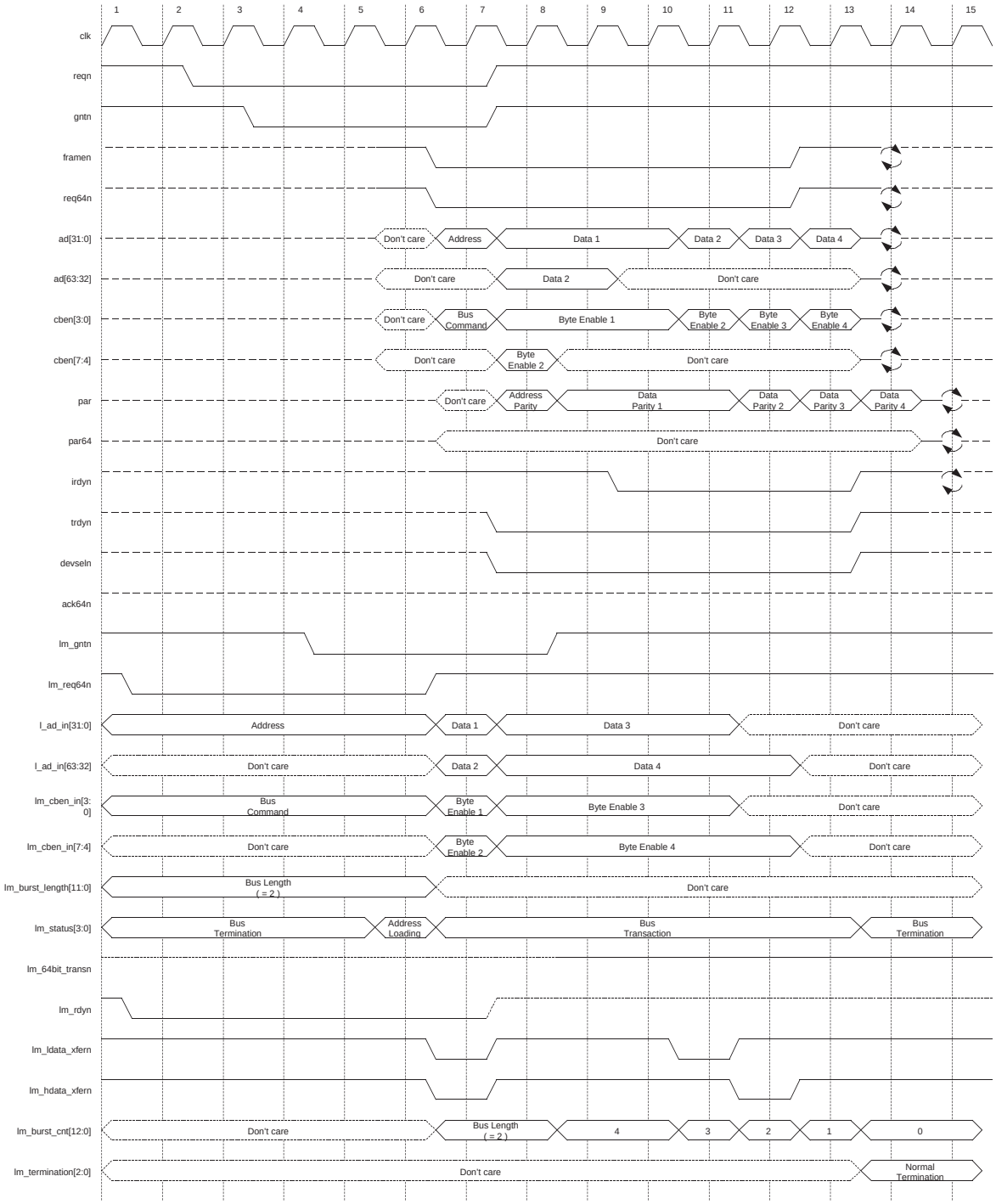


***Table 2-24. 32-bit Master Burst Write Transaction With a 64-bit Local Interface***

| CLK | Phase | Description |
|---|---|---|
| 1 | Idle | The local master asserts lm_req32n for the master 32-bit data transaction request. It also issues the PCI starting address, the bus command and the burst length on l_ad_in, lm_cben_in and lm_burst_length respectively during the same clock cycle. |
| 2 | Idle | The Core's Local Master Interface detects the asserted lm_req64n and asserts reqn to request the use of PCI bus. |
| 3 | Idle | gntn is asserted to grant the Core access to the PCI bus. Core is now the PCI master. |
| 4 | Idle | Since gntn is asserted and the current bus is idle, the Core starts the bus transactions. The Core asserts lm_gntn to inform the local master that the bus request is granted. |
| 5 | Idle | If both lm_req64n and gntn were asserted on the previous cycle, lm_status[3:0] is changed to 'Address Loading' to indicate the starting address, the bus command and the burst length are being latched. |
| 6 | Address | The local master de-asserts lm_req64n when the previous lm_status[3:0] was 'Address Loading' and if it doesn't want to request another PCI bus transaction.<br>The Core asserts framen and req64n to initiate the 64-bit write transaction when gntn was asserted and lm_status[3:0] was 'Address Loading' on the previous cycle. It also drives the PCI starting address on ad[31:0] and the PCI command on cben[3:0]. On the same cycle, it outputs lm_status[3:0] as 'Bus Transaction' to indicate the beginning of the address/data phases.<br>lm_burst_cnt gets the value of the burst length.<br>Because lm_rdyn was asserted on the previous cycle and the next cycle is the first data phase, the local master should provide Data 1 and Data 2 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0]. And the Core asserts lm_ldata_xfern and lm_hdata_xfern to the local master to signify these data and byte enables are being read and will be transferred to the PCI bus.<br>Asserting lm_rdyn means the local master is ready to write data. If it is not, it should keep lm_rdyn de-asserted until it is ready. |
| 7 | Wait | The Core de-asserts reqn when framen was asserted but lm_req64n was de-asserted on the previous cycle.<br>If the target completes the fast decode and is ready to receive 32-bit data, it asserts devseln and trdyn and doesn't asserts ack64n.<br>With lm_ldata_xfern and lm_hdata_xfern asserted on the previous cycle that was the address phase, the local master should increment the address counter while the Core transfers Data 1 and Data 2 and their byte enables to ad[63:0] and cben[7:0].<br>With lm_rdyn asserted on the previous cycle, the local master provides Data 3 and Data 4 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0].<br>Because this is the first write data phase and devseln is just asserted, the Core keeps framen asserted and irdyn de-asserted to judge 64-bit or 32-bit transaction. It also de-asserts lm_ldata_xfern and lm_hdata_xfern to the local master to signify Data 3 and Data 4 on l_ad_in[63:0] are not read.<br>Since irdyn is not asserted, the first data phase is not completed. |
| 8 | Wait | Since lm_ldata_xfern and lm_hdata_xfern were not asserted on the previous cycle, the local master keeps Data 3 and Data 4 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0].<br>Because the Core needs one more cycle to decide 64-bit or 32-bit transaction, it keeps framen asserted and irdyn de-asserted. It also keeps lm_ldata_xfern and lm_hdata_xfern de-asserted to the local master to signify Data 3 and Data 4 on l_ad_in[63:0] are not read.<br>The Core de-asserts lm_64bit_transn and changes lm_burst_cnt to four to indicate the current data transaction is 32-bit wide. It de-asserts lm_gntn to follow gntn.<br>Since irdyn is not asserted, the first data phase is not completed. |

*Table 2-24. 32-bit Master Burst Write Transaction With a 64-bit Local Interface (Continued)*

| CLK | Phase | Description |
|---|---|---|
| 9 | Data 1 | Since lm_ldata_xfern and lm_hdata_xfern were not asserted on the previous cycle, the local master keeps Data 3 and Data 4 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0].<br>With both devseln and lm_rdyn asserted previous cycle, the master asserts irdyn, and it prepares for the 32-bit write burst.<br>The Core keeps framen asserted. It also keeps lm_ldata_xfern and lm_hdata_xfern de-asserted to the local master to signify Data 3 and Data 4 on l_ad_in[63:0] are not read.<br>Because the Core performs the burst transactions, it keeps framen asserted.<br>Since both irdyn and trdyn are asserted, the first data phase is completed on this cycle. |
| 10 | Data 2 | Since the previous data phase was completed, the Core decreases 'lm_burst_cnt'.<br>Since Data 1 on PCI bus were read by the target, the Core transfers Data 2 and their byte enables to ad[31:0] and cben[3:0].<br>The Core keeps irdyn asserted.<br>Because trdyn were asserted on the previous cycle, the Core asserts lm_ldata_xfern and de-asserts lm_hdata_xfern to the local master to signify Data 3 on l_ad_in[31:0] and the byte enables on lm_cben_in[3:0] are being read and will be transferred to the PCI bus.<br>The Core keeps framen asserted to the target to signify the burst continues.<br>Since both irdyn and trdyn are asserted, the second data phase is completed on this cycle. |
| 11 | Data 3 | Since the previous data phase was completed, the Core decreases 'lm_burst_cnt'.Since Data 2 on PCI bus was read, the Core transfers Data 3 and its byte enables to ad[31:0] and cben[3:0].The Core keeps irdyn asserted. Because trdyn was asserted on the previous cycle, the master de-asserts lm_ldata_xfern and asserts lm_hdata_xfern to the local master to signify Data 4 on l_ad_in[63:32] and the byte enables on lm_cben_in[7:4] are being read and will be transferred to the PCI bus. Since both irdyn and trdyn are asserted, the third data phase is completed on this cycle. |
| 11 | Data 4 | Since the previous data phase was completed, the Core decreases 'lm_burst_cnt'.<br>Since Data 3 on PCI bus were read, the Core transfers Data 4 and their byte enables to ad[31:0] and cben[3:0].<br>The Core keeps irdyn asserted. Because the current transaction is the last, the Core de-asserts framen and req64n to signal the end of the burst, also it de-asserts lm_ldata_xfern and lm_hdata_xfern.<br>Since both irdyn and trdyn are asserted, the fourth data phase is completed on this cycle. |
| 12 | Turn around | The Core relinquishes control of framen, req64n, ad and cben. It de-asserts irdyn, decreases 'lm_burst_cnt' to zero and changes lm_status[3:0] into 'Bus Termination' with lm_termination as 'Normal Termination' because both trdyn and irdyn were asserted last cycle.The target de-asserts devseln and trdyn. |
| 13 | Idle | The Core relinquishes control of irdyn, par and par64. |

## Dual Address Cycle (DAC)

The PCI IP core application logic issues a Dual Address Cycle (DAC) command to inform the PCI IP core of its usage of 64-bit addressing. In response, the Core executes two back-to-back address phases for the target. The PCI IP core issues DAC to handle memory maps that are larger than the 4GB limitation of the 32-bit memory map. 64-bit addressing is not restricted to only 64-bit configurations of the PCI IP core.

Figure 2-21 shows an example of the DAC during a 32-bit read transaction. Table 2-25 gives a clock-by-clock description of the dual address cycle.

***Figure 2-21. Master Dual Address Cycle – Read Transaction***

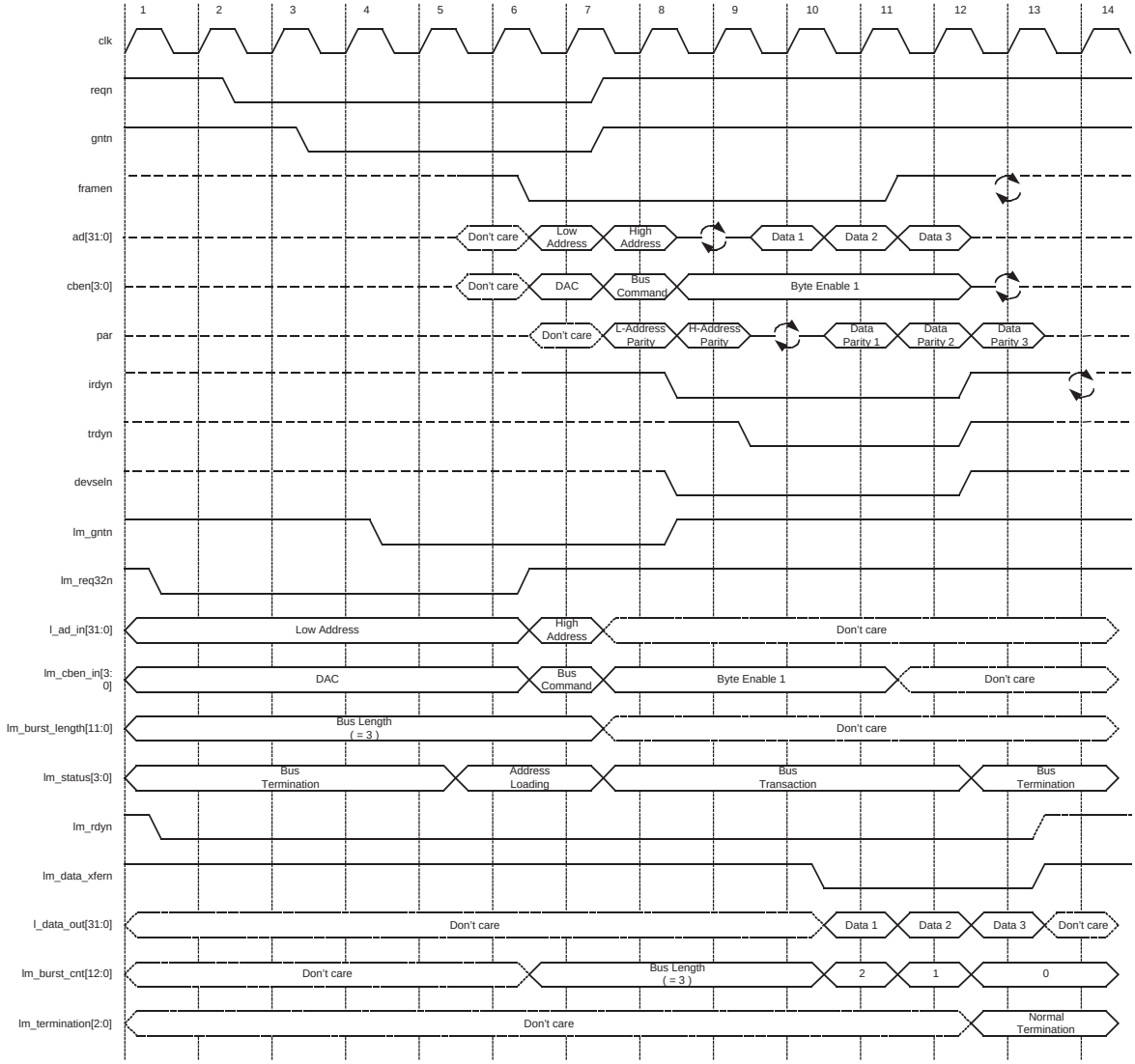



***Table 2-25. 32- Bit Dual Address Cycle – Read Transaction***

| CLK | Phase | Description |
|---|---|---|
| 1 | Idle | The `lm_req32n` signal is asserted by the local master to request for 32-bit data transaction. The local master issues the PCI lower starting address, the bus command (DAC), and the burst length on the same clock cycle to `l_ad_in`, `lm_cben_in`, and `lm_burst_length`, respectively. |
| 2 | Idle | The Core's Local Master Interface detects the asserted `lm_req32n` and asserts `reqn` to request the use of PCI bus. |
| 3 | Idle | `gntn` is asserted to grant the Core access to the PCI bus. Core is now the PCI master. |
| 4 | Idle | Since `gntn` is asserted and the bus is idle, the Core starts the bus transactions. The Core asserts `lm_gntn` to inform the local master that the bus request is granted. |
| 5 | Idle | If both `lm_req32n` and `gntn` were asserted on the previous cycle, `lm_status[3:0]` is changed to 'Address Loading' to indicate the lower starting address, the bus command, and the burst length are being latched. |

*Table 2-25. 32- Bit Dual Address Cycle – Read Transaction (Continued)*

| CLK | Phase | Description |
|---|---|---|
| 6 | Low Address | The Core asserts framen and the local master de-asserts lm_req32n when the previous lm_status[3:0] was 'Address Loading' and if it doesn't want to request another PCI bus transaction.<br>Since lm_status[3:0] was 'Address Loading' on the previous cycle. It also drives the PCI starting address on ad[31:0] and the PCI command (DAC) on cben[3:0]. On the same cycle, it keeps lm_status[3:0] as 'Address Loading' for the Dual Address Cycle.<br>Local master provides higher address on l_ad_in[31:0].<br>lm_burst_cnt gets the value of the burst length. |
| 7 | High Address | The Core de-asserts reqn when framen was asserted but lm_req32n was de-asserted on the previous cycle.<br>The Core keeps framen to start transaction.<br>Since lm_status[3:0] was 'Address Loading' on the previous cycle. It also drives the PCI higher starting address on ad[31:0] and the PCI command on cben[3:0]. On the same cycle, it outputs lm_status[3:0] as 'Bus Transaction' to indicate the beginning of the address/data phases.<br>Because lm_rdyn was asserted on the previous cycle and the next cycle is the first data phase, the local master provides the byte enables on lm_cben_in[3:0]. Asserting lm_rdyn also means the local master is ready to read. If it is not ready to read data, it keep lm_rdyn de-asserted until it is ready. |
| 8 | Turn around | The Core de-asserts lm_gntn to follow gntn.<br>The target asserts to indicate it acknowledges the 32-bit transaction.<br>The Core tri-states the ad[63:0] lines and drives the byte enables (Byte Enable 1). Since lm_rdyn was asserted on the previous cycle, it asserts irdyn to indicate it is ready to read data.<br>Because the Core performs the burst transactions, it keeps framen asserted. |
| 9 | Data 1 | The target asserts trdyn and puts Data 1 on ad[31:0].<br>With lm_rdyn asserted on the previous cycle, the Core keeps irdyn asserted.<br>The Core keeps framen asserted to the target to signify the burst continues.<br>If the local master is ready to read the first DWORD, it keeps lm_rdyn asserted.<br>Since both irdyn and trdyn are asserted, the first data phase is completed on this cycle. |
| 10 | Data 2 | Since the previous data phase was completed, the Core transfers Data 1 on l_data_out[31:0] and decreases the lm_burst_cnt.<br>If both trdyn and lm_rdyn were asserted on the previous cycle, the Core asserts lm_data_xfern to the local master to signify Data 1 are available on l_data_out[31:0]. With lm_data_xfern asserted, the local master can safely read Data 1 and increment the address counter.<br>If the local master keeps lm_rdyn asserted on the previous cycle, the master keeps irdyn asserted.<br>The Core keeps framen asserted to the target to signify the burst continues.<br>If the target is still ready to provide data, it keeps trdyn asserted and drives the next DWORD (Data 2) on ad[31:0].<br>If the local master is ready to read the next DWORD, it keeps lm_rdyn asserted.<br>Since both irdyn and trdyn are asserted, the second data phase is completed on this cycle. |

*Table 2-25. 32- Bit Dual Address Cycle – Read Transaction (Continued)*

| CLK | Phase | Description |
|---|---|---|
| 11 | Data 3 | Since the previous data phase was completed, the Core transfers Data2 on l_data_out[31:0] and decreases the lm_burst_cnt.<br>If both trdyn and lm_rdyn were asserted on the previous cycle, the master asserts lm_data_xfern to the local master to signify Data 2 are available on l_data_out[31:0]. With lm_data_xfern asserted, the local master can safely read Data 2 and increment the address counter.<br>With lm_rdyn asserted on the previous cycle, the Core keeps irdyn asserted.<br>Because the current transaction is the last, the Core de-asserts framen to signal the end of the burst.<br>If the target is still ready to provide data, it keeps trdyn asserted and drives the next DWORD (Data 3) on ad[31:0].<br>If the local master is ready to read the next DWORD, it keeps lm_rdyn asserted.<br>Since both irdyn and trdyn are asserted, the third data phase is completed in this cycle. |
| 11 | Turn around | Since the previous data phase was completed, the Core transfers Data 3 on l_data_out[31:0] and decreases the lm_burst_cnt to zero.<br>The Core relinquishes control of framen and cben. It de-asserts irdyn and changes lm_status[3:0] into 'Bus Termination' with lm_termination as 'Normal Termination' because both trdyn and irdyn were asserted during the last cycle.<br>The target relinquishes control of ad[31:0]. It de-asserts devseln and trdyn.<br>If both trdyn and lm_rdyn were asserted on the previous cycle, the Core asserts lm_data_xfern to the local master to signify Data 3 are available on l_data_out[31:0]. With lm_data_xfern asserted, the local master can safely read Data 3. |
| 12 | Idle | The Core relinquishes control of irdyn and de-asserts lm_data_xfern, and the local master de-asserts lm_rdyn since all of the burst data have been read. |

Figure 2-22 shows an example of the DAC during a 32-bit write transaction. Table 2-26 gives a clock-by-clock description of the dual address cycle.

***Figure 2-22. 32-Bit Master Dual Address Cycle – Write Transaction***

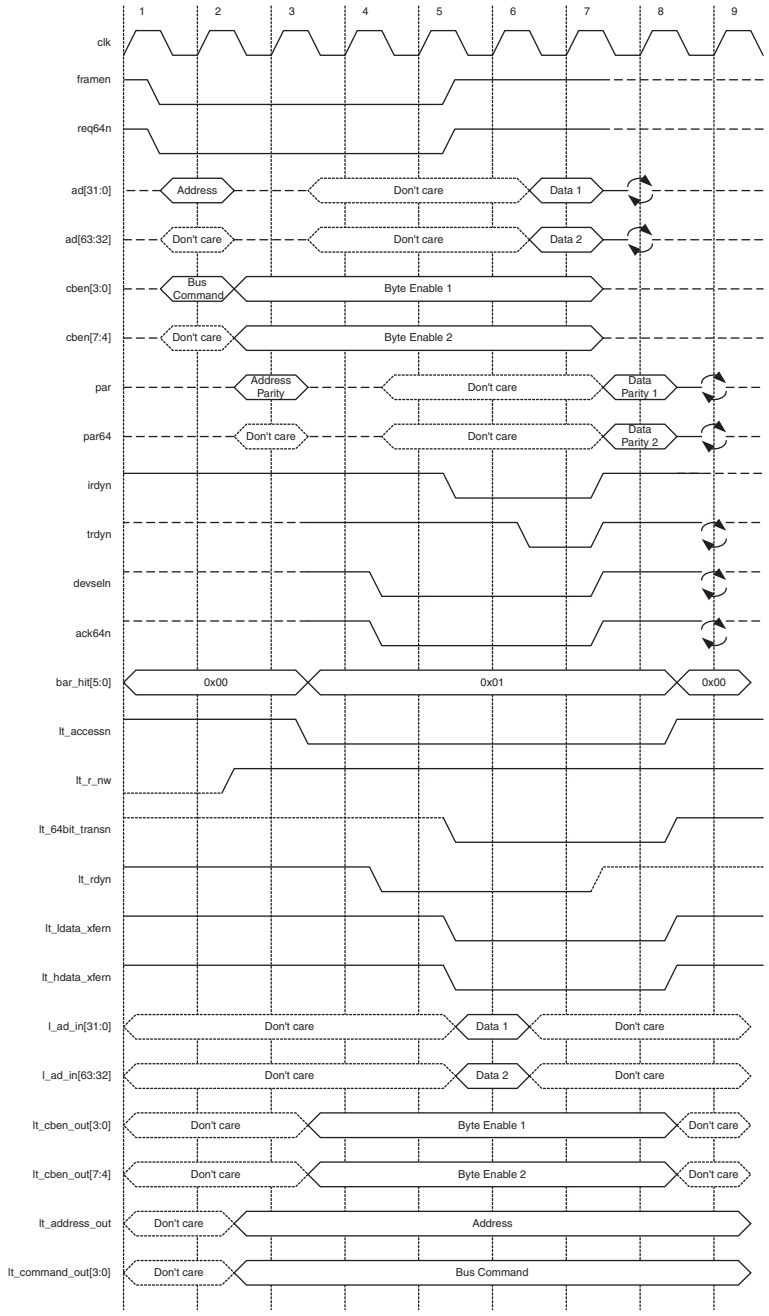



***Table 2-26. 32-bit Master Dual Address Cycle – Write Transaction***

| CLK | Phase | Description |
|---|---|---|
| 1 | Idle | The `lm_req32n` signal is asserted by the local master to request for 32-bit data transaction. The local master issues the PCI starting address, the bus command (DAC), and the burst length during the same clock cycle on `l_ad_in`, `lm_cben_in` and `lm_burst_length`, respectively. |
| 2 | Idle | The Core's Local Master Interface detects the asserted `lm_req32n` and asserts `reqn` to request the use of PCI bus. |
| 3 | Idle | `gntn` is asserted to grant the Core access to the PCI bus. Core is now the PCI master |
| 4 | Idle | Since `gntn` is asserted and the current bus is idle, the Core is going to start the bus transactions. The Core asserts `lm_gntn` to inform the local master that the bus request is granted. |

*Table 2-26. 32-bit Master Dual Address Cycle – Write Transaction (Continued)*

| CLK | Phase | Description |
|---|---|---|
| 5 | Idle | If both lm_req32n and gntn were asserted on the previous cycle, lm_status[3:0] is changed to 'Address Loading' to indicate the starting address, the bus command and the burst length are being latched. |
| 6 | Low Address | The Core asserts framen and the local master de-asserts lm_req32n when the previous lm_status[3:0] was 'Address Loading' and if it doesn't want to request another PCI bus transaction.<br><br>Since lm_status[3:0] was 'Address Loading' on the previous cycle, it also drives the PCI starting address on ad[31:0] and the PCI command (DAC) on cben[3:0]. On the same cycle, it keeps lm_status[3:0] as 'Address Loading' for the Dual Address Cycle.<br><br>Local master provides high 32-bit address on l_ad_in[31:0].<br><br>lm_burst_cnt gets the value of the burst length. |
| 7 | High Address | The Core de-asserts reqn when framen was asserted but lm_req32n was de-asserted on the previous cycle.<br><br>The Core keeps framen to start transaction.<br><br>Since lm_status[3:0] was 'Address Loading' on the previous cycle, it also drives the PCI higher starting address on ad[31:0] and the PCI command on cben[3:0]. On the same cycle, it outputs lm_status[3:0] as 'Bus Transaction' to indicate the beginning of the address/data phases.<br><br>Because lm_rdyn was asserted on the previous cycle and the next cycle is the first data phase, the local master should provide Data 1 on l_ad_in[31:0] and the byte enables on lm_cben_in[3:0]. And the Core asserts lm_data_xfern to the local master to signify these data and byte enables are being read and will be transferred to the PCI bus.<br><br>Asserting lm_rdyn means the local master is ready to write data. If it is not, it should keep lm_rdyn de-asserted until it is ready. |
| 8 | Wait | The Core de-asserts reqn when framen was asserted.<br><br>lm_data_xfern is kept asserted to signify Data 2 on l_ad_in[31:0] and the byte enables on lm_cben_in[3:0] are being read and will be transferred to the PCI bus.<br><br>If the local master is ready to provide the next DWORD, it keeps lm_rdyn asserted. |
| 9 | Data 1 | The Core keeps framen asserted and asserts irdyn. It also de-asserts lm_data_xfern to the local master to signify Data 3 on l_ad_in[31:0] are not read.<br><br>If the local master is ready to provide the next DWORD, it keeps lm_rdyn asserted.<br><br>Because the Core performs the burst transactions, it keeps framen asserted.<br><br>Since both irdyn and trdyn are asserted, the first data phase is completed on this cycle. |
| 10 | Data 2 | Since the previous data phase was completed, the Core decreases lm_burst_cnt.<br><br>Since Data 1 on PCI bus were read by the target, the Core transfers Data 2 and their byte enables to ad[31:0] and cben[3:0].<br><br>With lm_rdyn asserted previous cycle, the Core keeps irdyn asserted.<br><br>Because both lm_rdyn and trdyn were asserted on the previous cycle, the Core asserts lm_data_xfern to signify Data 3 on l_ad_in[31:0] and the byte enables on lm_cben_in[3:0] are being read and will be transferred to the PCI bus.<br><br>Because Data 3 are the last data, the local master de-asserts lm_rdyn.<br><br>Since both irdyn and trdyn are asserted, the second data phase is completed on this cycle. |

*Table 2-26. 32-bit Master Dual Address Cycle – Write Transaction (Continued)*

| CLK | Phase | Description |
|---|---|---|
| 11 | Data 3 | Since the previous data phase was completed, the Core decreases lm_burst_cnt.<br><br>Since Data 2 on the PCI bus were read, the Core transfers Data 3 and their byte enables to ad[31:0] and cben[3:0].<br><br>Because the current transaction is the last, the Core de-asserts framen to signal the end of the burst and it de-asserts lm_data_xfern.<br><br>Since both irdyn and trdyn are asserted, the third data phase is completed on this cycle. |
| 12 | Turn around | The Core relinquishes control of framen, ad and cben. It de-asserts irdyn, decreases lm_burst_cnt to zero and changes lm_status[3:0] into 'Bus Termination" with lm_termination as 'Normal Termination' because both trdyn and irdyn were asserted during the last cycle. The target de-asserts devseln and trdyn. |
| 13 | Idle | The Core relinquishes control of irdyn and par. |

## Fast Back-to-Back Transactions

The PCI IP core, as a master, is capable of executing fast back-to-back transactions if two or more consecutive transactions are required. The fast back-to-back transaction consists of two or more complete PCI transactions without an idle state between them. To execute fast back-to-back transaction with the PCI IP core, lm_req32n or lm_req64n must be asserted once lm_status changes to the 'Address Loading' state. Otherwise, the assertion will not be recognized and the next transaction will be treated as a basic transaction having the 'Idle State' on the PCI bus. An effective way for handling fast back-to-back transfers is to keep lm_req32n or lm_req64n asserted until required data has been transferred.

For fast back-to-back transaction, the previous transaction must be a write transaction.

Figure 2-23 and Table 2-27 illustrate a 64-bit, fast back-to-back write transaction. The figure illustrates how the PCI interface correlates to the Local Master Interface. The table explains each event in the figure with a clock-by-clock description.

***Figure 2-23. 64-bit Master Fast Back-to-Back Transaction***

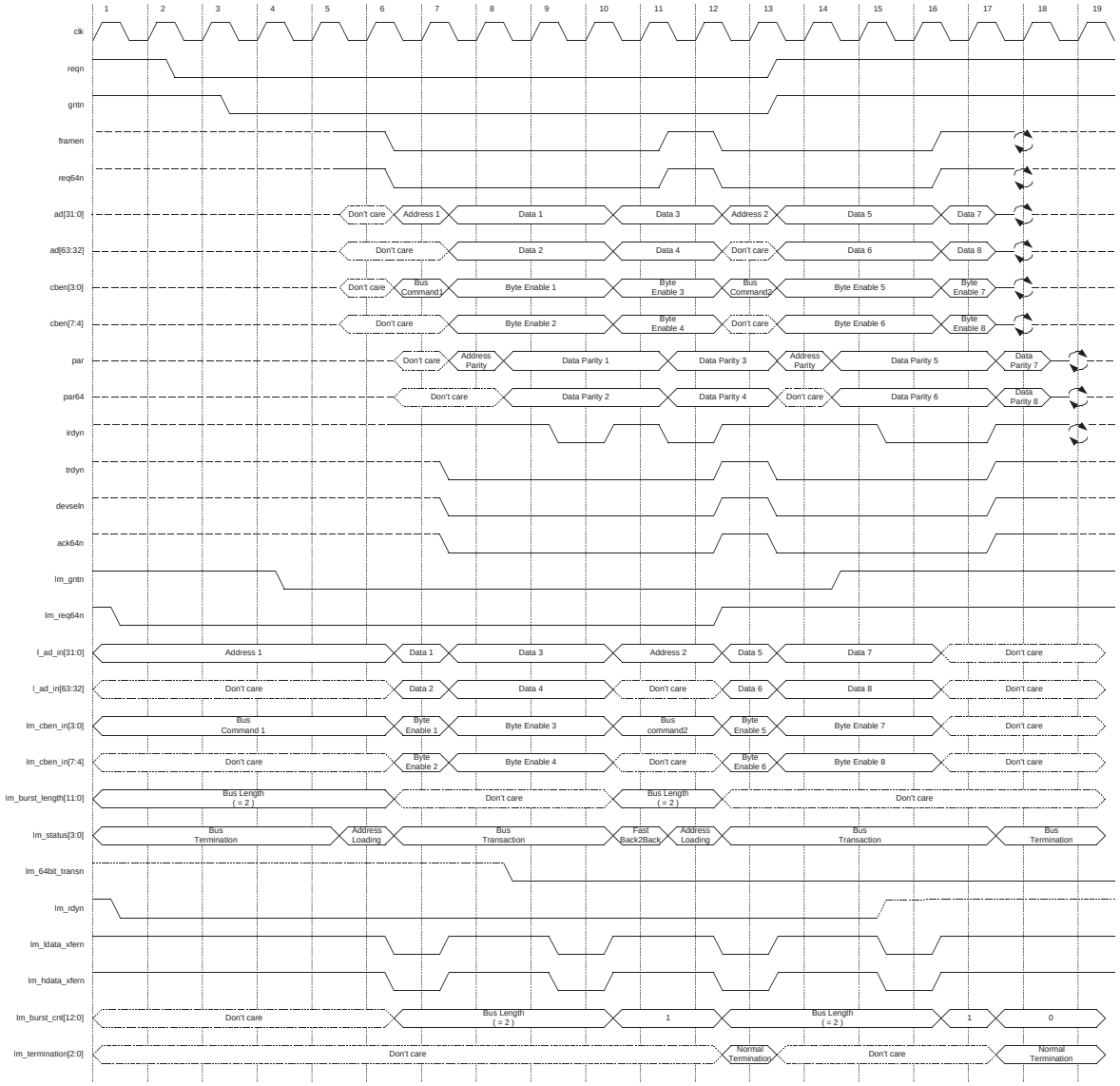


*Table 2-27. Fast Back-to-Back Transaction*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Idle | The local master asserts lm_req64n to request for 64-bit data transaction. It also issues the PCI starting address, the bus command and the burst length on l_ad_in, lm_cben_in and lm_burst_length respectively during the same clock cycle. |
| 2 | Idle | The Core's Local Master Interface detects the asserted lm_req64n and asserts reqn to request the use of PCI bus. |
| 3 | Idle | gntn is asserted to grant the Core access to the PCI bus. Core is now PCI master |
| 4 | Idle | Since gntn is asserted and the current bus is idle, the Core is going to start the bus transactions. The Core asserts lm_gntn to inform the local master that the bus request is granted. |
| 5 | Idle | If both lm_req64n and lm_gntn were asserted on the previous cycle, lm_status[3:0] is changed to 'Address Loading' to indicate the starting address, the bus command and the burst length are being latched. |
| 6 | Address | The local master keeps lm_req64n because it wants to request another PCI bus transaction for fast back-to-back transaction.<br>The Core asserts framen and req64n to initiate the 64-bit write transaction when gntn was asserted and lm_status[3:0] was 'Address Loading' on the previous cycle. It also drives the PCI starting address on ad[31:0] and the PCI command on cben[3:0]. On the same cycle, it outputs lm_status[3:0] as 'Bus Transaction' to indicate the beginning of the address/data phases.<br>lm_burst_cnt gets the value of the burst length.<br>Because lm_rdyn was asserted on the previous cycle and the next cycle is the first data phase, the local master provides Data 1 and Data 2 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0]. And the Core asserts lm_ldata_xfern and lm_hdata_xfern to the local master to signify these data and byte enables are being read and will be transferred to the PCI bus.<br>Asserting lm_rdyn means the local master is ready to write data. If it is not, it keeps lm_rdyn de-asserted until it is ready. |
| 7 | Wait | The Core keeps reqn when framen and lm_req64n were asserted to indicate fast back-to-back transaction.<br>If the target completes the fast decode and is ready to receive 64-bit data, it asserts devseln, ack64n and trdyn.<br>With lm_ldata_xfern and lm_hdata_xfern asserted on the previous cycle that was the address phase, the local master should increment the address counter while the Core transfers Data 1 and Data 2 and their byte enables to ad[63:0] and cben[7:0].<br>With lm_rdyn asserted on the previous cycle, the local master provides Data 3 and Data 4 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0].<br>Because this is the first write data phase and devseln is just asserted, the Core keeps framen asserted and irdyn de-asserted to judge 64-bit or 32-bit transaction. It also de-asserts lm_ldata_xfern and lm_hdata_xfern to the local master to signify Data 3 and Data 4 on l_ad_in[63:0] are not read.<br>If the local master is ready to provide the next QWORD, it keeps lm_rdyn asserted.<br>Since irdyn is not asserted, the first data phase is not completed. |
| 8 | Wait | Since lm_ldata_xfern and lm_hdata_xfern were not asserted on the previous cycle, the local master keeps Data 3 and Data 4 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0].<br>Because the Core needs one more cycle to decide 64-bit or 32-bit transaction, it keeps framen asserted and irdyn de-asserted. It also keeps lm_ldata_xfern and lm_hdata_xfern de-asserted to the local master to signify Data 3 and Data 4 on l_ad_in[63:0] are not read.<br>The Core asserts lm_64bit_transn to indicate the current data transaction is 64-bit wide. It de-asserts lm_gntn to follow gntn.<br>If the local master is ready to provide the next QWORD, it keeps lm_rdyn asserted.<br>Since irdyn is not asserted, the first data phase is not completed. |

*Table 2-27. Fast Back-to-Back Transaction (Continued)*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 9 | Data 1 and 2 | Since lm_ldata_xfern and lm_hdata_xfern were not asserted on the previous cycle, the local master keeps Data 3 and Data 4 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0].<br>With both devseln and lm_rdyn asserted previous cycle, the Core asserts irdyn, and it prepares for the 64-bit write burst. So it asserts lm_ldata_xfern and lm_hdata_xfern to the local master to signify Data 3 and Data 4 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0] are being read and will be transferred to the PCI bus.<br>The Core keeps framen asserted and irdyn de-asserted. It also keeps lm_ldata_xfern and lm_hdata_xfern de-asserted to the local master to signify Data 3 and Data 4 on l_ad_in[63:0] are not read.<br>If the local master is ready to provide the next QWORD, it keeps lm_rdyn asserted. Because the Core performs the burst transactions, it keeps framen asserted.<br>Since both irdyn and trdyn are asserted, the first data phase is completed on this cycle. |
| 10 | Wait | Since the previous data phase was completed, the Core decreases 'lm_burst_cnt'.<br>At the last data phase of first transaction, the Core inserts a wait cycle to prepare next transaction. So it de-asserts irdyn and changes lm_status[3:0] from 'Bus Transaction' to 'Fast Back2Back', and it also de-asserts lm_ldata_xfern and lm_data_xfern.<br>With lm_ldata_xfern and lm_hdata_xfern asserted on the previous cycle, the local master should put next transaction's address, bus command and burst length on l_ad_in[31:0], lm_cbe_in[3:0] and lm_burst_length[11:0] respectively.<br>Since Data 1 and Data 2 on PCI bus were read by the target, the Core transfers Data 3 and Data 4 and their byte enables to ad[63:0] and cben[7:0]. |
| 11 | Data 3 and 4 | The Core de-asserts framen and req64n, asserts irdyn to signal Data 3 and 4 transferred.<br>Since both irdyn and trdyn are asserted, the second data phase is completed on this cycle.<br>If both lm_req64n and lm_gntn were asserted on the previous cycle, lm_status[3:0] is changed to 'Address Loading' to indicate the starting address, the bus command and the burst length are being latched. |
| 12 | Address | The local master de-asserts lm_req64n when the previous lm_status[3:0] was 'Address Loading'.<br>The Core asserts framen and req64n to initiate the second 64-bit write transaction when gntn was asserted and lm_status[3:0] was 'Address Loading' on the previous cycle. It also drives the PCI starting address on ad[31:0] and the PCI command on cben[3:0]. On the same cycle, it outputs lm_status[3:0] as 'Bus Transaction' to indicate the beginning of the address/data phases.<br>lm_burst_cnt gets the value of the burst length.<br>Because lm_rdyn was asserted on the previous cycle and the next cycle is the first data phase, the local master provides Data 5 and Data 6 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0]. And the Core asserts lm_ldata_xfern and lm_hdata_xfern to the local master to signify these data and byte enables are being read and will be transferred to the PCI bus.<br>Asserting lm_rdyn means the local master is ready to write data. If it is not, it keeps lm_rdyn de-asserted until it is ready. |

*Table 2-27. Fast Back-to-Back Transaction (Continued)*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 13 | Wait | The Core de-asserts reqn when framen was asserted but lm_req64n was de-asserted on the previous cycle.<br>If the target completes the fast decode and is ready to receive 64-bit data, it asserts devseln, ack64n and trdyn.<br>With lm_ldata_xfern and lm_hdata_xfern asserted on the previous cycle that was the address phase, the local master should increment the address counter while the Core transfers Data 5 and Data 6 and their byte enables to ad[63:0] and cben[7:0].<br>With lm_rdyn asserted on the previous cycle, the local master provides Data 7 and Data 8 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0].<br>Because this is the first write data phase and devseln is just asserted, the Core keeps framen asserted and irdyn de-asserted to judge 64-bit or 32-bit transaction. It also de-asserts lm_ldata_xfern and lm_hdata_xfern to the local master to signify Data 7 and Data 8 on l_ad_in[63:0] are not read.<br>If the local master is ready to provide the next QWORD, it keeps lm_rdyn asserted.<br>Since irdyn is not asserted, the first data phase is not completed. |
| 14 | Wait | Since lm_ldata_xfern and lm_hdata_xfern were not asserted on the previous cycle, the local master keeps Data 7 and Data 8 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0].<br>Because the Core needs one more cycle to decide 64-bit or 32-bit transaction, it keeps framen asserted and irdyn de-asserted. It also keeps lm_ldata_xfern and lm_hdata_xfern de-asserted to the local master to signify Data 7 and Data 8 on l_ad_in[63:0] are not read.<br>The Core asserts lm_64bit_transn to indicate the current data transaction is 64-bit wide. It de-asserts lm_gntn to follow gntn.<br>If the local master is ready to provide the next QWORD, it keeps lm_rdyn asserted.<br>Since irdyn is not asserted, the first data phase is not completed. |
| 15 | Data 5 and 6 | Since lm_ldata_xfern and lm_hdata_xfern were not asserted on the previous cycle, the local master keeps Data 7 and Data 8 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0].<br>With both devseln and lm_rdyn asserted previous cycle, the Core asserts irdyn, and it prepares for the 64-bit write burst. So it asserts lm_ldata_xfern and lm_hdata_xfern to the local master to signify Data 7 and Data 8 on l_ad_in[63:0] and the byte enables on lm_cben_in[7:0] are being read and will be transferred to the PCI bus.<br>The Core keeps framen asserted and asserts irdyn. It also keeps lm_ldata_xfern and lm_hdata_xfern de-asserted to the local master to signify Data 3 and Data 4 on l_ad_in[63:0] are not read.<br>If the local master is ready to provide the next QWORD, it keeps lm_rdyn asserted.<br>Because the Core performs the burst transactions, it keeps framen asserted.<br>Since both irdyn and trdyn are asserted, the first data phase is completed on this cycle. |
| 16 | Data 7 and 8 | Since the previous data phase was completed, the Core decreases 'lm_burst_cnt'.<br>Since Data 5 and Data 6 on PCI bus were read, the Core transfers Data 7 and Data 8 and their byte enables to ad[63:0] and [7:0].<br>With lm_rdyn asserted previous cycle, the Core keeps irdyn asserted.<br>Because the current transaction is the last, the Core de-asserts framen and req64n to signal the end of the burst, also it de-asserts lm_ldata_xfern and lm_hdata_xfern.<br>Since both irdyn and trdyn are asserted, the second data phase is completed on this cycle. |
| 17 | Turn around | The Core relinquishes control of framen, req64n, ad and cben. It de-asserts irdyn, decreases 'lm_burst_cnt' to zero and changes lm_status[3:0] into 'Bus Termination' with lm_termination as 'Normal Termination' because both trdyn and irdyn were asserted last cycle.The target de-asserts devseln, ack64n and trdyn. |
| 18 | Idle | The Core relinquishes control of irdyn, par and par64. |

## Master and Target Termination

The signal lm_termination[2:0] indicates the different types of Master-initiated terminations. In addition, the state of the target's response when the master executes the transaction is made available for the user's master application. This enables the master application to complete, terminate or refer the transaction to software via interrupt.

The master-initiated early termination commands include timeout and Master Abort. When the master's gntn line is de-asserted and its internal latency timer is expired, the master ends the current transaction. When it doesn't detect the assertion of devseln within the required period after it asserts framen, the master terminates the current transaction. This is called Master Abort termination.

The back-end application monitors and controls early termination of PCI transactions by asserting lm_abortn. Any lm_abortn assertion is ignored during 'Address Loading' and the first clock cycle of 'Bus Transaction'. If lm_abortn is asserted after first clock cycle of 'Bus Transaction', the transaction is terminated at next data phase. When next clock cycle is a wait cycle, the transaction is not terminated until one data phase is completed, except when the target aborts the transaction.

A summary of the four types of target-initiated termination commands are described in Table 2-28.

***Table 2-28. Master Initiated Termination Summary***

| Lm_termination[2:0] | Name | Description |
|---|---|---|
| 000 | Normal termination | Normal Termination takes place. |
| 001 | Timeout termination | The cycle timed out. |
| 010 | No target response termination | Also known as Master Abort. The Master terminates the transaction because devseln was not asserted during the expected time. |
| 011 | Target abort termination | The Target issues an abort termination |
| 100 | Retry termination | The target of the transaction is not ready for the transaction. The Master issues a retry. |
| 101 | Disconnect data termination | The target device is terminating the burst transaction. |
| 110 | Grant abort termination | A Grant termination has occurred. |
| 111 | Local master termination | The Local Interface cannot complete the transaction. |

## Basic PCI Target Read and Write Transactions

Read and write transactions to memory and I/O space are used to transfer data on the PCI bus. The basic read and write transactions use the following PCI commands:

- I/O Read
- I/O Write
- Memory Read
- Memory Write
- Configuration Read
- Configuration Write

To make the integration of the PCI IP core as simple as possible, the basic transactions are described based on different bus configurations supported with this PCI IP core. Although the fundamentals of the basic transactions are the same, different bus configurations require slightly different local bus signaling. The PCI and local bus configurations do not affect configuration access because configuration accesses require no local bus intervention. Refer to the following sections for more information on the basic bus transactions with specific PCI IP core configurations:

- 32-Bit PCI Target with a 32-Bit Local Bus
- 64-Bit PCI Target with a 64-Bit Local Bus
- 32-Bit PCI Target with a 64-Bit Local Bus

Refer to the advanced bus transactions in the Advanced Target Transactions section for more information on proper wait state insertion and early termination of bus transactions by the PCI IP core.

*Design Hint: Using the base address registers as memory space and not I/O space in a device is highly recommended. In a legacy PC environment the I/O space is extremely limited and fragmented due to legacy issues.*

## 32-bit PCI Target with a 32-bit Local Bus Memory Transactions

This section discusses read and write transactions for the PCI IP core, operating as a Target, configured with a 32-bit PCI bus and a 32-bit local bus. Because 32-bit I/O and memory transactions are alike, they are discussed together.

Figure 2-24 illustrates an example of a basic 32-bit read transaction. Table 2-29 gives a clock-by-clock description of the basic 32-bit transaction in Figure 2-24. On a read transaction it is important to realize and understand the latency between the PCI and Local Target Interface. For instance, two clock cycles of latency exist between `lt_rdyn` and `trdyn`.

***Figure 2-24. 32-bit Target Single Read Transaction with a 32-bit Local Interface***

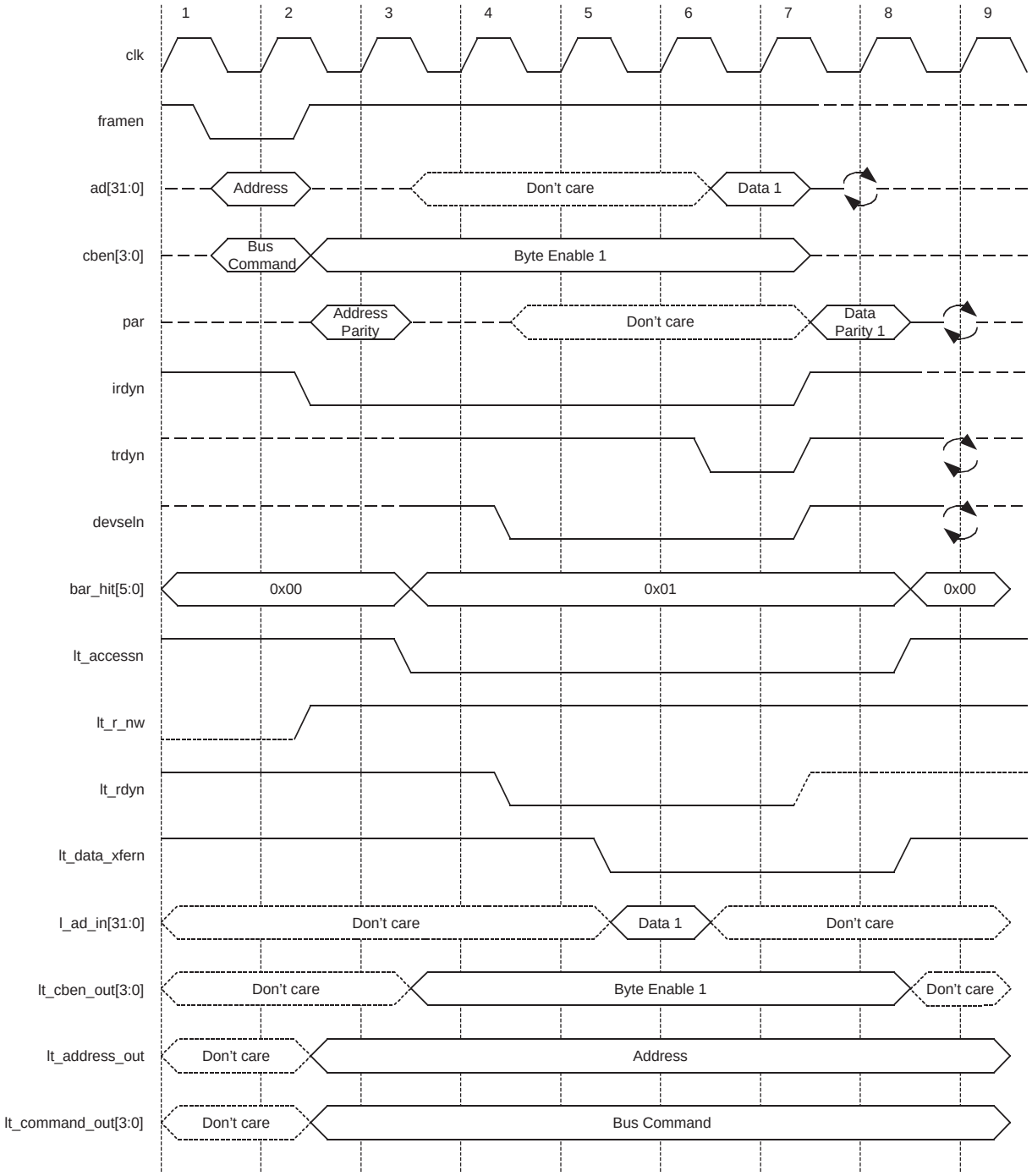


*Table 2-29. 32-bit Target Single Read Transaction with a 32-Bit Local Interface*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Address | The master asserts framen and drives ad[31:0] and cben[3:0]. |
| 2 | Turn around | The master tri-states the ad[31:0] lines and drives the byte enables cben[3:0]. If the master is ready to receive single data, it asserts irdyn and de-asserts framen to indicate single data phase transaction.<br>The PCI IP core starts to decode the address and command It also registers and drives the lt_address_out lt_command_out to the back end. |
| 3 | Wait | If there is an address match, the Core drives the bar_hit signals to the back-end. It also asserts lt_accessn. The back-end can use the bar_hit signals as a chip select. |
| 4 | Wait | With the device select timing set to Slow, the Core asserts devseln one clock after bar_hit. If the back-end is ready to put data out on the next cycle, it can assert lt_rdyn. The local target can insert wait states by not asserting lt_rdyn. |
| 5 | Wait | The Core's Local Target Interface asserts lt_data_xfern since lt_rdyn was asserted the previous cycle. The back end drives the first DWORD (Data 1) on l_ad_in[31:0]. The Core asserts lt_data_xfern to indicate that data from the back-end logic must be valid at this time in order for the master to read the data correctly. |
| 6 | Data 1 | With lt_rdyn asserted during the previous two cycles the Core asserts trdyn and puts Data 1 on ad[31:0]. |
| 7 | Turn around | The master relinquishes control of framen and cben[3:0] and de-asserts irdyn since the data transfer only requires one data phase.<br>The Core relinquishes control of ad[31:0] and de-asserts both devseln and trdyn. |
| 8 | Idle | The Core relinquishes control of devseln and trdyn.The Core also clears bar_hit, to signal to the back end that the transaction is complete, and de-asserts lt_data_xfern. |

Figure 2-25 illustrates an example of a basic 32-bit write transaction to the PCI IP core operating as a Target. Table 2-30 gives a clock-by-clock description of the 32-bit write transaction.

***Figure 2-25. 32-bit Target Single Write Transaction with a 32-bit Local Interface***

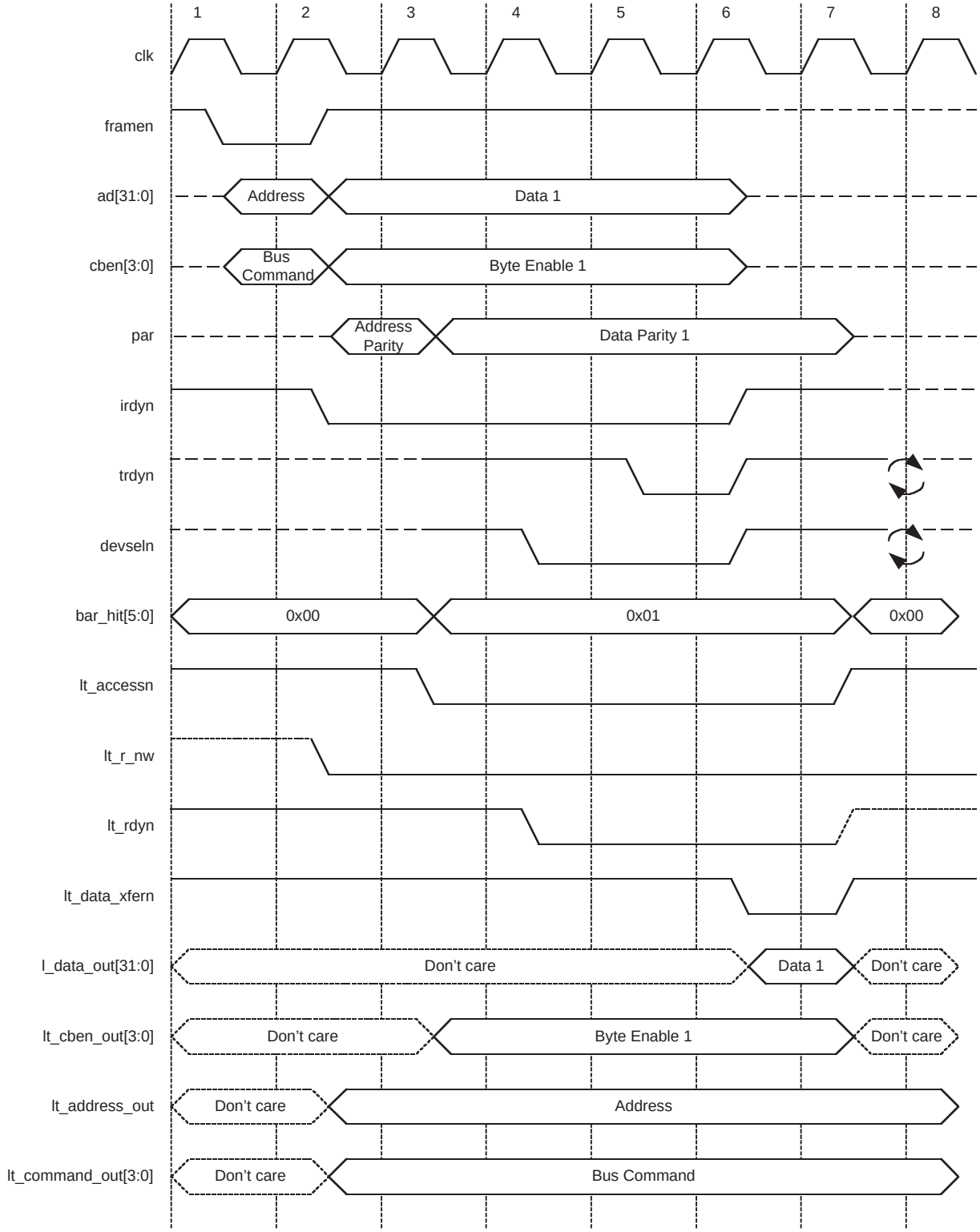


*Table 2-30. 32-bit Target Single Write Transaction with a 32-bit Local Interface*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Address | The master asserts framen and drives ad[31:0] and cben[3:0]. |
| 2 | Wait | The master drives the byte enable (Byte Enable 1).The master asserts irdyn, indicating that it is ready to write the data, and de-asserts framen. To indicate a single data phase transaction, it drives DWORD (Data 1) on ad[31:0].The Core starts to decode the address and command and drives the lt_address_out to the back-end. |
| 3 | Wait | If an address match is present, the Core drives the bar_hit signals to the back-end. The back-end can use bar_hit as a chip select. |
| 4 | Wait | With the DEVSEL_TIMING set to slow, the Core asserts devseln one clock after bar_hit. If the back-end will be ready to write data in two cycles, it can assert lt_rdyn. |
| 5 | Data 1 | trdyn is asserted by the Core since lt_rdyn was asserted by the application logic during the previous cycle. |
| 6 | Turn around | If both irdyn and trdyn were asserted on the previous cycle, the master relinquishes control of framen, ad[31:0] and cben[3:0]. The master also de-asserts irdyn since only one data phase is required. The Core asserts lt_data_xfern to indicate that the valid PCI data is available for writing. |
| 7 | Idle | The Core relinquishes control of devseln and trdyn.The target clears bar_hit to signal to the back-end that the transaction is complete. It also de-asserts lt_data_xfern. |

## 64-Bit PCI Target with a 64-Bit Local Bus

This section discusses read and write transactions for a PCI IP core, operating as a target, configured with a 64-bit PCI bus and a 64-bit local bus. All 64-bit PCI devices are required by the PCI Specification to handle both 64-bit and 32-bit applications. The 32-bit transactions, described in the 32-Bit PCI Target with a 32-Bit Local Bus Memory Transactions section, are similar to a 32-bit transaction for the 64-bit PCI IP core configuration with the exception that when the 64-bit Core responds to a 32-bit transaction the upper 32 bits of the data bus should be ignored.

The 64-bit memory read transaction is similar to the 32-bit target read transaction with additional PCI signals required for 64-bit signaling. Figure 2-27 and Table 2-31 illustrate a basic 64-bit read transaction.

***Figure 2-26. 64-Bit Target Single Read Transaction with a 64-Bit Local Interface***

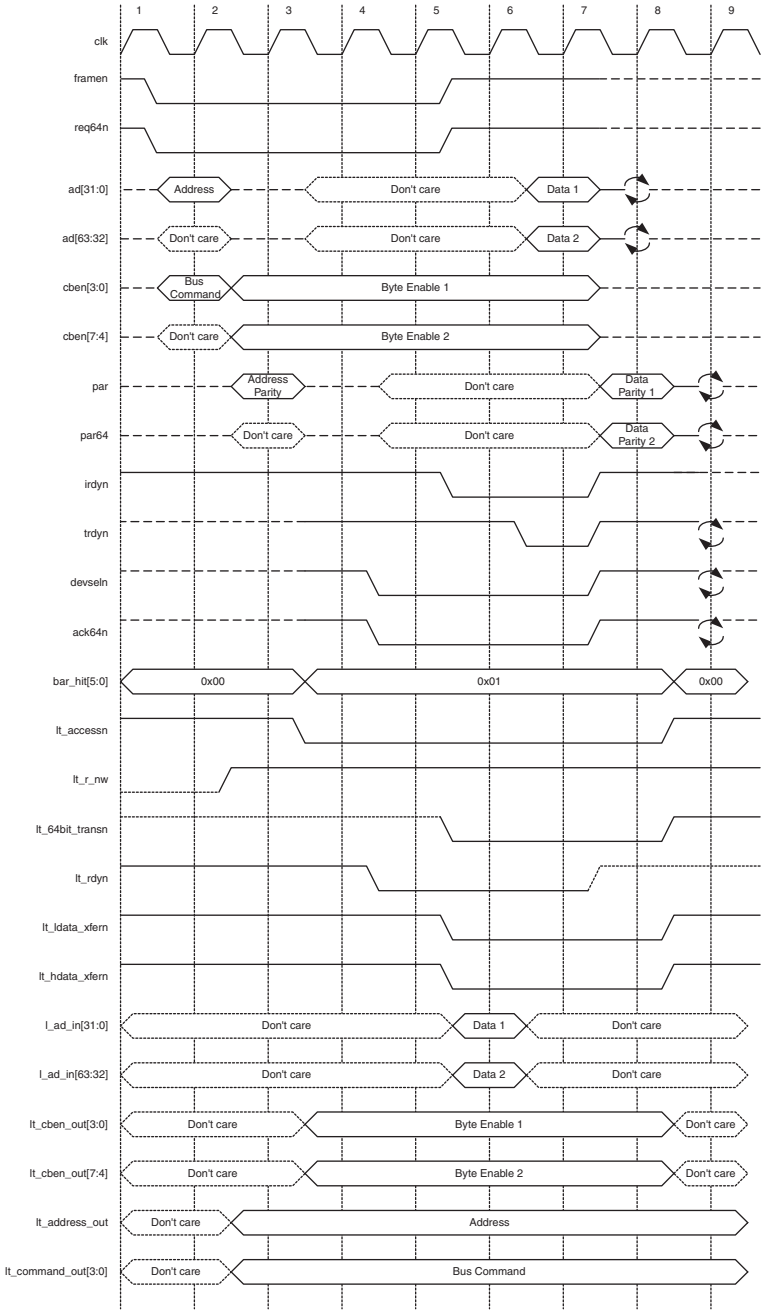


*Table 2-31. 64-bit Target Single Read Transaction with a 64-bit Local Interface*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Address | The master asserts framen, req64n and drives ad[31:0] and cben[3:0]. |
| 2 | Turn around | The master tri-states the ad[63:0] lines and drives the first byte enables (Byte Enable 1 and 2). If the master is ready to receive data, it asserts irdyn.<br>The Core starts to decode the address and command. The Core drives the lt_address_out to the back-end. The lt_64bit_transn signal is driven low to signal the back-end that a 64-bit transaction has been requested. |
| 3 | Wait | If an address match is present, the Core drives the bar_hit signals to the back-end. The back-end can use the bar_hit as a chip select. |
| 4 | Wait | If the DEVSEL_TIMING is set to slow, the Core asserts devseln one clock after bar_hit. The ack64n signal is also asserted to acknowledge the 64-bit request. If the back-end is ready to put data out on the next cycle, it can assert lt_rdyn. |
| 5 | Wait | The local target asserts lt_hdata_xfern and lt_ldata_xfern since lt_rdyn was asserted the previous cycle. The back-end drives the first QWORD (Data 1) on l_ad_in[63:0].<br>With lt_rdyn asserted during the previous two cycles, the burst cycle starts the Core asserts trdyn and puts Data 1 on ad[31:0]. |
| 6 | Data 1 | With lt_rdyn asserted previous two cycles, the burst cycle starts. The Core asserts trdyn and puts Data 1 on ad[63:0].<br>If both irdyn and lt_rdyn are asserted on the previous cycle, the Core keeps lt_hdata_xfern and lt_ldata_xfern asserted to the back-end. The back-end can increment the address counter and put the next QWORD (Don't care) on l_ad_in[63:0]. |
| 9 | Turn around | The master relinquishes control of framen, ack64n and cben[7:0]. It de-asserts irdyn if both trdyn and irdyn were asserted last cycle.<br>The Core relinquishes control of ad[63:0]. It de-asserts both devseln and trdyn if both trdyn and irdyn were asserted during the last cycle. The Core also clears bar_hit to signal to the back-end that the transaction is complete. The Core de-asserts lt_hdata_xfern and lt_ldata_xfern. |
| 10 | Idle | |

The 64-bit memory write transaction is similar to the 32-bit target write transaction with additional PCI signals required for 64-bit signaling. Figure 2-27 and Table 2-31 show a basic 64-bit write transaction.

***Figure 2-27. 64-bit Target Single Write Transaction with a 64-bit Local Interface***

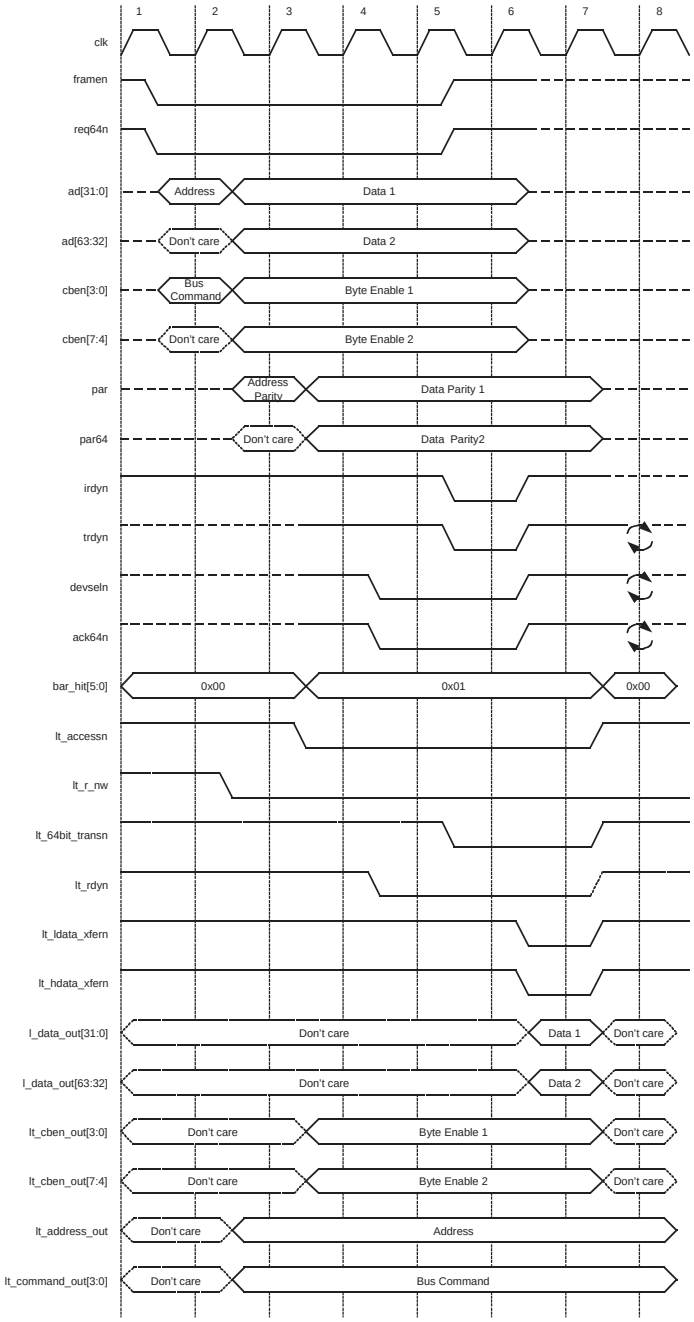


*Table 2-32. 64-bit Target Single Write Transaction with a 64-bit Local Interface*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Address | The master asserts framen and req64n and drives ad[63:0] and cben[3:0]. |
| 2 | Wait | The master drives the first byte enables (Byte Enable 1). If the master is ready to write data, it asserts irdyn and drives the first QWORD (Data 1) on ad[63:0].The Core starts to decode the address and command. The Core drives the lt_address_out to the back-end. The lt_64bit_transn signal is driven low to signal the back-end that a 64-bit transaction has been requested. |
| 3 | Wait | If there is an address match, the Core drives the bar_hit signals to the back-end. The back-end can use the bar_hit as a chip select. |
| 4 | Wait | If the DEVSEL_TIMING is set to slow, the Core asserts devseln on clock after bar_hit. The ack64n signal is also asserted to acknowledge the 64-bit request. If the back-end will be ready to put data out on the next cycle, it can assert lt_rdyn. |
| 5 | Data 1 | trdyn is asserted since lt_rdyn was asserted during the previous cycle. |
| 6 | Turn around | If both irdyn and trdyn are asserted during the previous cycle, the master relinquishes control of framen, req64n, ad[63:0] and cben[7:0]. It also de-asserts irdyn if both trdyn and irdyn were asserted last cycle. |
| 7 | Idle | The Core signals to the back-end that the transaction is complete by clearing bar_hit. It also de-asserts lt_hdata_xfern and lt_ldata_xfern. |
| 8 | Idle | |

## 32-Bit PCI Target with a 64-Bit Local Bus

This section discusses read and write transactions for a PCI IP core, operating as a target, configured with a 32-bit PCI bus and a 64-bit local bus. The 32-bit PCI transactions, described in the 32-Bit PCI Target with a 32-Bit Local Bus Memory Transactions section, look similar to the transaction; however; the data is handled differently at the Local Target Interface.

In order to present a full 64 bits of data to the Local Target Interface, two PCI data phase are required. Like retrieving 64 bits of data from the Local Target Interface, two PCI data phases are required

The Local Target Interface control latches the complete QWORD and routes the proper DWORD to the PCI data bus. The lt_ldata_xfern and lt_hdata_xfern signals specify which DWORD is transferred.

If the starting address is QWORD aligned, the first DWORD is assumed to be the lower DWORD of a QWORD and is placed on the PCI data bus. Otherwise, the upper DWORD is placed on the PCI data bus.

The 64-bit memory write transaction is similar to the 32-bit target write transaction with additional PCI signals required for 64-bit signaling. Figure 2-28 and Table 2-33 illustrate a basic 64-bit write transaction.

***Figure 2-28. 32-bit Target Single Read Transaction with a 64-bit Local Interface***

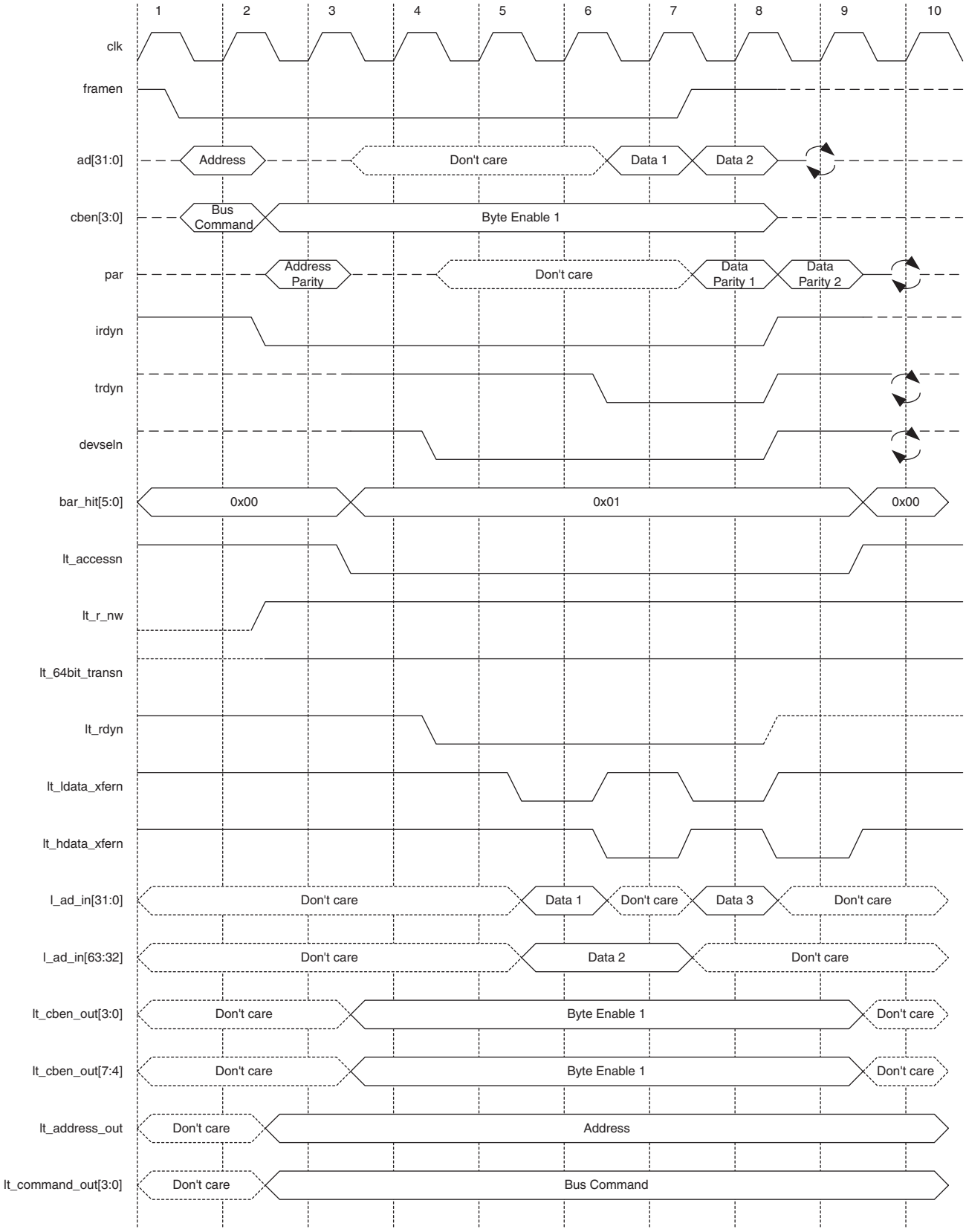


*Table 2-33. 32-bit Target Single Read Transaction with a 64-bit Local Interface*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Address | The master asserts framen and drives ad[31:0] and cben[3:0]. |
| 2 | Turn around | The master tri-states the ad lines and drives the first byte enables (Byte Enable 1). If the master is ready to receive data, it asserts irdyn.<br>The Core starts to decode the address and command. The Core drives the lt_address_out to the back-end. |
| 3 | Wait | If there is an address match, the Core drives the bar_hit signals to the back-end. The back-end can use the bar_hit as a chip select. |
| 4 | Wait | If the DEVSEL_TIMING is set to slow, the Core asserts devseln on clock after bar_hit. If the back-end will be ready to put data out on the next cycle, it can assert lt_rdyn. |
| 5 | Wait | Quad Word Aligned<br>With lt_rdyn asserted on the previous cycle, the local interface asserts lt_ldata_xfern. The back-end drives the first DWORD (Data 1) on l_ad_in[31:0].<br>Double Word Aligned<br>With lt_rdyn asserted on the previous cycle, the local interface asserts lt_hdata_xfern. The back-end drives the first DWORD (Data 1) on l_ad_in[63:32]. |
| 6 | Data 1 | With trdyn and irdyn asserted Data 1 is placed on ad[31:0]<br>Quad Word Aligned<br>The Core de-asserts lt_ldata_xfern. If irdyn is asserted on the previous cycle, the Core asserts lt_hdata_xfern to the back-end. With lt_hdata_xfern de-asserted the previous cycle, the back-end does not increment the address counter and holds the QWORD (Data 2) on l_ad_in[63:0].<br>Double Word Aligned<br>With lt_rdyn asserted during the previous two cycles, the burst cycle starts, so the Core asserts trdyn and puts Data 1 on ad since the initial address is DWORD aligned. Notice that the lower DWORD from l_ad_in[31:0] is discarded.<br>The Core de-asserts lt_hdata_xfern. If both irdyn and lt_rdyn are asserted on the previous cycle, the Core asserts lt_ldata_xfern to the back-end. With lt_hdata_xfern asserted the previous cycle, the back-end can increment the address counter and put the next QWORD (Data 2) on l_ad_in[31:0]. |
| 7 | Data 2 | The Core keeps trdyn asserted and puts Data 2 on ad[31:0]. |
| 8 | Turn around | The master relinquishes control of framen and cben[3:0]. It de-asserts irdyn if both trdyn and irdyn were asserted last cycle.The Core relinquishes control of ad[31:0]. It de-asserts both devseln and trdyn if both trdyn and irdyn were asserted last cycle. |
| 9 | Idle | The Core relinquishes control of devseln and trdyn.The Core also signals to the back-end that the transaction is complete by clearing bar_hit. The Core de-asserts lt_data_xfern. |

The 64-bit memory write transaction is very similar to the 32-bit target write transaction with additional PCI signals required for 64-bit signaling. Figure 2-29 and Table 2-34 show a basic 64-bit write transaction.

***Figure 2-29. 32-bit Target Single Write Transaction with a 64-bit Local Interface***

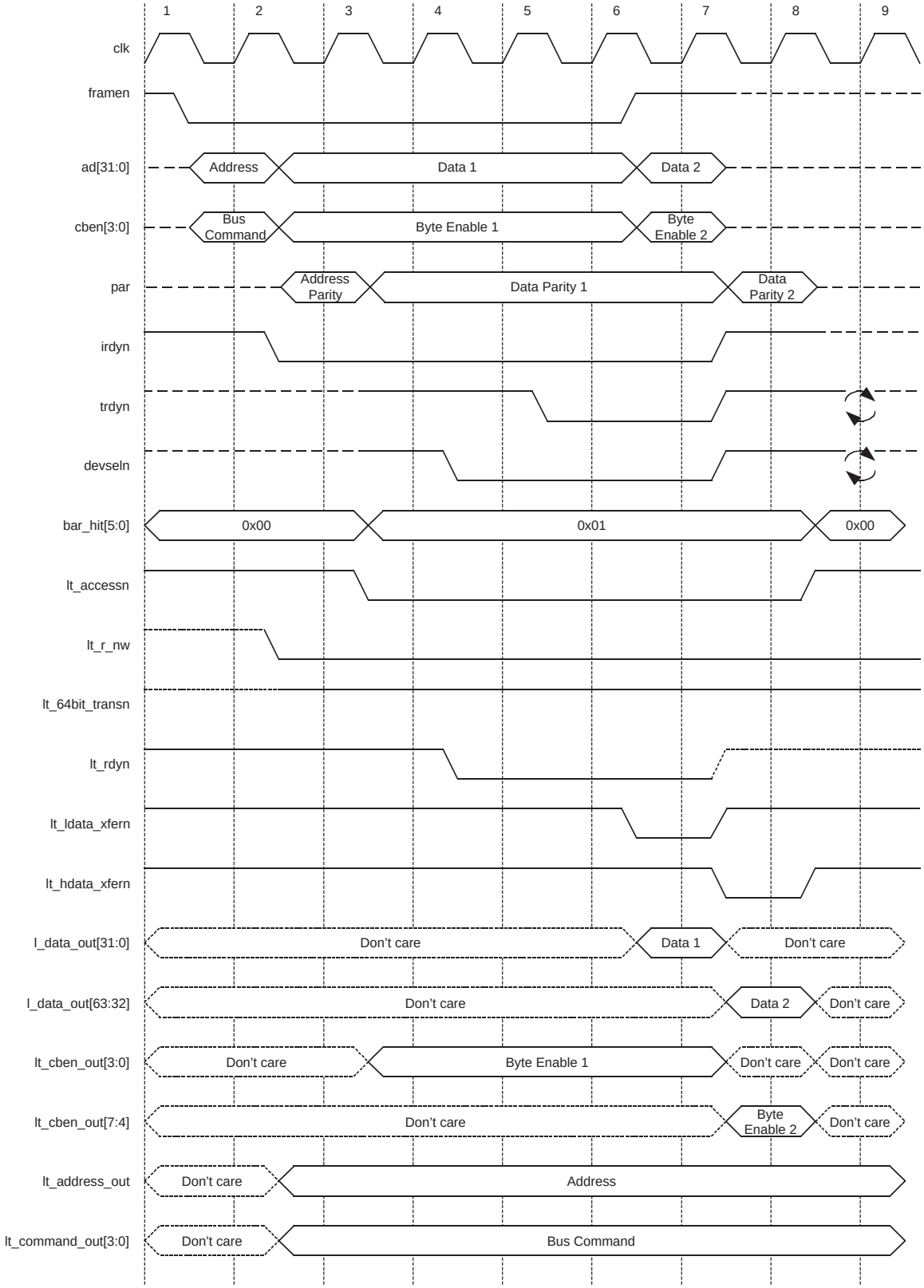


*Table 2-34. 32-bit Target Single Write Transaction with a 64-bit Local Interface*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Address | The master asserts framen and drives ad[31:0] and cben[3:0]. |
| 2 | Wait | The master drives the first byte enables (Byte Enable 1). If the master is ready to write data, it asserts irdyn and drives the first DWORD (Data 1) on ad[31:0].<br>The Core starts to decode the address and command. The Core drives the lt_address_out and lt_command_out to the back-end. |
| 3 | Wait | If there is an address match, the Core drives the bar_hit signals to the back-end. The back-end can use the bar_hit as a chip select. |
| 4 | Wait | If the DEVSEL_TIMING is set to slow, the Core asserts devseln on clock after bar_hit. If the back-end will be ready to write data in two cycles, it can assert lt_rdyn. |
| 5 | Data 1 | trdyn is asserted since lt_rdyn was asserted the previous cycle. |
| 6 | Data 2 | Quad Word Aligned<br>The Core keeps trdyn asserted and puts Data 1 on the lower DWORD of lt_data_out.<br>If both irdyn and trdyn are asserted on the previous cycle, the master drives the next byte enables (Byte Enable 2) on cben[3:0]. If the PCI master is still ready to write data, it keeps irdyn asserted and drives the next DWORD (Data 2) on ad[31:0].<br>If both irdyn and trdyn were asserted on the previous cycle, the Core asserts lt_ldata_xfern to the back-end to signify that Data 1 is valid. With lt_ldata_xfern asserted, the back-end doesn't write the data or increment the address counter.<br>Double Word Aligned<br>The Core keeps trdyn asserted and puts Data 1 on the upper DWORD of lt_data_out.<br>If both irdyn and trdyn are asserted on the previous cycle, the master drives the next byte enables (Byte Enable 2) on cben[3:0]. If the master is still ready to write data, it keeps irdyn asserted and drives the next DWORD (Data 2) on ad[31:0].<br>If irdyn, trdyn and lt_rdyn are asserted on the previous cycle, the Core asserts lt_hdata_xfern to the back-end to signify that Data 1 is valid. With lt_hdata_xfern asserted, the back-end can safely write the QWORD (Don't care and Data 1) and increment the address counter. |
| 7 | Turn around | If both irdyn and trdyn are asserted on the previous cycle, the master relinquishes control of framen, ad[31:0] and cben[3:0].<br>It also de-asserts irdyn if both trdyn and irdyn were asserted last cycle.It de-asserts both devseln and trdyn if both trdyn and irdyn were asserted last cycle. |
| 8 | Idle | The Core relinquishes control of devseln and trdyn.The target signals to the back-end that the transaction is complete by clearing bar_hit. It also de-asserts lt_data_xfern. |

## Configuration Read and Write Transactions

The PCI IP core handles configuration transactions from addresses 00h to 40h. The Local Target Interface has no control of these types of accesses and is independent these transactions. However, these transactions are still provided for verification purposes.

The PCI IP core only supports 32-bit, single data phase transactions to configuration registers. An individual idsel signal is connected to each PCI IP core device. Otherwise, read and write transactions are like the standard memory and I/O transactions. Figure 2-30 and Table 2-35 illustrate an example of a configuration read. Figure 2-31 and Table 2-36 shows an example of a configuration write.

The Capabilities List accesses and other configuration accesses over address 40h are beyond the PCI IP core's ability to complete the transaction without intervention from the Local Target Interface. Therefore, accesses to memory locations over address 40h are treated as local accesses and handled by the local target interface control. These configuration accesses are discussed further in Advanced Configuration Accesses section.

***Figure 2-30. Basic Configuration Read***

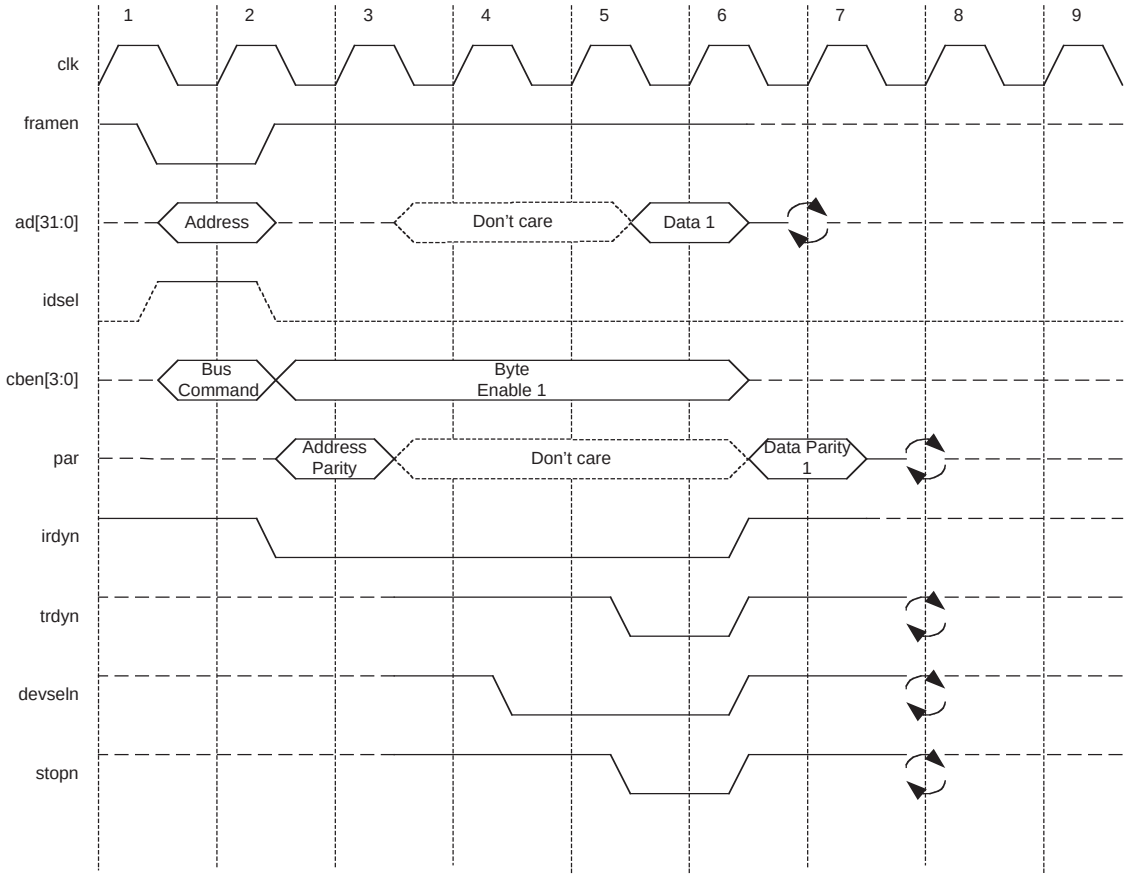



***Table 2-35. Basic Configuration Read***

| CLK | PCI Bus Phase | Description |
|---|---|---|
| 1 | Address | The master asserts `framen` and `idsel`. It drives the configuration address and Configuration Read command. The configuration address is `ad[1:0]` = 00 (type zero access); `ad[7:2]` = configuration DWORD address; `ad[10:8]` = function number; and `ad[31:11]` = unused. |
| 2 | Turn around | The master tri-states `ad[31:0]` and drives the first byte enable (Byte Enable 1). If the master is ready to receive data, it asserts `irdyn`.The Core starts to decode the address and command. |
| 3 | Wait | The address decode continues. |
| 4 | Wait | If the devsel_timing is set to slow, the Core asserts `devseln`. The Core is ready to put data out on the next cycle. |
| 5 | Data 1 | The data cycle starts as the target Core `trdyn` and puts Data 1 on `ad`. The Core also asserts stopn to ensure the configuration transaction is single data phase. |
| 6 | Turn around | The master relinquishes control of `framen` and `cben[3:0]`. It de-asserts `irdyn` if both `trdyn` and `irdyn` were asserted during the last cycle.<br>The Core relinquishes control of `ad[31:0]`. It de-asserts both `devseln`, `trdyn` and `stopn` if both `trdyn` and `irdyn` were asserted during the last cycle. |
| 7 | Idle | The Core relinquishes control of `devseln`, `trdyn` and `stopn`. |

***Figure 2-31. Basic Configuration Write***

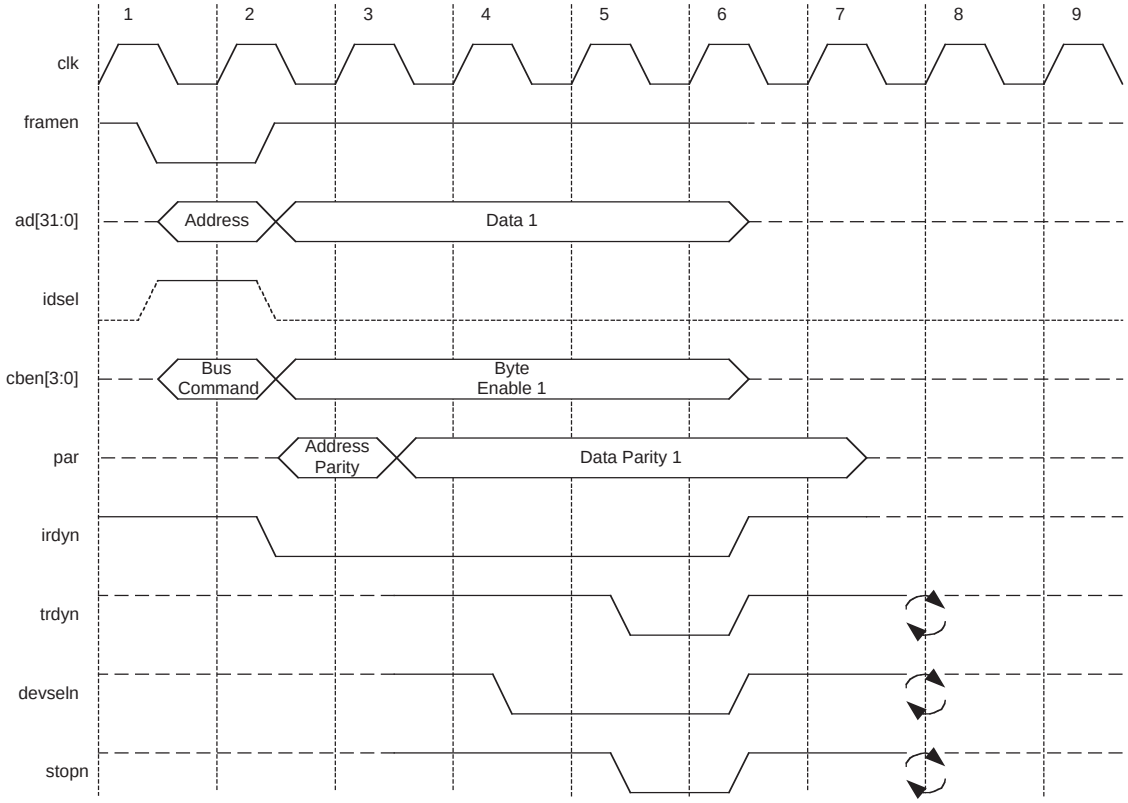



***Table 2-36. Basic Configuration Write***

| CLK | PCI Bus Phase | Description |
|---|---|---|
| 1 | Address | The master asserts `framen` and `idsel`. It drives the configuration address and configuration write command. The configuration address is `ad[1:0]` = 00 (type zero access); `ad[7:2]` = (Configuration DWORD address); `ad[10:8]` = (function number); `ad[31:11]` = unused. |
| 2 | Wait | The master drives the first byte enables (Byte Enable 1). If the bridge is ready to write data, it asserts `irdyn` and drives the first DWORD (Data 1) on `ad[31:0]`. The master signals the last data phase when it de-asserts `framen`.The Core starts to decode the address and command. |
| 3 | Wait | The address decode continues. |
| 4 | Wait | If the DEVSEL_TIMING is set to slow, the Core asserts `devseln`. The Core should be ready to get the data on the next cycle. |
| 5 | Data | The `trdyn` signal is asserted and the Core writes the DWORD. The Core also asserts `stopn` to ensure the configuring transaction is single data phase. |
| 6 | Turn around | The master relinquishes control of `framen`, `ad[31:0]` and `cben[3:0]`. It de-asserts `irdyn` if both `trdyn` and `irdyn` were asserted last cycle.<br>The Core de-asserts `devseln`, `trdyn` and `stopn` if both `trdyn` and `irdyn` were asserted last cycle. |
| 7 | Idle | The Core relinquishes control of `devseln`, `trdyn` and `stopn`. |

## PCI Target I/O Read and Write Transactions

Designing a PCI target application using I/O space is not recommended for several reasons. They include legacy device conflicts, and full address and byte enable decoding for all I/O locations. However, the PCI IP core does support I/O space. Transactions to I/O locations are similar to the basic memory transactions discussed in the Basic PCI Target Read and Write Transactions section.

In legacy systems, I/O space is limited. The system generally uses I/O space for vital system components such as interrupt controllers. These system components are spread throughout the I/O space, leaving only small gaps for additional devices that require I/O space. If I/O space is used in a legacy system, it is limited to 256 bytes.

By definition, read and write transactions to I/O space can only be completed using 32-bit PCI transactions. Decoding all 32 bits in the address and determining which byte enables (`cben[3:0]`) are supported is necessary. The back-end application responds with a target abort if any unsupported byte enable combinations are requested.

## Advanced Target Transactions

Some PCI applications require more than basic read and write transactions. For these applications, the *PCI Local Bus Specification, Revision 3.0* offers advanced features to handle the more difficult aspects of the PCI bus. The advanced features are used to provide the PCI application with more flexibility and improve the overall PCI system performance. The following sections offer more detail on these advanced PCI bus features.

### Wait States

Care must be taken when processing wait states to be compliant with the *PCI Local Bus Specification, Revision 3.0.* Once a PCI master or a PCI target signals that it is ready to send or receive data, it must complete the current PCI data phase. For example, if the PCI IP core is ready to write data and the PCI master inserts wait states, the PCI IP core must wait to write the data until the master is ready again. Additionally, if the PCI IP core has committed to a data phase by asserting `trdyn`, it can not insert any wait states until the next data phase. Coincident master and target wait state insertion is also a possibility. Refer to the PCI Specification for more information regarding coincident wait state insertion.

Two types of wait states that can occur on the PCI bus. The first is master wait state insertion. When the PCI master inserts wait states, the PCI IP core must hold off data until the PCI master is ready. The PCI IP core inserts the second type of wait states. The back-end application controls the PCI IP core's wait state insertion via the Local Target Interface.

Figure 2-32 and Table 2-37 illustrate master-inserted and target-inserted wait states for read transactions. The figure illustrates how the PCI interface correlates to the Local Target Interface. The table gives a clock-by-clock description of each event in the figure.

***Figure 2-32. 32-bit Target Read Transaction with Master Wait State***

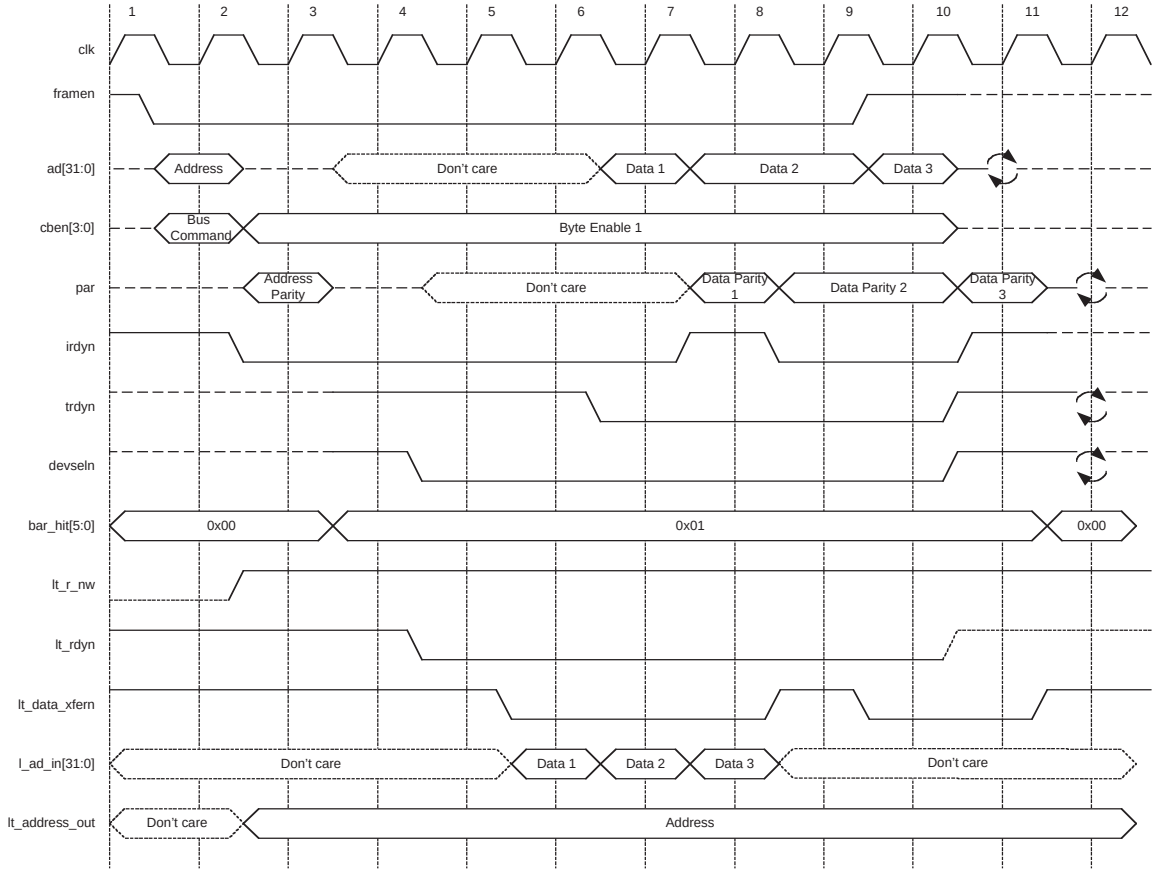



***Table 2-37. 32-bit Target Read Transaction with Master Wait State***

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Address | The PCI master asserts `framen` and drives `ad[31:0]` and `cben[3:0]`. |
| 2 | Turn around | The PCI master tri-states `ad[31:0]`.The PCI master is ready to receive data. It asserts `irdyn`. |
| 3 | Target Wait | The Core starts to decode the address and command. |
| 4 | Target Wait | If the DEVSEL_TIMING is set to slow, the Core asserts `devseln` one clock after `bar_hit`. `trdyn` is kept asserted since the back-end logic did not assert `lt_rdyn` during clock cycle 3.<br><br>The back-end logic asserts `lt_rdyn` during this cycle. |
| 5 | Target Wait | The PCI IP core inserts a wait state as it has not yet asserted the `trdyn` signal. Since both `irdyn` and `lt_rdyn` were asserted on the previous cycle, the Core asserts `lt_data_xfern`. |
| 6 | Data 1 | The Core asserts `trdyn` and drives Data 1 from the local target on to the PCI `ad[31:0]` bus. If the PCI master is still ready to receive data, it keeps `irdyn` asserted and drives the next byte enables (Byte Enables) on `cben[3:0]`. If the back-end kept `lt_rdyn` asserted in the previous two cycles, the Core keeps `trdyn` asserted and puts Data 2 on `ad[31:0]`.If both `irdyn` and `lt_rdyn` are asserted on the previous cycle, the Core re-asserts `lt_data_xfern` to the back-end. |
| 7 | Master Wait | The PCI master is not ready to receive data, it de-asserts `irdyn` If the back-end keeps `lt_rdyn` asserted previous two cycles, the Core keeps `trdyn` asserted and puts Data 2 on `ad[31:0]`.If both `irdyn` and `lt_rdyn` are asserted on the previous cycle, the Core re-asserts `lt_data_xfern` to the back-end. The back-end should increment the address counter. |

*Table 2-37. 32-bit Target Read Transaction with Master Wait State (Continued)*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 8 | Data 2 | If the PCI master is ready to receive data, it asserts `irdyn` and drives the next byte enables (Byte Enable 3) on `cben[3:0]`.Because `irdyn` is not asserted on the previous cycle, the Core de-asserts `lt_data_xfern` on the local interface. |
| 9 | Data 3 | Since the last data phase, the master asserts `irdyn` and de-asserts `framen`.If both `irdyn` and `lt_rdyn` are asserted on the previous cycle, the Core re-asserts `lt_data_xfern` to the back-end. |
| 10 | Turn around | The master relinquishes control of `framen`, `ad[31:0]` and `cben[3:0]`.The Core de-asserts both `devseln` and `trdyn`. |
| 11 | Idle | The Core relinquishes control of `devseln` and `trdyn`. |

Figure 2-33 and Table 2-38 show master-inserted and target-inserted wait states that are inserted on write transactions. The figure illustrates how the PCI interface correlates to the Local Target Interface. The table gives a clock-by-clock description of each event in the figure.

*Figure 2-33. 32-bit Target Write Transaction with Master Wait State*

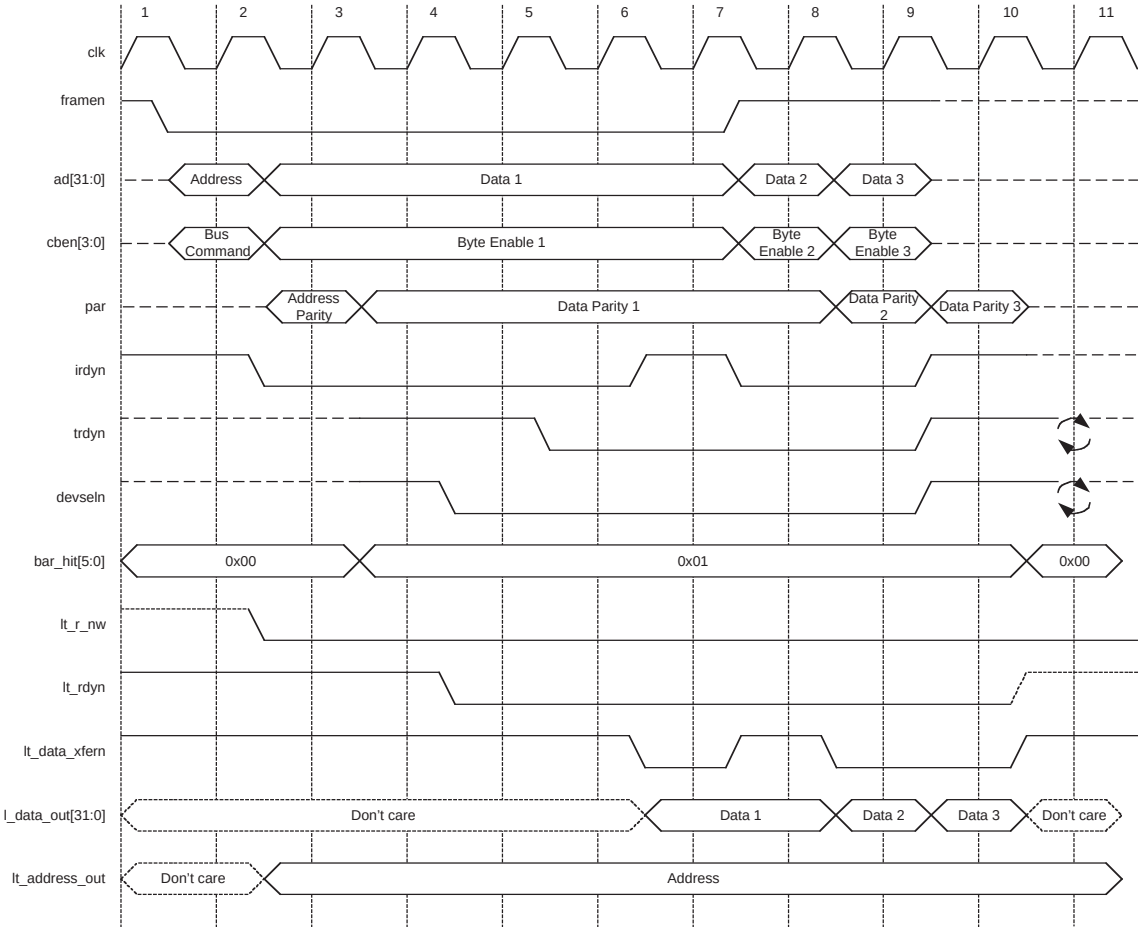


*Table 2-38. 32-bit Target Write Transaction with Master Wait State*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Address | The master asserts framen and drives ad[31:0] and cben[3:0]. |
| 2 | Target Wait | The PCI master is ready to receive data. It asserts irdyn. |
| 3 | Target Wait | The Core starts to decode the address and command. |
| 4 | Target Wait | If the DEVSEL_TIMING is set to slow, the target asserts devseln on clock after bar_hit.The lt_rdyn signal is driven low to indicate that the back-end application is ready to receive data. |
| 5 | Data 1 | The irdyn and trdyn signals are asserted Data 1 is registered from ad[31:0]. |
| 6 | Master Wait | With lt_data_xfern signal asserted Data1 is registered on lt_data_out[31:0].The master is not ready to receive data. It inserts a wait state by de-asserting irdyn. The master holds Data 1 on the ad[31:0] lines and continues to drive the first byte enables (Byte Enable 1). |
| 7 | Data 2 | With the irdyn signal asserted Data 2 is driven on to ad[31:0].Because irdyn is not asserted on the previous cycle, the Core de-asserts lt_data_xfern on the local interface. |
| 8 | Data 3 | With the irdyn signal asserted Data 3 is driven on to ad[31:0].If both irdyn and lt_rdyn are asserted on the previous cycle, the Core re-asserts lt_data_xfern to the back-end. |
| 9 | Turn around | With lt_data_xfern signal asserted Data 2 is registered on lt_data_out[31:0].The master relinquishes control of framen, ad[31:0] and cben[3:0].The Core de-asserts both devseln and trdyn if both trdyn and irdyn were asserted last cycle. |
| 10 | Idle | The Core relinquishes control of devseln and trdyn. |

## Burst Read and Write Target Transactions

Burst read and write transactions to memory addresses are used to achieve the high throughput that is typically associated with the PCI bus. The following lists the commands for the PCI IP core that support bursting.

- Memory Read
- Memory Write
- Memory Read Multiple
- Dual Address Cycle
- Memory Read Line
- Memory Write and Invalidate

These PCI burst transactions are described based on the different PCI and Local bus configurations supported by the PCI IP core. Although the fundamentals of bursting are similar for all PCI IP core configurations, different bus configurations require slightly different Local Target Interface signaling. The PCI IP core does not support bursting for Configuration Space or I/O space accesses. Refer to the following sections for more information on bursting with specific PCI Target configurations:

- 32-Bit PCI Target with a 32-Bit Local Bus
- 64-Bit PCI Target with a 64-Bit Local Bus
- 32-Bit PCI Target with a 64-Bit Local Bus

Typically for burst transactions, the PCI master and the PCI target has a predefined number of PCI data phases that are to be transferred. The PCI master will know the number of data phases that are to be transferred based on the software driver and specifications that were defined by the PCI IP core's implementation. The PCI IP core will have a predefined number of data phases based on the design requirements of the PCI Target core's application. The design requirements include items like FIFO depth and the general ability to handle throughput. Handling these requirements is covered in more detail in the *PCI Local Bus Specification, Revision 3.0.*

### 32-Bit PCI Bus and a 32-Bit Local Bus

The following section discusses read and write, burst transactions for a PCI IP core configured with a 32-bit PCI bus and a 32-bit Local bus. Figure 2-34 and Table 2-39 show a 32-bit burst read transaction. The figure illustrates how the PCI interface correlates to the Local Target Interface. The table gives a clock-by-clock description of each event that occurs in the figure.

***Figure 2-34. 32-bit Target Burst Read Transaction with a 32-bit Local Interface***

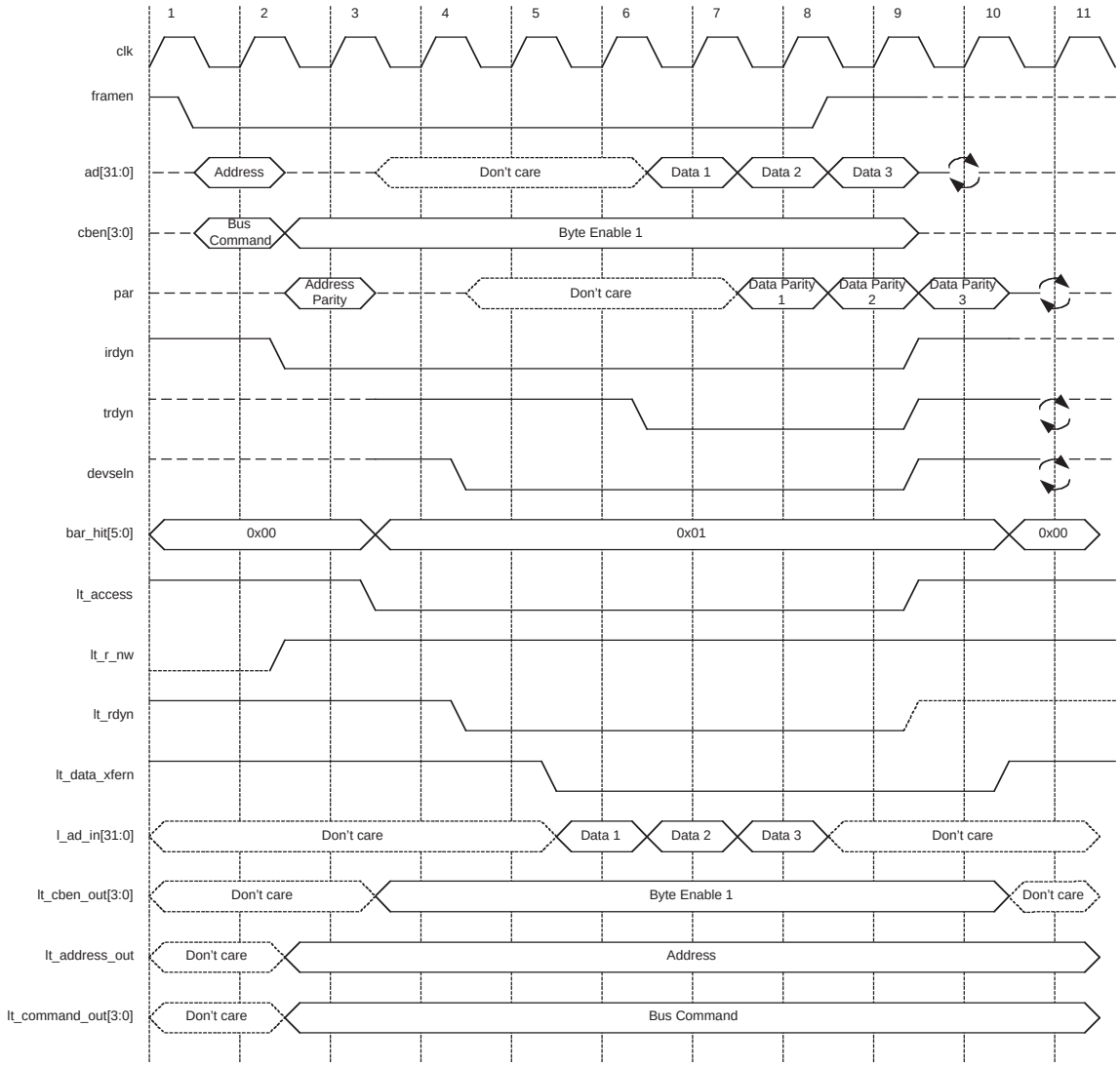



***Table 2-39. 32-bit Target Burst Read Transaction with a 32-bit Local Interface***

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Address | The PCI master asserts `framen` and drives `ad[31:0]` and `cben[3:0]`. |
| 2 | Turn around | The PCI master tri-states `ad[31:0]` and drives the first byte enables (Byte Enable 1) on `cben[3:0]`. If the PCI master is ready to receive data, it asserts `irdyn`.The Core starts to decode the address and command and drives the `lt_address_out` to the back-end application. |
| 3 | Wait | If there is an address match, the Core drives the `bar_hit` signals to the back-end. The back-end can use the `bar_hit` as a chip select. |

*Table 2-39. 32-bit Target Burst Read Transaction with a 32-bit Local Interface (Continued)*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 4 | Wait | If the DEVSEL_TIMING is set to slow, the Core asserts devseln on clock after bar_hit. If the back-end will be ready to put data out on the next cycle, it can assert lt_rdyn. |
| 5 | Wait | The Core asserts lt_data_xfern since lt_rdyn was asserted the previous cycle. The back-end drives the first DWORD (Data 1) on l_ad_in. |
| 6 | Data 1 | With lt_rdyn asserted for the previous two cycles, the burst cycle starts, so the Core asserts trdyn and puts Data 1 on ad[31:0].<br>If both irdyn and lt_rdyn are asserted on the previous cycle, the target keeps lt_data_xfern asserted to the back-end. The back-end can increment the address counter and put the next DWORD (Data 2) on l_ad_in. |
| 7 | Data 2 | If the PCI master is still ready to receive data, it keeps irdyn asserted and drives the next byte enables (Byte Enable 2) on cben[3:0].<br>If the back-end keeps lt_rdyn asserted for the previous two cycles, the Core keeps trdyn asserted and puts Data 2 on ad[31:0].<br>If both irdyn and lt_rdyn are asserted on the previous cycle, the Core keeps lt_data_xfern asserted to the back-end. The back-end can increment the address counter and put the next DWORD (Data 3) on l_ad_in. |
| 8 | Data 3 | If the PCI master is still ready to receive data, it keeps irdyn asserted and drives the next byte enables (Byte Enable 3) on cben[3:0].<br>The master signals the end of the burst when it de-asserts framen.If the back-end keeps lt_rdyn asserted previous two cycles, the Core keeps trdyn asserted and puts Data 3 on ad[31:0].<br>If both irdyn and lt_rdyn are asserted on the previous cycle, the Core keeps lt_data_xfern asserted to the back-end application. The back-end can increment the address counter and put the next DWORD (Don't care) on l_ad_in. |
| 9 | Turn around | The master relinquishes control of framen and cben[3:0]. It de-asserts irdyn if both trdyn and irdyn were asserted last cycle.The Core relinquishes control of ad[31:0]. It de-asserts both devseln and trdyn if both trdyn and irdyn were asserted last cycle. The Core also signals to the back-end that the transaction is complete by clearing bar_hit. The Core de-asserts lt_data_xfern. |
| 10 | Idle | The Core relinquishes control of devseln and trdyn. |

Figure 2-35 and Table 2-40 show an example of a 32-bit burst write transaction. The assumption is that the device select timing is set to slow and wait states are not inserted. The figure illustrates how the PCI interface correlates to the Local Target Interface. The table gives a clock-by-clock description of each event that occurs in the figure.

***Figure 2-35. 32-bit Target Burst Write Transaction with a 32-bit Local Interface***

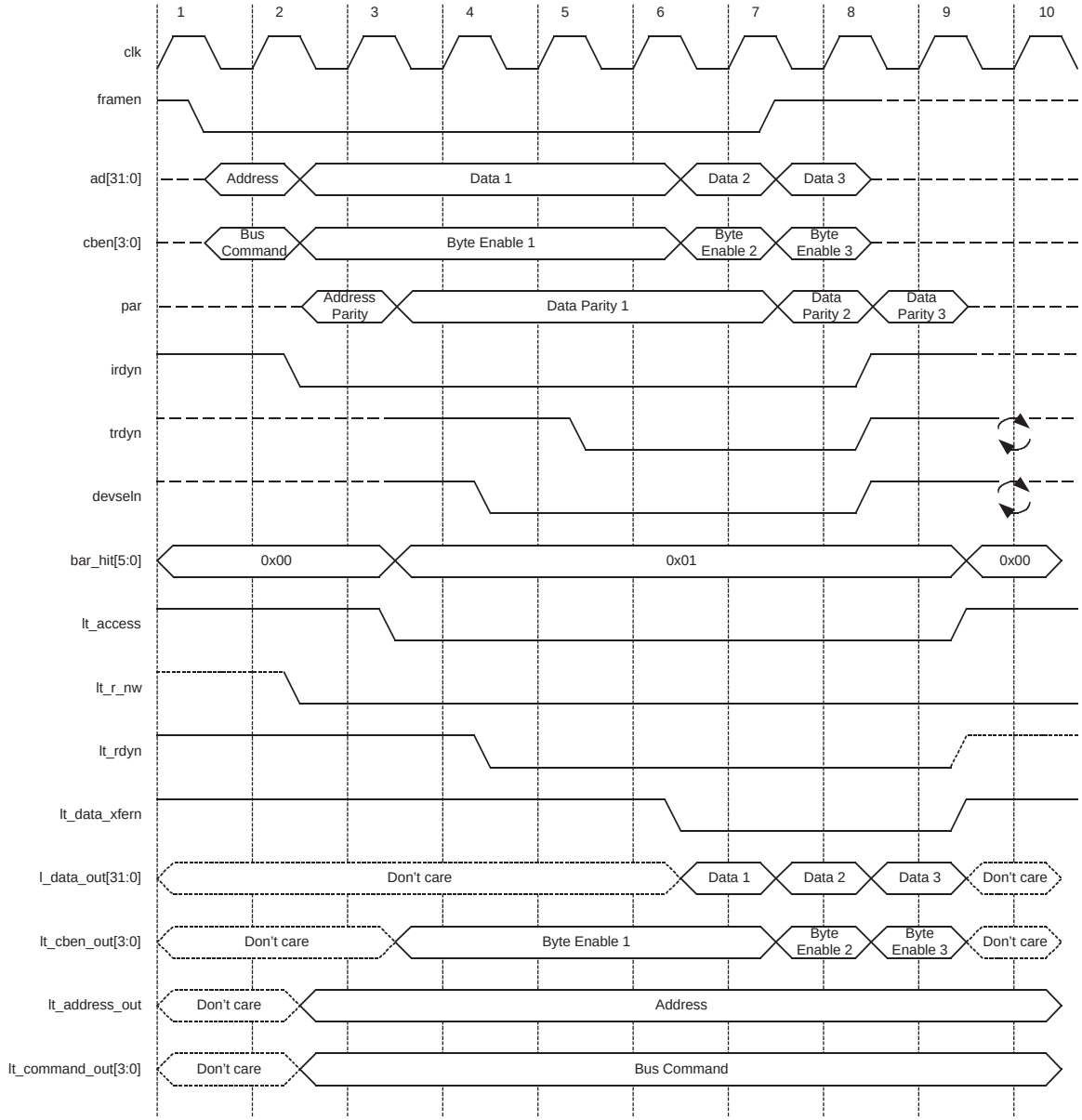


*Table 2-40. 32-bit Target Burst Write Transaction with a 32-bit Local Interface*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Address | The PCI master asserts framen and drives ad[31:0] and cben[3:0]. |
| 2 | Wait | The PCI master drives the first byte enables (Byte Enable 1) on cben[3:0]. If the master is ready to write data, it asserts irdyn and drives the first DWORD (Data 1) on ad[31:0]. The PCI IP core starts to decode the address and command and drives the lt_address_out to the back-end. |
| 3 | Wait | If there is an address match, the Core drives the bar_hit signals on the Local Interface. The back-end can use the bar_hit as a chip select. |
| 4 | Wait | If the DEVSEL_TIMING is set to slow, the Core asserts devseln on clock after bar_hit. If the back-end will be ready to write data in two cycles, it can assert lt_rdyn. |
| 5 | Data 1 | trdyn is asserted since lt_rdyn was asserted the previous cycle. |
| 6 | Data 2 | If the back-end keeps lt_rdyn asserted for the previous cycle, the Core keeps trdyn asserted and puts Data 1 on l_data_out. If both irdyn and trdyn are asserted on the previous cycle, the master drives the next byte enables (Byte Enable 2) on cben[3:0]. If the master is still ready to write data, it keeps irdyn asserted and drives the next DWORD (Data 2) on ad[31:0]. If both irdyn and lt_rdyn are asserted on the previous cycle, the Core asserts lt_data_xfern to the back-end to signify Data 1 was transferred successfully. With lt_data_xfern asserted, the back-end can safely write Data 1 and increment the address counter. |
| 7 | Data 3 | If the back-end keeps lt_rdyn asserted for the previous cycle, the Core keeps trdyn asserted and puts Data 2 on l_data_out.<br>If both irdyn and trdyn are asserted on the previous cycle, the master drives the next byte enables (Byte Enable 3) on cben[3:0]. If the master is still ready to write data, it keeps irdyn asserted and drives the next DWORD (Data 3) on ad[31:0].<br>The master signals the end of the burst when it de-asserts framen. If both irdyn and lt_rdyn are asserted on the previous cycle, the Core keeps lt_data_xfern asserted to the back-end to signify Data 2 was transferred successfully. |
| 8 | Turn around | If the back-end keeps lt_rdyn asserted for the previous cycle, the Core puts Data 3 on l_data_out.<br>If both irdyn and trdyn are asserted on the previous cycle, the master relinquishes control of framen, ad and cben. It also de-asserts irdyn if both trdyn and irdyn were asserted last cycle.<br>If both irdyn and lt_rdyn are asserted on the previous cycle, the Core keeps lt_data_xfern asserted to the back-end to signify Data 3 was transferred successfully. With lt_data_xfern asserted the back-end can safely write Data 3 and increment the address counter.<br>The PCI IP core de-asserts both devseln and trdyn if both trdyn and irdyn were asserted last cycle. |
| 9 | Idle | The target signals to the back-end that the transaction is complete by clearing bar_hit. It also de-asserts lt_data_xfern. The Core relinquishes control of devseln and trdyn. |

### 64-Bit PCI Target with a 64-Bit Local Bus

The following discusses read and write burst transactions for the PCI IP core configured with a 64-bit PCI bus and a 64-bit local bus. Figure 2-36 and Table 2-41 illustrate a 64-bit burst write transaction. The figure shows how the PCI interface correlates to the local interface. The table gives a clock-by-clock description of each event that occurs in the figure.

The 32-bit burst transaction, as described in the 32-Bit PCI Bus and a 32-Bit Local Bus section, is similar to a 32-bit burst transaction for the 64-bit PCI IP core configuration. When the 64-bit target core responds to a 32-bit burst transaction, the upper 32 bits of the data bus should be ignored.

***Figure 2-36. 64-bit Target Burst Read Transaction with a 64-bit Local Interface***

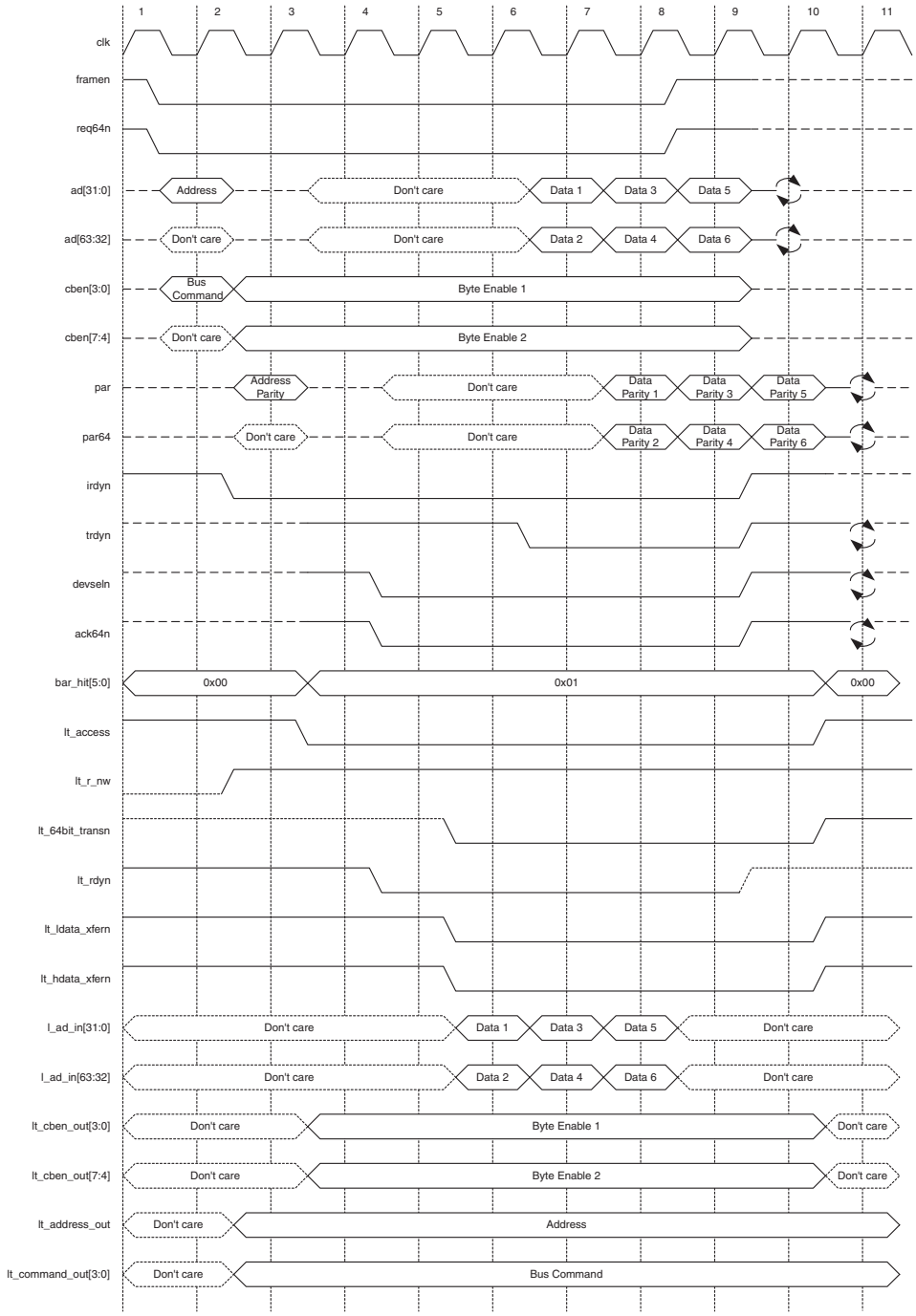


*Table 2-41. 64-bit Target Burst Read Transaction with a 64-bit Local Interface*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Address | The PCI master asserts framen and drives ad[31:0] and cben[3:0]. It requests a 64-bit transaction by asserting req64n with framen. |
| 2 | Turn around | The master tri-states ad[63:0] and drives the first byte enables (Byte Enable 1 and 2) cben[7:0]. If the PCI master is ready to receive data, it asserts irdyn.The Core starts to decode the address and command.<br>The target drives the lt_address_out to the back-end. The lt_64bit_trans signal is driven high to signal the back-end that a 64-bit transaction has been requested. |
| 3 | Wait | If there is an address match, the Core drives the bar_hit signals to the Local Interface. The back-end application can use the bar_hit as a chip select. |
| 4 | Wait | If the DEVSEL_TIMING is set to slow, the PCI IP core asserts devseln and ack64n on clock after bar_hit. If the back-end will be ready to put data out on the next cycle, it can assert lt_rdyn. The Core acknowledges the 64-bit transaction by asserting ack64n. |
| 5 | Wait | The PCI IP core asserts lt_ldata_xfern and lt_hdata_xfern since lt_rdyn was asserted the previous cycle. The back-end drives the first QWORD (Data 1 and 2) on l_ad_in. |
| 6 | Data 1 and 2 | With lt_rdyn asserted previous two cycles, the burst cycle starts, so the Core asserts trdyn and puts (Data 1 and 2) on ad[63:0].If both irdyn and lt_rdyn are asserted on the previous cycle, the Core keeps lt_ldata_xfern and lt_hdata_xfern asserted to the back-end. The back-end can increment the address counter and put the next QWORD (Data 3 and 4) on l_ad_in. |
| 7 | Data 3 and 4 | If the master is still ready to receive data, it keeps irdyn asserted and drives the next byte enables (Byte Enable 3 and 4) on cben[7:0].If the back-end keeps lt_rdyn asserted previous two cycles, the PCI IP core keeps trdyn asserted and puts (Data 3 and 4) on ad[63:0].<br>If both irdyn and lt_rdyn are asserted on the previous cycle, the Core keeps lt_ldata_xfern and lt_hdata_xfern asserted. The back-end application can increment the address counter and put the next QWORD (Data 5 and 6) on l_ad_in. |
| 8 | Data 5 and 6 | If the PCI master is still ready to receive data, it keeps irdyn asserted and drives the next byte enables (Byte Enable 5 and 6) on cben[7:0].<br>The PCI master signals the end of the burst when it de-asserts framen and req64n.If the back-end application keeps lt_rdyn asserted for the previous two cycles, the Core keeps trdyn asserted and puts Data 5 and 6 on ad[63:0].<br>If both irdyn and lt_rdyn are asserted on the previous cycle, the Core keeps lt_ldata_xfern and lt_hdata_xfern asserted. The back-end application can increment the address counter and put the next QWORD (Don't care) on l_ad_in. |
| 9 | Turn around | The master relinquishes control of framen, req64n and cben[7:0]. It de-asserts irdyn if both trdyn and irdyn were asserted last cycle.<br>The Core relinquishes control of ad[63:0]. It de-asserts devseln, ack64n and trdyn if both trdyn and irdyn were asserted last cycle. The Core also signals to the back-end that the transaction is complete by clearing bar_hit. The target de-asserts lt_ldata_xfern and lt_hdata_xfern. |
| 10 | Idle | The Core relinquishes control of devseln, ack64n and trdyn. |

Figure 2-37 and Table 2-42 illustrate a 64-bit burst write transaction. The figure shows how the PCI interface correlates to the Local Interface. The table gives a clock-by-clock description of each event that occurs in the figure.

***Figure 2-37. 64-bit Target Burst Write Transaction with a 64-bit Local Interface***

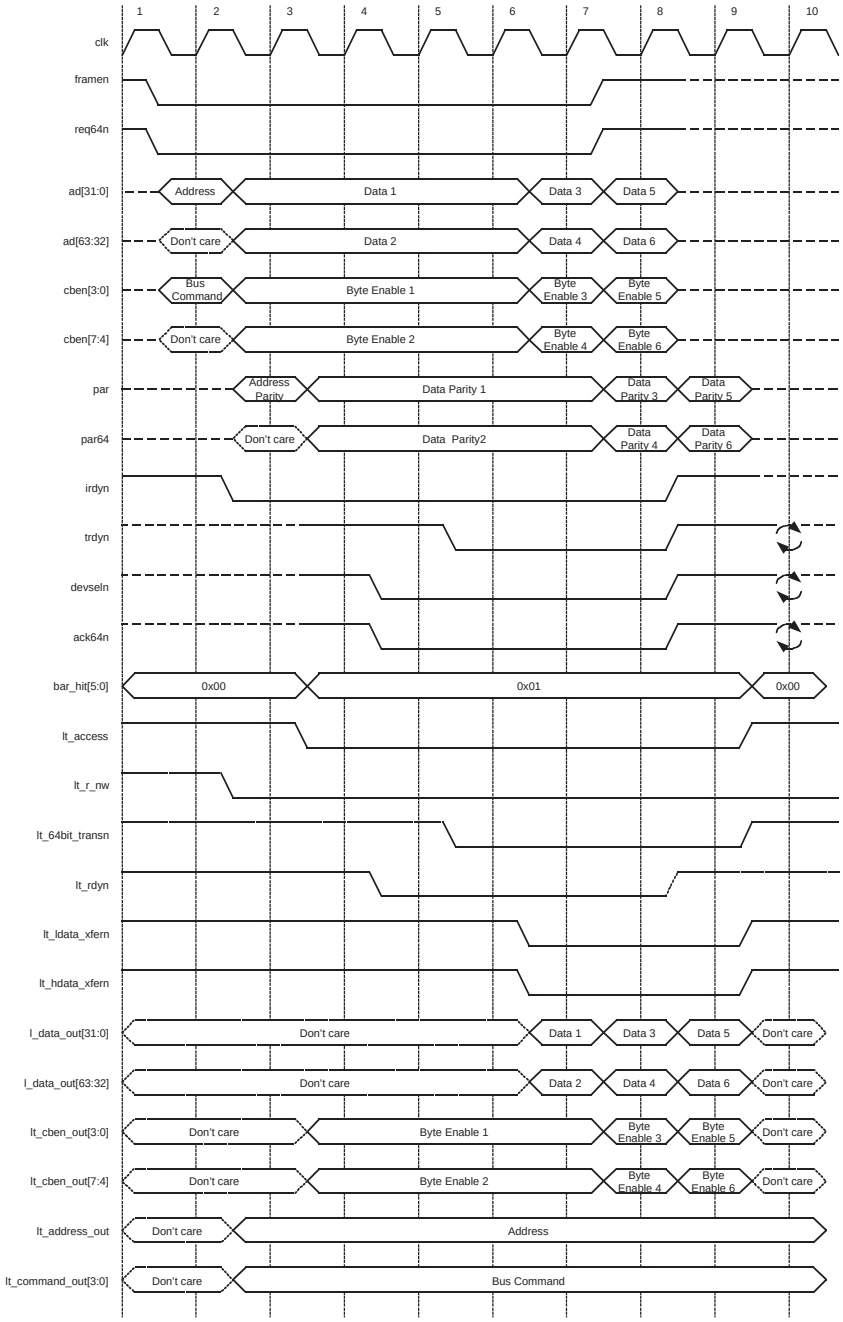


*Table 2-42. 64-bit Target Burst Write Transaction with a 64-bit Local Interface*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Address | The PCI master asserts framen and drives ad[31:0] and cben[3:0]. It requests a 64-bit transaction by asserting req64n with framen. |
| 2 | Wait | The PCI master drives the first byte enables (Byte Enable 1 and 2) on cben[7:0]. If the PCI master is ready to write data, it asserts irdyn and drives the first QWORD (Data 1 and 2) on ad[63:0].The Core starts to decode the address and command. The target drives lt_address_out to the back-end. The lt_64bit_trans signal is driven high to signal the back-end application that a 64-bit transaction has been requested. |
| 3 | Wait | If there is an address match, the Core drives the bar_hit signals to the back-end application. It can use bar_hit as a chip select. |
| 4 | Wait | If the DEVSEL_TIMING is set to slow, the Core asserts devseln on the clock after bar_hit. If the back-end is ready to write data in two cycles it can assert lt_rdyn. The PCI IP core acknowledges the 64-bit transaction by asserting ack64n. |
| 5 | Data 1 and 2 | The trdyn signal is asserted since lt_rdyn was asserted on the previous cycle. |
| 6 | Data 3 and 4 | If the back-end keeps lt_rdyn asserted on the previous cycle, the Core keeps trdyn asserted and puts Data 1 and 2 on lt_data_out.If both irdyn and trdyn are asserted on the previous cycle, the master drives the next byte enables (Byte Enable 3 and 4) on cben[7:0]. If the PCI master is still ready to write data, it keeps irdyn asserted and drives the next QWORD (Data 3 and 4) on ad[63:0]. If both irdyn and lt_rdyn are asserted on the previous cycle, the Core asserts lt_ldata_xfern and lt_hdata_xfern to the back-end to signify Data 1 and 2 is valid. With lt_ldata_xfern and lt_hdata_xfern asserted the back-end can safely write Data 1 and 2 and increment the address counter. |
| 7 | Data 5 and 6 | If the back-end keeps lt_rdyn asserted on the previous cycle, the Core keeps trdyn asserted and puts Data 3 and 4 on lt_data_out. If both irdyn and trdyn are asserted on the previous cycle, the PCI master drives the next byte enables (Byte Enable 5 and 6) on cben[7:0]. If it is still ready to write data, it keeps irdyn asserted and drives the next QWORD (Data 5 and 6) on ad[63:0]. The master signals the end of the burst when it de-asserts framen and req64n. If both irdyn and lt_rdyn are asserted on the previous cycle, the Core keeps lt_ldata_xfern and lt_hdata_xfern asserted to the back-end to signify Data 3 and 4 is valid. With lt_ldata_xfern and lt_hdata_xfern asserted the back-end can safely write Data 3 and 4 and increment the address counter. There is no signal yet to the back-end that the burst is over. |
| 8 | Turn around | If the back-end keeps lt_rdyn asserted the previous cycle, the target puts Data 5 and 6 on lt_data_out.If both irdyn and trdyn are asserted on the previous cycle, the master relinquishes control of framen, req64n, ad[63:0] and cben[7:0]. It also de-asserts irdyn if both trdyn and irdyn were asserted on the last cycle. If both irdyn and lt_rdyn were asserted on the previous cycle, the target keeps lt_ldata_xfern and lt_hdata_xfern asserted to the back-end to signify Data 5 and 6 is valid. With lt_ldata_xfern and lt_hdata_xfern asserted the back-end can safely write Data 5 and 6 and increment the address counter. It de-asserts devseln, ack64n and trdyn if both trdyn and irdyn were asserted last cycle. |
| 9 | Idle | The Core signals to the back-end that the transaction is complete by clearing bar_hit. It also de-asserts lt_ldata_xfern and lt_hdata_xfern.The target relinquishes control of devseln, ack46n and trdyn. |

**32-Bit PCI Target with a 64-bit Local Bus**

The following discusses read and write transactions for a PCI IP core configured with a 32-bit PCI bus and a 64-bit local bus. In order to present a full 64 bits of data to the Local Interface, two PCI data phase are required. Likewise retrieving 64 bits of data from the Local Interface, two PCI data phases are required.

The 32-bit PCI transaction, as described in the 32-Bit PCI Bus and 32-Bit Local Bus section, looks similar to these transactions; however, the data is handled differently at the Local Interface. When the 32-bit target core responds to a 32-bit burst transaction, the upper 32 bits of the Local data bus should be ignored or return 0's.

With a 64-bit back-end, it is assumed that the address counter needs to increment only by a Quad Word (QWORD) (8 bytes), so the local back-end control latches the complete QWORD and routes the proper DWORD to the PCI data bus. The lt_ldata_xfern and lt_hdata_xfern signals specify which DWORD is transferred.

If the starting address is QWORD aligned, the first DWORD is assumed to be the lower DWORD of a QWORD. Otherwise, it is the upper DWORD. If the starting address is not QWORD aligned, it must be DWORD aligned.

Figure 2-38 and Table 2-43 illustrate a burst transaction to a 32-bit PCI IP core with a 64-bit Local Interface. The figure illustrates how the PCI interface correlates to the Local Interface. The table gives a clock-by-clock description of each event in the figure.

***Figure 2-38. 32-bit Target Burst Read Transaction with a 64-bit Local Interface***

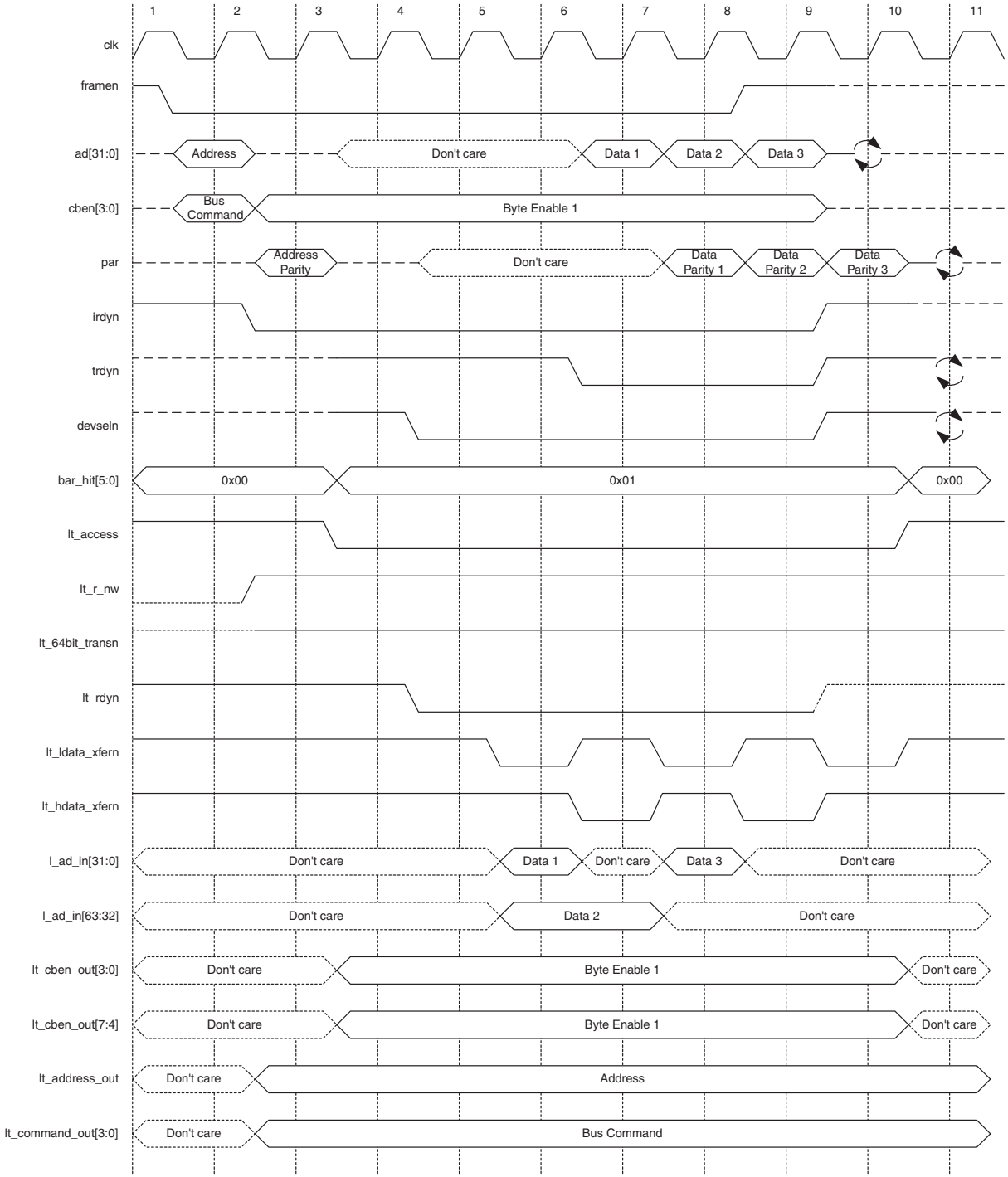


*Table 2-43. 32-bit Target Burst Read Transaction with a 64-bit Local Interface*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Address | The master asserts framen and drives ad[31:0] and cben[3:0]. |
| 2 | Turn around | The master tri-states ad[31:0] and drives the first byte enables (Byte Enable 1) on cben[3:0]. If the master is ready to receive data, it asserts irdyn.The Core starts to decode the address and command. It drives the lt_address_out to the back-end. |
| 3 | Wait | If there is an address match, the Core drives the bar_hit signals to the back-end. The back-end can use bar_hit as a chip select. |
| 4 | Wait | If the DEVSEL_TIMING is set to slow, the Core asserts devseln on the clock following bar_hit. If the back-end application is ready to put data out on the next cycle, it asserts lt_rdyn. |
| 5 | Wait | The Core asserts lt_ldata_xfern since lt_rdyn was asserted on the previous cycle and the initial address is QWORD aligned. The back-end drives the first QWORD (Data 1 and Data 2) on l_ad_in. |
| 6 | Data 1 | Quad Word Aligned<br>With lt_rdyn asserted for the previous two cycles, the burst cycle starts. The PCI IP core asserts trdyn and puts Data 1 on ad[31:0].<br>The Core de-asserts lt_ldata_xfern. If irdyn is asserted on the previous cycle, the Core asserts lt_hdata_xfern to the back-end. With lt_hdata_xfern de-asserted the previous cycle, the back-end does not increment the address counter and holds the QWORD (Data 1 and Data 2) on l_ad_in.<br>Double Word Aligned<br>With lt_rdyn asserted for the previous two cycles, the burst cycle starts, so the Core asserts trdyn and puts Data 1 on ad since the initial address is DWORD aligned. Notice that the lower DWORD from l_ad_in is discarded.<br>The Core de-asserts lt_hdata_xfern. If both irdyn and lt_rdyn are asserted on the previous cycle, the Core asserts lt_ldata_xfern to the back-end. With lt_hdata_xfern asserted the previous cycle, the back-end can increment the address counter and put the next QWORD (Data 2 and Data 3) on l_ad_in. |
| 7 | Data 2 | Quad Word Aligned<br>If the master is still ready to receive data, it keeps irdyn asserted and drives the next byte enables. The Core keeps trdyn asserted and puts Data 2 on and loads the appropriate byte enables.<br>The Core de-asserts lt_hdata_xfern. If both irdyn and lt_rdyn are asserted on the previous cycle, the Core asserts lt_ldata_xfern to the back-end. With lt_hdata_xfern asserted the previous cycle, the back-end can increment the address counter and put the next QWORD (Data 3 and Data 4) on l_ad_in.<br>Double Word Aligned<br>If the master is still ready to receive data, it keeps irdyn asserted and drives the next byte enables on cben[3:0].<br>If the back-end keeps lt_rdyn asserted previous two cycles, the Core keeps trdyn asserted and puts Data 2 on ad[31:0]. If the master is still ready to receive data, it keeps irdyn asserted and drives the byte enables on cben[3:0].<br>The Core de-asserts lt_ldata_xfern. If irdyn is asserted on the previous cycle, the Core asserts lt_hdata_xfern to the back-end. With lt_hdata_xfern de-asserted on the previous cycle, the back-end does not increment the address counter and holds the QWORD (Data 2 and Data 3) on l_ad_in. |

*Table 2-43. 32-bit Target Burst Read Transaction with a 64-bit Local Interface (Continued)*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 8 | Data 3 | Quad Word Aligned<br>If the PCI master is still ready to receive data, it keeps irdyn asserted and drives the next byte enables (Byte Enable 3) on cben[3:0]. It signals the end of the burst when it de-asserts framen.<br>If the back-end keeps lt_rdyn asserted for the previous two cycles, the Core keeps trdyn asserted and puts Data 3 on ad[31:0].<br>The Core de-asserts lt_ldata_xfern. If irdyn is asserted on the previous cycle, the Core asserts lt_hdata_xfern to the back-end. With lt_hdata_xfern de-asserted on the previous cycle, the back-end does not increment the address counter and holds the QWORD (Data 3 and Data 4) on l_ad_in.<br>Double Word Aligned<br>If the master is still ready to receive data, it keeps irdyn asserted and drives the next byte enables (Byte Enable 3) on cben[3:0]. It signals the end of the burst when it de-asserts framen.<br>The Core keeps trdyn asserted and puts Data 3 on ad[31:0] since it latched it with Data 2.<br>The Core de-asserts lt_hdata_xfern. If both irdyn and lt_rdyn are asserted on the previous cycle, the Core asserts lt_ldata_xfern to the back-end. With lt_hdata_xfern asserted on the previous cycle, the back-end can increment the address counter and put the next QWORD (Don't care and Don't care) on l_ad_in. |
| 9 | Turn around | The master relinquishes control of framen and cben[3:0]. It de-asserts irdyn if both trdyn and irdyn were asserted for the last cycle.<br>The PCI IP core relinquishes control of ad[31:0]. It de-asserts both devseln and trdyn. If both trdyn and irdyn were asserted last cycle. The PCI IP core also signals to the back-end application that the transaction is complete by clearing bar_hit. The target de-asserts lt_hdata_xfern. |
| 10 | Idle | The Core relinquishes control of devseln and trdyn. |

Figure 2-39 and Table 2-44 illustrate a burst transaction to a 32-bit PCI IP core with a 64-bit Local Interface. The figure shows how the PCI interface correlates to the Local Interface. The table gives a clock-by-clock description of each event illustrated in the figure.

***Figure 2-39. 32-bit Target Burst Write Transaction With a 64-bit Local Interface***

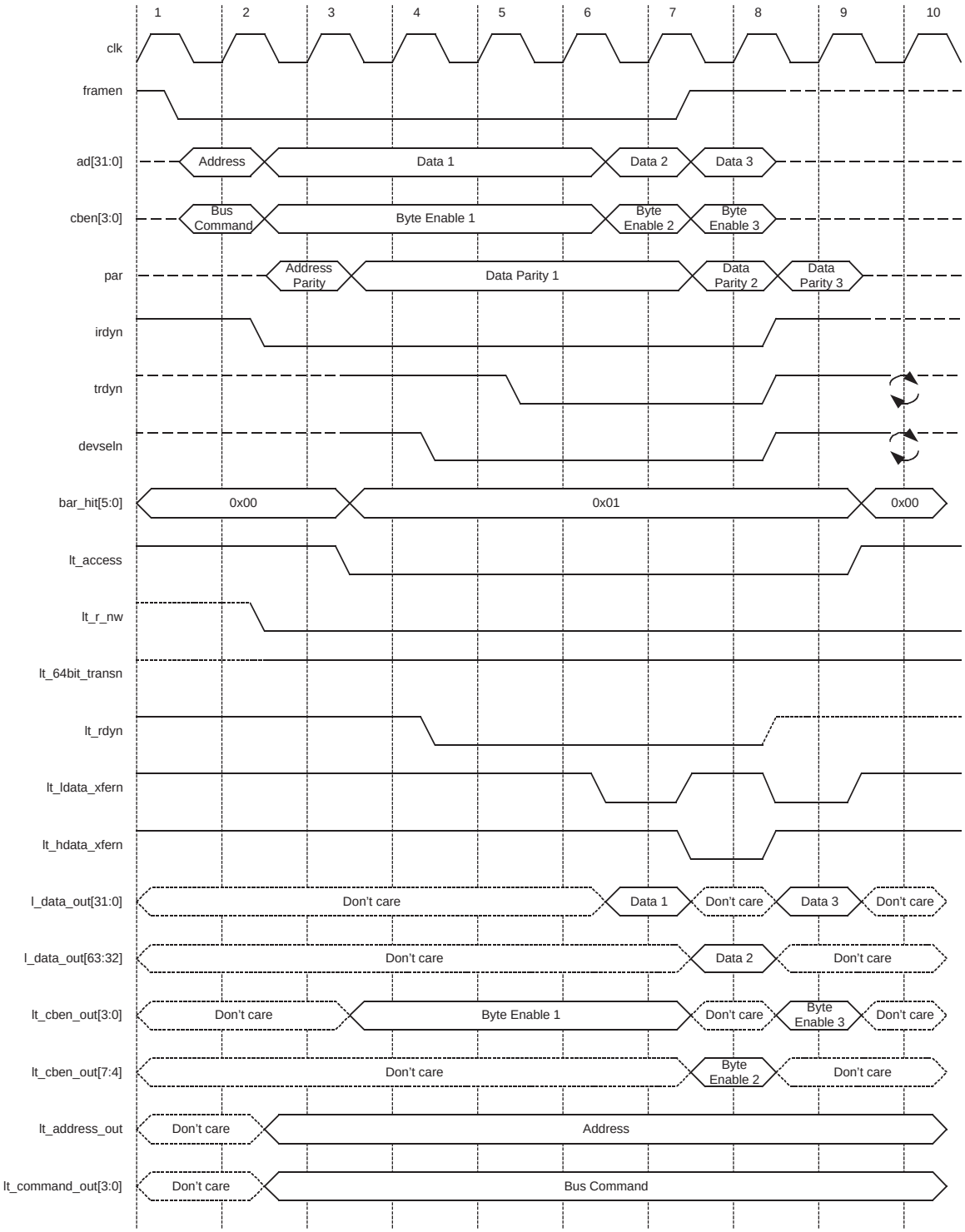


*Table 2-44. 32-bit Target Burst Write Transaction With a 64-bit Local Interface*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Address | The master asserts framen and drives ad[31:0] and cben[3:0]. |
| 2 | Wait | The PCI master drives the first byte enable (Byte Enable 1) on cben[3:0]. If it is ready to write data, it asserts irdyn and drives the first DWORD (Data 1) on ad[31:0].The Core starts to decode the address and command. It drives the lt_address_out to the back-end. |
| 3 | Wait | If there is an address match, the Core drives the bar_hit signals to the back-end application. The back-end can use bar_hit as a chip select. |
| 4 | Wait | If the DEVSEL_TIMING is set to slow, the Core asserts devseln on the clock after bar_hit. If the back-end will be ready to write data in two cycles, it can assert lt_rdyn. |
| 5 | Data 1 | trdyn is asserted since lt_rdyn was asserted the previous cycle. |
| 6 | Data 2 | Quad Word Aligned<br>The Core keeps trdyn asserted and puts Data 1 on the lower DWORD of lt_data_out.<br>If both irdyn and trdyn are asserted on the previous cycle, the master drives the next byte enable (Byte Enable 2) on cben[3:0]. If the PCI master is still ready to write data, it keeps irdyn asserted and drives the next DWORD (Data 2) on ad[31:0].<br>If both irdyn and trdyn were asserted on the previous cycle, the Core asserts lt_ldata_xfern to the back-end to signify that Data 1 is valid. With lt_ldata_xfern asserted, the back-end doesn't write the data or increment the address counter.<br>Double Word Aligned<br>The Core keeps trdyn asserted and puts Data 1 on the upper DWORD of lt_data_out.<br>If both irdyn and trdyn are asserted on the previous cycle, the master drives the next byte enables (Byte Enable 2) on cben[3:0]. If the master is still ready to write data, it keeps irdyn asserted and drives the next DWORD (Data 2) on ad[31:0].<br>If irdyn, trdyn and lt_rdyn are asserted on the previous cycle, the Core asserts lt_hdata_xfern to the back-end to signify that Data 1 is valid. With lt_hdata_xfern asserted, the back-end can safely write the QWORD (Don't care and Data 1) and increment the address counter. |
| 7 | Data 3 | Quad Word Aligned<br>The back-end puts Data 2 on lt_data_out. If the back-end keeps lt_rdyn asserted on the previous cycle, the Core keeps trdyn asserted.<br>If both irdyn and trdyn are asserted on the previous cycle, the master drives the next byte enables (Byte Enable 3) on cben[3:0]. If the master is still ready to write data, it keeps irdyn asserted and drives the next DWORD (Data 3) on ad[31:0]. The master signals the end of the burst when it de-asserts framen.<br>The Core de-asserts lt_ldata_xfern. If irdyn, trdyn and lt_rdyn are asserted on the previous cycle, the Core asserts lt_hdata_xfern to the back-end to signify that Data 2 is valid. With lt_hdata_xfern asserted, the back-end can safely write the QWORD (Data 1 and Data 2) and increment the address counter.<br>Double Word Aligned<br>The Core keeps trdyn asserted and puts Data 2 on lt_data_out.<br>If both irdyn and trdyn are asserted on the previous cycle, the master drives the next byte enables (Byte Enable 3) on cben[3:0]. If the master is still ready to write data, it keeps irdyn asserted and drives the next DWORD (Data 3) on ad. The master signals the end of the burst when it de-asserts framen.<br>The Core de-asserts lt_hdata_xfern. If both irdyn and trdyn are asserted on the previous cycle, the Core asserts lt_ldata_xfern to the back-end to signify Data 2 is valid. With lt_ldata_xfern asserted, the back-end doesn't write the data or increment the address counter. |

*Table 2-44. 32-bit Target Burst Write Transaction With a 64-bit Local Interface (Continued)*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 8 | Turn around | Quad Word Aligned<br>The Core puts Data 3 on lt_data_out.<br>If both irdyn and trdyn are asserted on the previous cycle, the master relinquishes control of framen, ad and cben. It also de-asserts irdyn if both trdyn and irdyn were asserted last cycle.<br>The Core de-asserts lt_hdata_xfern. If both irdyn and trdyn are asserted on the previous cycle, the Core asserts lt_ldata_xfern to the back-end to signify Data 3 was transferred successfully. With lt_ldata_xfern asserted, the back-end doesn't write the data yet nor increment the address counter. It de-asserts both devseln and trdyn if both trdyn and irdyn were asserted last cycle. |
| 9 | Cleanup | The Core de-asserts lt_ldata_xfern. If lt_rdyn is asserted on the previous cycle, the Core asserts lt_hdata_xfern to signify to the back-end that it can safely write the QWORD (Data 3 and Don't care). |
| 10 | Idle | The Core signals to the back-end that the transaction is complete by clearing bar_hit. It also de-asserts lt_hdata_xfern.The target relinquishes control of devseln and trdyn. |

## Dual Address Cycle (DAC)

The PCI master uses a Dual Address Cycle (DAC) to inform the PCI IP core, operating as a target, that it is using 64-bit addressing with two back-to-back address phases. The PCI IP core can respond to 64-bit addressing when the memory address being accessed is over the 4GB limit. 64-bit addressing is not restricted to only 64-bit configurations of the PCI IP core.

Figure 2-40 shows an example of the DAC during a 32-bit read transaction. Table 2-45 gives a clock-by-clock description of the dual address cycle.

***Figure 2-40. 32-Bit Target Dual Address Cycle***

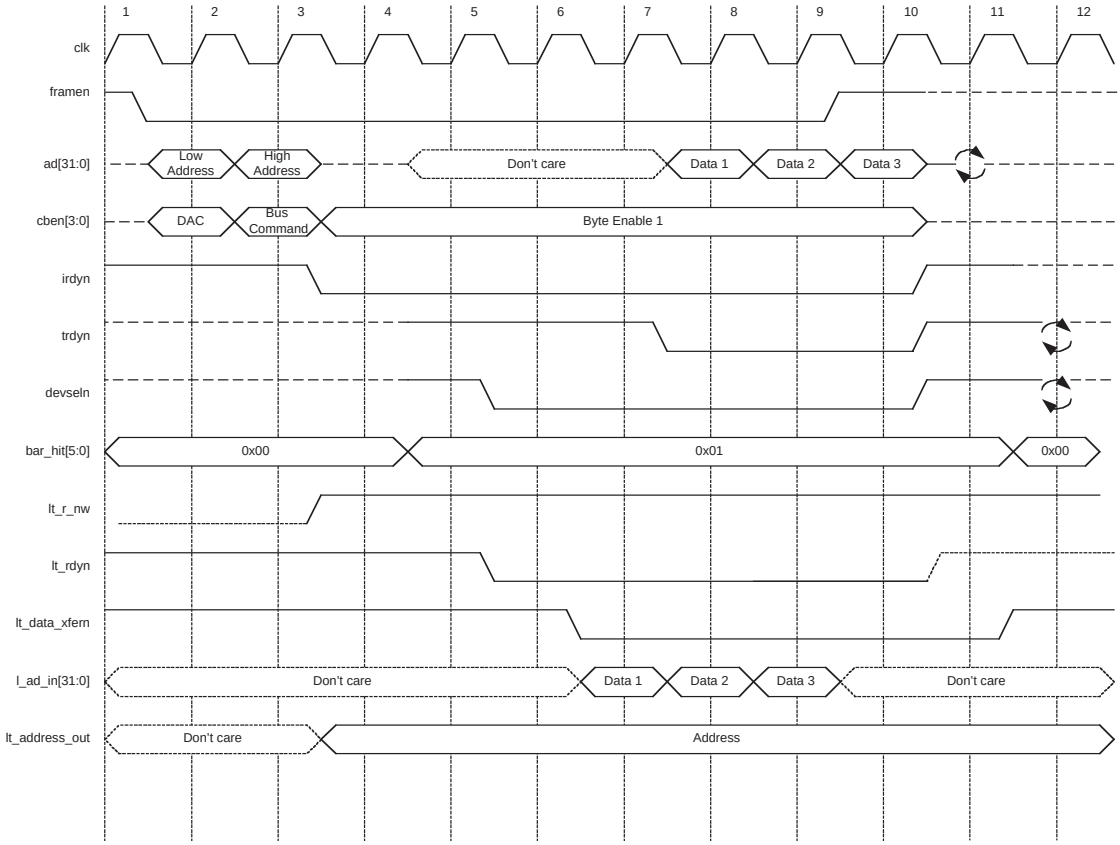


*Table 2-45. 32-Bit Target Dual Address Cycle*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Address | The master asserts framen and drives ad[31:0] (lower address and cben[3:0] (DAC). |
| 2 | Address | The master drives the ad[31:0] (higher address) and cben[3:0] (Bus command). |
| 3 | Turn around | The master tri-states ad[31:0] and drives the first byte enables (Byte Enable 1) on cben[3:0]. If the master is ready to receive data, it asserts irdyn.<br>The Core starts to decode the address and command. The Core drives the lt_address_out to the back-end application. |
| 4 | Wait | If there is an address match, the Core drives the bar_hit signals to the back-end. The back-end can use bar_hit as a chip select. |
| 5 | Wait | If the DEVSEL_TIMING is set to slow, the Core asserts devseln on the clock after bar_hit. If the back-end will be ready to put data out on the next cycle, it asserts lt_rdyn. |
| 6 | Wait | The Core asserts lt_data_xfern since lt_rdyn was asserted on the previous cycle. The back-end drives the first DWORD (Data 1) on l_ad_in. |
| 7 | Data 1 | With lt_rdyn asserted previous two cycles, the PCI IP core asserts trdyn and puts Data 1 on ad[31:0].<br>The Core asserts lt_data_xfern since lt_rdyn was asserted on the previous cycle. The back-end drives the second DWORD (Data 2) on l_ad_in. |
| 8 | Data 2 | With lt_rdyn asserted previous two cycles, the PCI IP core asserts trdyn and puts Data 1 on ad[31:0].<br>The Core asserts lt_data_xfern since lt_rdyn was asserted on the previous cycle. The back-end drives the second DWORD (Data 3) on l_ad_in. |
| 9 | Data3 | The master asserts irdyn and de-asserts framen.<br>With lt_rdyn asserted previous two cycles, the PCI IP core asserts trdyn and puts Data 3 on ad[31:0]. |
| 10 | Turn around | The master relinquishes control of framen and cben[3:0]. It de-asserts irdyn if both trdyn and irdyn were asserted last cycle.<br>The Core relinquishes control of ad. It de-asserts both devseln and trdyn if both trdyn and irdyn were asserted last cycle. The Core also signals to the back-end that the transaction is complete by clearing bar_hit. The Core de-asserts lt_data_xfern. |
| 11 | Idle | The Core relinquishes devseln and trdyn. |

## Fast Back-to-Back Transactions

The PCI IP core, as a target, can respond to a fast back-to-back transaction if a PCI master wants to perform two or more consecutive transactions to the PCI IP core. The fast back-to-back transaction consists of two or more complete PCI transactions without an idle state between them. Figure 2-41 and Table 2-46 illustrate a fast back-to-back write transaction. The figure illustrates how the PCI interface correlates to the Local Interface. The table explains each event in the figure with a clock-by-clock description.

***Figure 2-41. 32-bit Target Fast Back-to-Back Transaction***

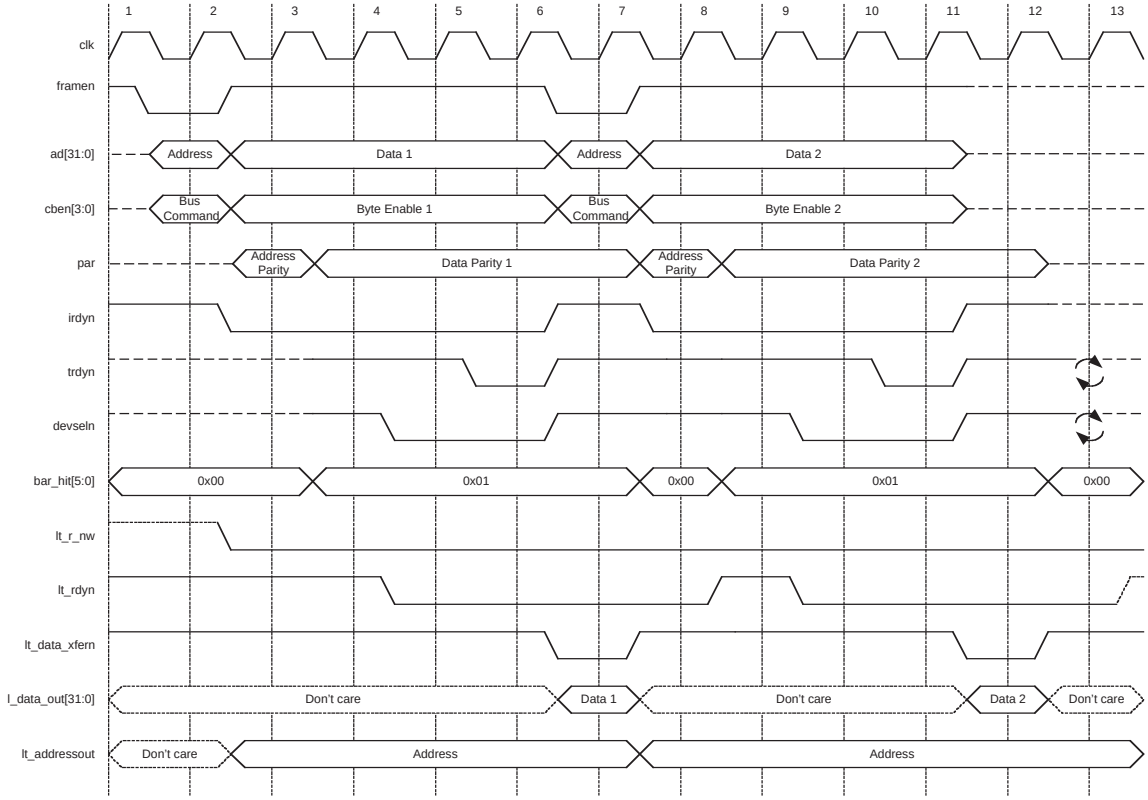



***Table 2-46. 32-bit Target Fast Back-to-Back Transaction***

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Address | The PCI master asserts `framen` and drives `ad[31:0]` and `cben[3:0]`. |
| 2 | Wait | The PCI master drives the first byte enables (Byte Enable 1) on `cben[3:0]`. If the master is ready to write data, it asserts `irdyn` and drives the first DWORD (Data 1) on `ad[31:0]`.<br>The Core starts to decode the address and command. The target drives the `lt_address_out` to the back-end application. |
| 3 | Wait | If there is an address match, the Core drives the `bar_hit` signals to the back-end. The back-end application can use `bar_hit` as a chip select. |
| 4 | Wait | If the DEVSEL_TIMING is set to slow, the Core asserts `devseln` on the clock after `bar_hit`. If the back-end will be ready in two cycles to write data, it can assert `lt_rdyn`. |
| 5 | Data 1 | `trdyn` is asserted since `lt_rdyn` was asserted the previous cycle. |
| 6 | Address | The master asserts `framen` and drives the `ad[31:0]` and `cben[3:0]`.<br>If both `irdyn` and `lt_rdyn` are asserted on the previous cycle, the Core asserts `lt_data_xfern` to the back-end to signify Data 1 is valid. With `lt_data_xfern` asserted the back-end can safely write Data 1. |
| 7 | Wait | The PCI master drives the first byte enables (Byte Enable 1) on `cben[3:0]`. If the master is ready to write data, it asserts `irdyn` and drives the first DWORD (Data 1) on `ad[31:0]`.<br>The Core starts to decode the address and command. The Core drives the `lt_address_out` to the back-end. |
| 8 | Wait | If there is an address match, the Core drives the `bar_hit` signals to the back-end. The back-end can use `bar_hit` as a chip select. |

*Table 2-46. 32-bit Target Fast Back-to-Back Transaction (Continued)*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 9 | Wait | If the DEVSEL_TIMING is set to slow, the Core asserts devseln on the clock after bar_hit. If the back-end will be ready to write data in two cycles, it can assert lt_rdyn. |
| 10 | Data 1 | trdyn is asserted since lt_rdyn was asserted the previous cycle. |
| 12 | Termination | If both irdyn and trdyn are asserted on the previous cycle, the master relinquishes control of framen, ad[31:0] and cben[3:0]. It also de-asserts irdyn if both trdyn and irdyn were asserted last cycle. |
| 13 | Idle | The Core relinquishes devseln and trdyn. |

## Advanced Configuration Accesses

Advanced Configuration Space read accesses are very similar to the 32-bit target read transactions with additional PCI and Local bus signals. Figure 2-42 and Table 2-47 illustrate advanced Configuration Space read transactions.

***Figure 2-42. Advanced Configuration Read Transaction***

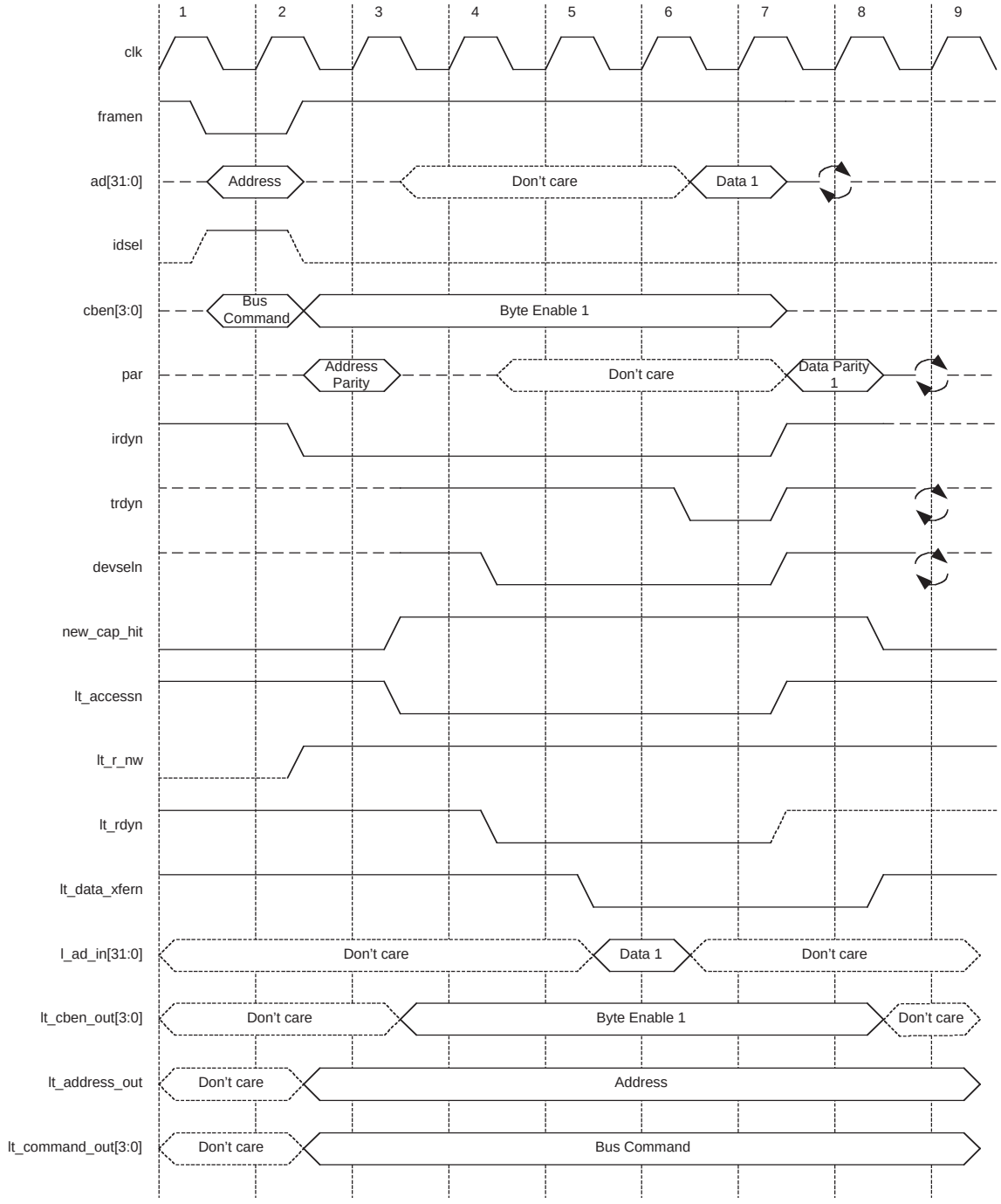


*Table 2-47. Advanced Configuration Read Transactions*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Address | The master asserts framen and idsel. It and drives the configuration address on ad[31:0] and the read command on cben[3:0]. |
| 2 | Turn around | The master tri-states the ad[31:0] lines and drives the first byte enables cben[3:0]. If the master is ready to receive data, it asserts irdyn.<br>The Core starts to decode the address and command. The target drives the lt_address_out to the back-end. |
| 3 | Wait | If there is an address match, the Core drives the new_cap_hit signals to the back-end. The back-end can use the new_cap_hit as a chip select. |
| 4 | Wait | If the device select timing is set to slow, the Core asserts devseln on the clock after new_cap_hit. If the back-end is ready to put data out on the next cycle, it can assert lt_rdyn. |
| 5 | Wait | The Core asserts lt_data_xfern since lt_rdyn was asserted the previous cycle. The back-end drives the first DWORD (Data 1) on l_ad_in. |
| 6 | Data 1 | With lt_rdyn asserted for the previous two cycles, the Core asserts trdyn and puts Data 1 on ad[31:0]. |
| 7 | Turn around | The master relinquishes control of framen and cben[3:0]. It de-asserts irdyn if both trdyn and irdyn were asserted on the last cycle.<br>The Core relinquishes control of ad[31:0]. It de-asserts both devseln and trdyn if both trdyn and irdyn were asserted on the last cycle. The Core also signals to the back-end that the transaction is complete by clearing new_cap_hit. The Core de-asserts lt_data_xfern. |
| 8 | Idle | The Core relinquishes devseln and trdyn. |

Advanced Configuration Space write accesses are similar to the 32-bit target write transactions with additional PCI and Local bus signals. Figure 2-43 and Table 2-48 illustrate advanced Configuration Space write transactions.

***Figure 2-43. Advanced Configuration Write Transaction***

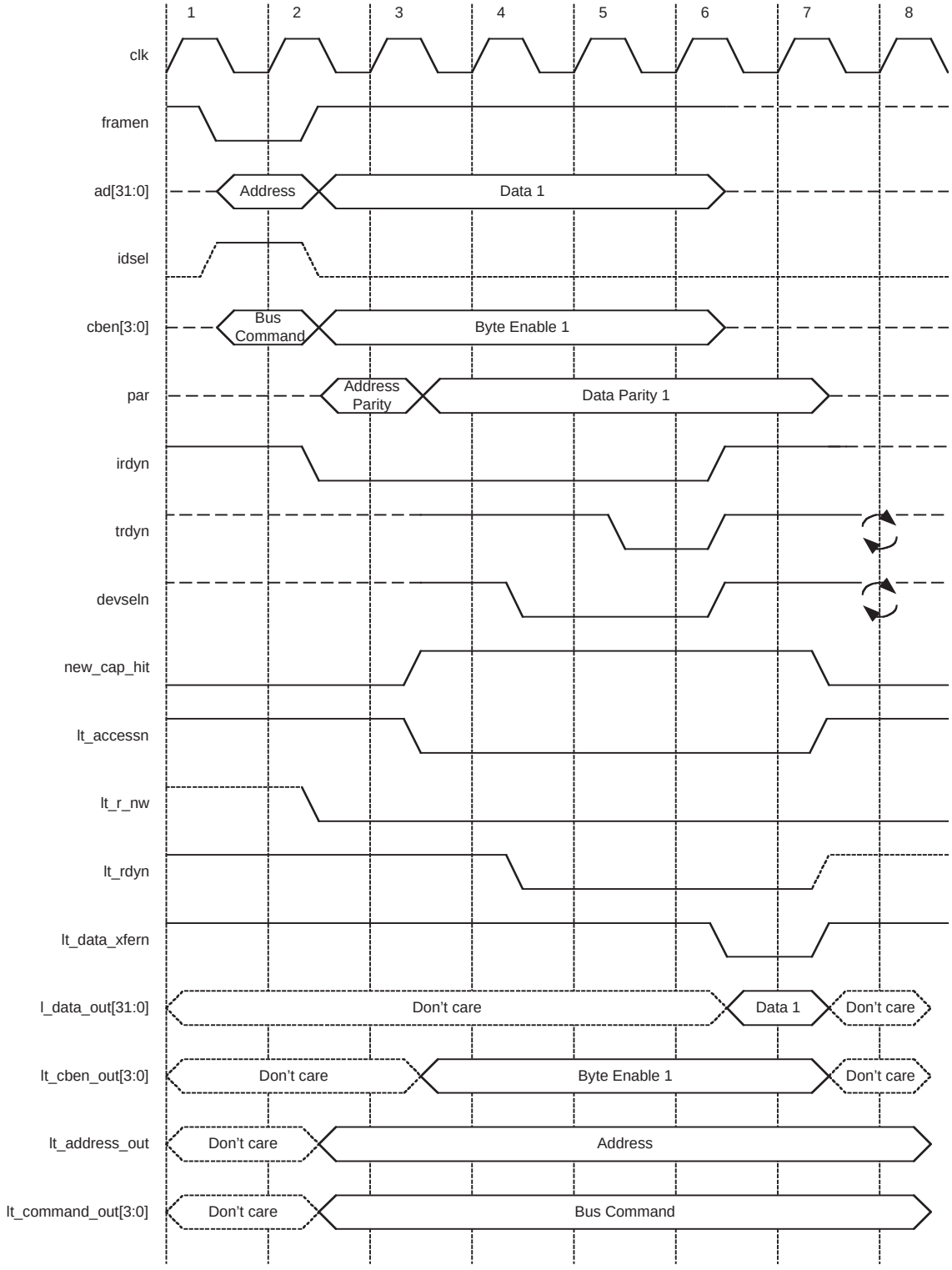


*Table 2-48. Advanced Configuration Write Transaction*

| CLK | PCI Data Phase | Description |
|---|---|---|
| 1 | Address | The master asserts framen and idsel. It and drives the configuration address on ad[31:0] and the read command on cben[3:0]. |
| 2 | Wait | The PCI master drives the first byte enables (Byte Enable 1) on cben[3:0]. If it is ready to write data, it asserts irdyn and drives the first DWORD (Data 1) on ad[31:0].The Core starts to decode the address and command. |
| 3 | Wait | If there is an address match, the Core drives the new_cap_hit signals to the back-end. The back-end can use new_cap_hit as a chip select. |
| 4 | Wait | If the DEVSEL_TIMING is set to slow, the Core asserts devseln on the clock after new_cap_hit. If the back-end is ready to write data in two cycles, it can assert lt_rdyn. |
| 5 | Data 1 | trdyn is asserted since lt_rdyn was asserted the previous cycle. |
| 6 | Turn around | If both irdyn and trdyn are asserted on the previous cycle, the master relinquishes control of framen, ad[31:0] and cben[3:0]. It also de-asserts irdyn if both trdyn and irdyn were asserted last cycle.<br>It de-asserts both devseln and trdyn if both trdyn and irdyn were asserted last cycle. |
| 7 | Idle | The Core signals to the back-end that the transaction is complete by clearing new_cap_hit. It also de-asserts lt_data_xfern.<br>The Core relinquishes devseln and trdyn. |

## Target Termination

The back-end application has full control over the termination of all PCI transactions requesting information from the Local Interface. The back-end application must handle the termination properly. The four types of target-initiated termination are:

- Retry
- Target Abort
- Disconnect With Data
- Disconnect Without Data

The back-end application utilizes the Local Interface signals lt_abortn, lt_disconnectn, and lt_rdyn to initiate termination. These signals control the target's response and termination of PCI transactions that request data from the Local Interface. Table 2-49 shows a summary of the different target initiated termination types.

In order to prevent a PCI IP core from monopolizing the PCI bus, the *PCI Local Bus Specification, Revision 3.0* includes limitations on the amount of transferring time for a target. During the initial data phase, the target must issue a Retry if it cannot respond within 16 clocks of framen being asserted. For subsequent data phases following the initial data phase, the PCI IP core must respond within eight clock cycles or issue a Disconnect Without Data or a Target Abort. The first option is preferred. The different target initiated termination sequences are discussed in the following section.

*Table 2-49. Target Initiated Termination Summary*

| Termination Type | lt_rdyn | lt_disconnectn | lt_abortn | Comments |
|---|---|---|---|---|
| Disconnect With Data | Asserted | Asserted | De-asserted | Some data is transferred. This includes the current DWORD or QWORD. If the PCI master needs to transfer more data, the transaction is re-initiated using the next address. |
| Disconnect Without Data | De-asserted | Asserted | De-asserted | Some data is transferred but not the current DWORD or QWORD. This occurs in a data phase other than the first phase. The master may resume the transaction or not. If resumed, it starts on the same address. |
| Retry | De-asserted | Asserted | De-asserted | No data is transferred. This occurs during the first data phase. Master may or may not try the same transaction. |
| Target Abort | Don't Care | Don't Care | Asserted | Indicates a fatal error. Data is disregarded. |

## Disconnect With Data

A Disconnect With Data occurs after at least one DWORD or QWORD has been transferred. The difference between a Disconnect With Data and a Disconnect Without Data depends on the state of `lt_rdyn` when the Local Interface requests a disconnect using the `lt_disconnectn` signal. This condition indicates if the bus transaction is terminated before or after the completion of the current data phase. Once the current data phase is completed, the bus transaction is terminated with a Disconnect With Data. Figure 2-44 and Table 2-50 show a Disconnect With Data on a read transaction.

Below is a list of the reasons for the PCI IP core to perform a Disconnect With Data:

- Target is slow to complete subsequent data phase
- Target does not support requested burst mode
- Memory target does not understand addressing sequence
- Transfer crosses over target's address boundary

***Figure 2-44. 32-bit Target Disconnect with Data for Read Transaction***

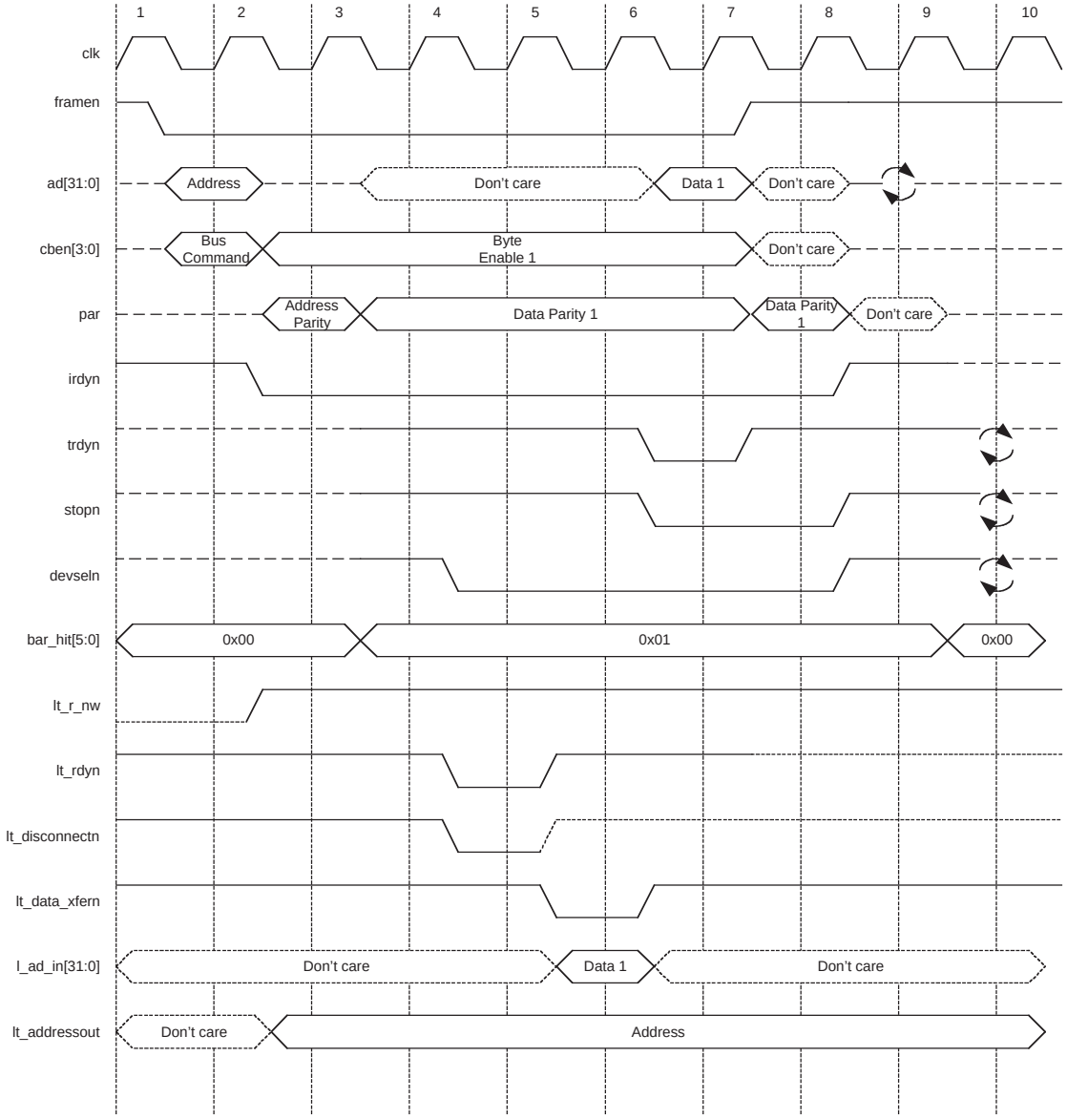



***Table 2-50. 32-bit Target Disconnect with Data for Read Transaction***

| CLK | Description |
|---|---|
| 4 | The devseln signal is driven low to indicate that the PCI IP core is selected for the transaction. The lt_rdyn signal is driven low to indicate that the back-end application is ready to provide data on the next clock cycle. Because the target cannot complete any more PCI data phases, the lt_disconnectn signal is also driven low. |
| 5 | The lt_data_xfern signal is driven low by the PCI IP core to the back-end to indicate that data in available on l_ad_in. |
| 6 | The trdyn and stopn signals are driven low because both lt_rdyn and lt_disconnectn were driven low during the previous two clock cycles. The lt_data_xfern signal is de-asserted because lt_rdyn was de-asserted during the previous cycle.Data 1 is presented on the PCI bus via ad[31:0] |
| 7 | The PCI master de-asserts framen to acknowledge the disconnection initiated by the target.The PCI IP core de-asserts trdyn since the completion of the last PCI data phase and the assertion of stopn. |

*Table 2-50. 32-bit Target Disconnect with Data for Read Transaction (Continued)*

| CLK | Description |
|---|---|
| 8 | De-asserting `irdyn` disconnects the PCI master. De-asserting `devseln` and `stopn`. Disconnects the PCI IP core from the PCI bus. |
| 9 | The larger relinquishes `devseln`, `stopn` and `trdyn`. |

Figure 2-45 and Table 2-51 illustrate a Disconnect With Data on a write transaction.

*Figure 2-45. 32-bit Target Disconnect with Data for Write Transaction*

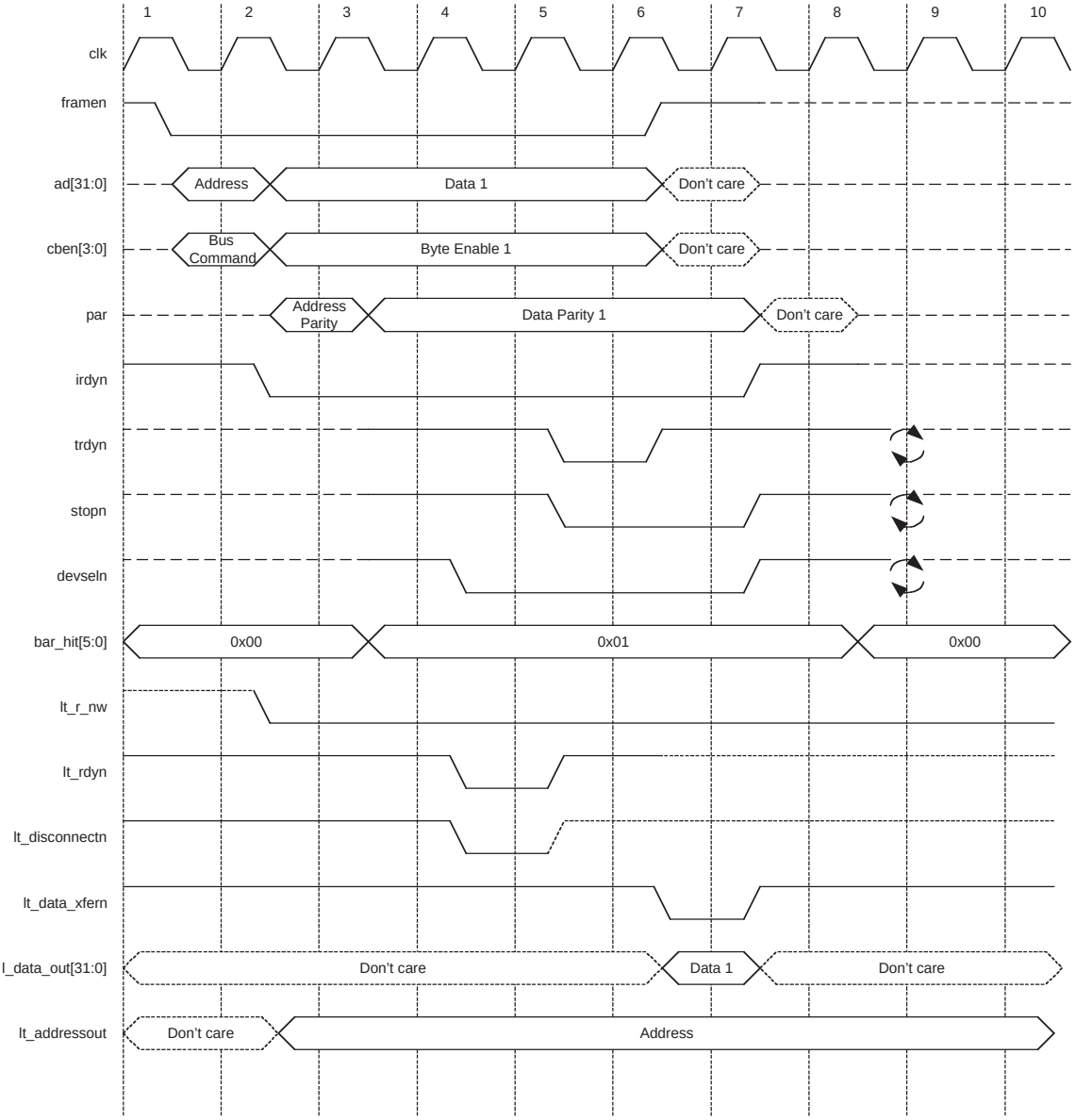


*Table 2-51. 32-bit Target Disconnect with Data for Write Transaction*

| CLK | Description |
|---|---|
| 4 | The devseln signal is driven low to indicate that the PCI IP core has been selected for the transaction. The lt_rdyn signal is driven low to indicate that the back-end application is ready to receive data. Because the target can not complete any more PCI data phases the lt_disconnectn signal is also driven low. |
| 5 | The trdyn and the stopn signals are driven low because both the lt_rdyn and the lt_disconnectn signals were driven low the previous cycle. |
| 6 | The target asserts lt_data_xfern to the back-end to signify Data 1 is available on the lt_data_out.<br>The PCI IP core de-asserts trdyn since the last PCI data phase was complete and the stopn was asserted.<br>The PCI master de-asserts the framen to acknowledge the disconnection initiated by the target. |
| 7 | The PCI master disconnects by de-asserting irdyn. The PCI IP core disconnects from the PCI bus by de-asserting devseln and stopn. |
| 8 | The target relinquishes devseln, stopn and trdyn. |

## Disconnect Without Data

A Disconnect Without Data occurs after at least one data DWORD or QWORD has been transferred. A Disconnect Without Data is used if the PCI IP core is incapable of completing the current PCI data phase. Figure 2-46 and Table 2-52 show a Disconnect Without Data for a read transaction.

Below is a list of the reasons that the PCI IP core may Disconnect Without Data:

- Target slow to complete subsequent data phase
- Target does not support burst mode requested
- Memory target doesn't understand addressing sequence
- Transfer crosses over target's address boundary

***Figure 2-46. 32-bit Target Disconnect Without Data for Read Transaction***

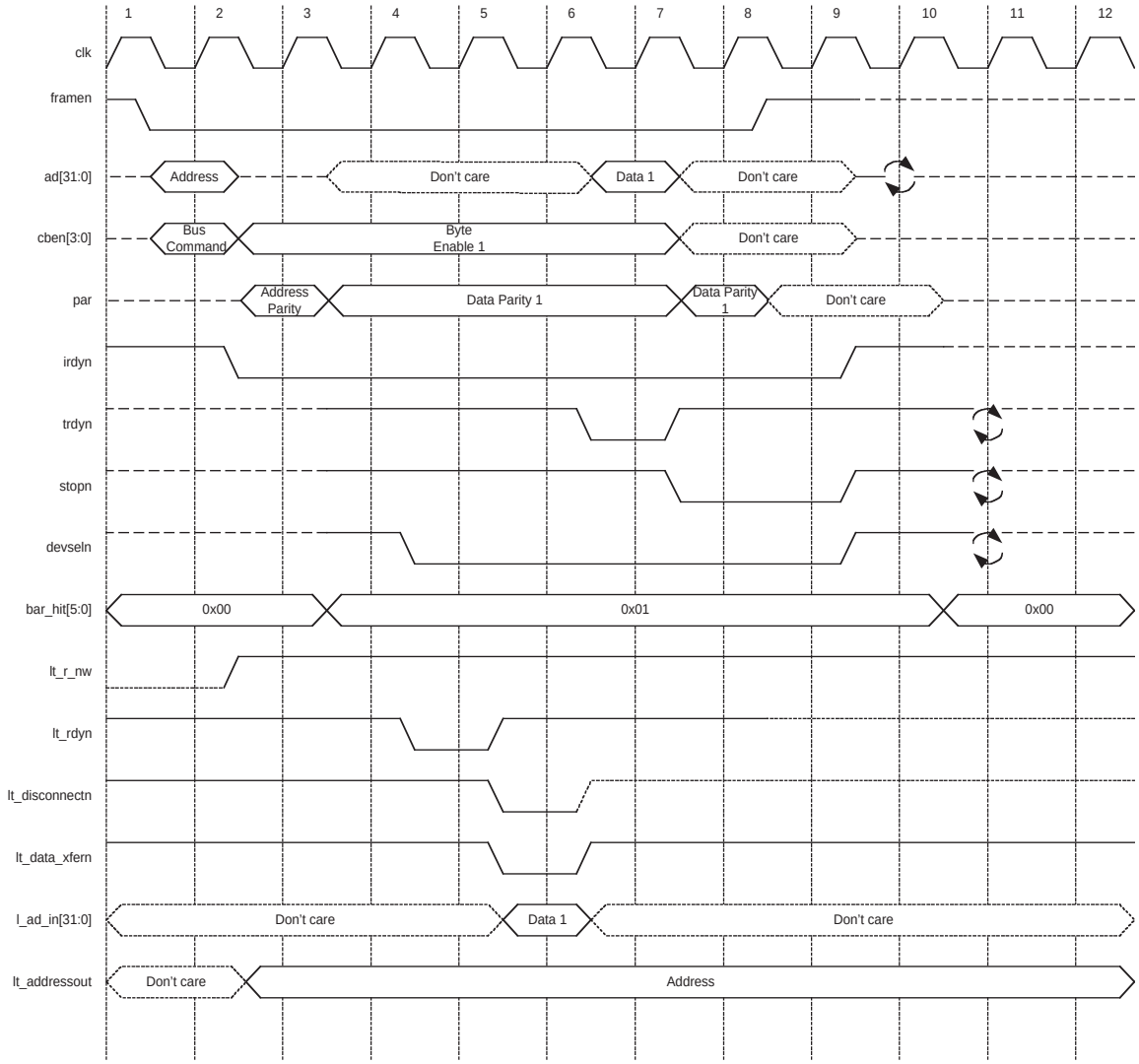



***Table 2-52. 32-bit Target Disconnect Without Data for Read Transaction***

| CLK | Description |
|---|---|
| 4 | The devseln signal is driven low to indicate that the PCI IP core is selected for the transaction. The lt_rdyn signal is driven low to indicate that the back-end application is ready to provide data on the next clock cycle. |
| 5 | The lt_data_xfern signal is driven low by the PCI IP core to indicate that data is valid on l_ad_in[31:0].Because the target can not complete any more PCI data phases the lt_rdyn signal is driven high and lt_disconnectn signals are driven low. |
| 6 | The trdyn signal is driven low because the lt_rdyn signal was driven low two clock cycle before. The lt_data_xfern signal is de-asserted because the lt_rdyn signal was de-asserted in the previous cycle.Data 1 is presented to the PCI bus via ad[31:0]. |
| 7 | The trdyn signal is de-asserted since the lt_rdyn signal was driven high two clock cycles before. The stopn signal is asserted since the lt_disconnectn signal was driven low two clock cycles before. |
| 8 | The PCI master de-asserts the framen to acknowledge the disconnection initiated by the target. |
| 9 | The PCI master disconnects by de-asserting the irdyn. The PCI IP core disconnects from the PCI bus by de-asserting the devseln and stopn. |

Figure 2-47 and Table 2-53 illustrate a Disconnect Without Data for a write transaction.

***Figure 2-47. 32-bit Target Disconnect Without Data for Write Transaction***

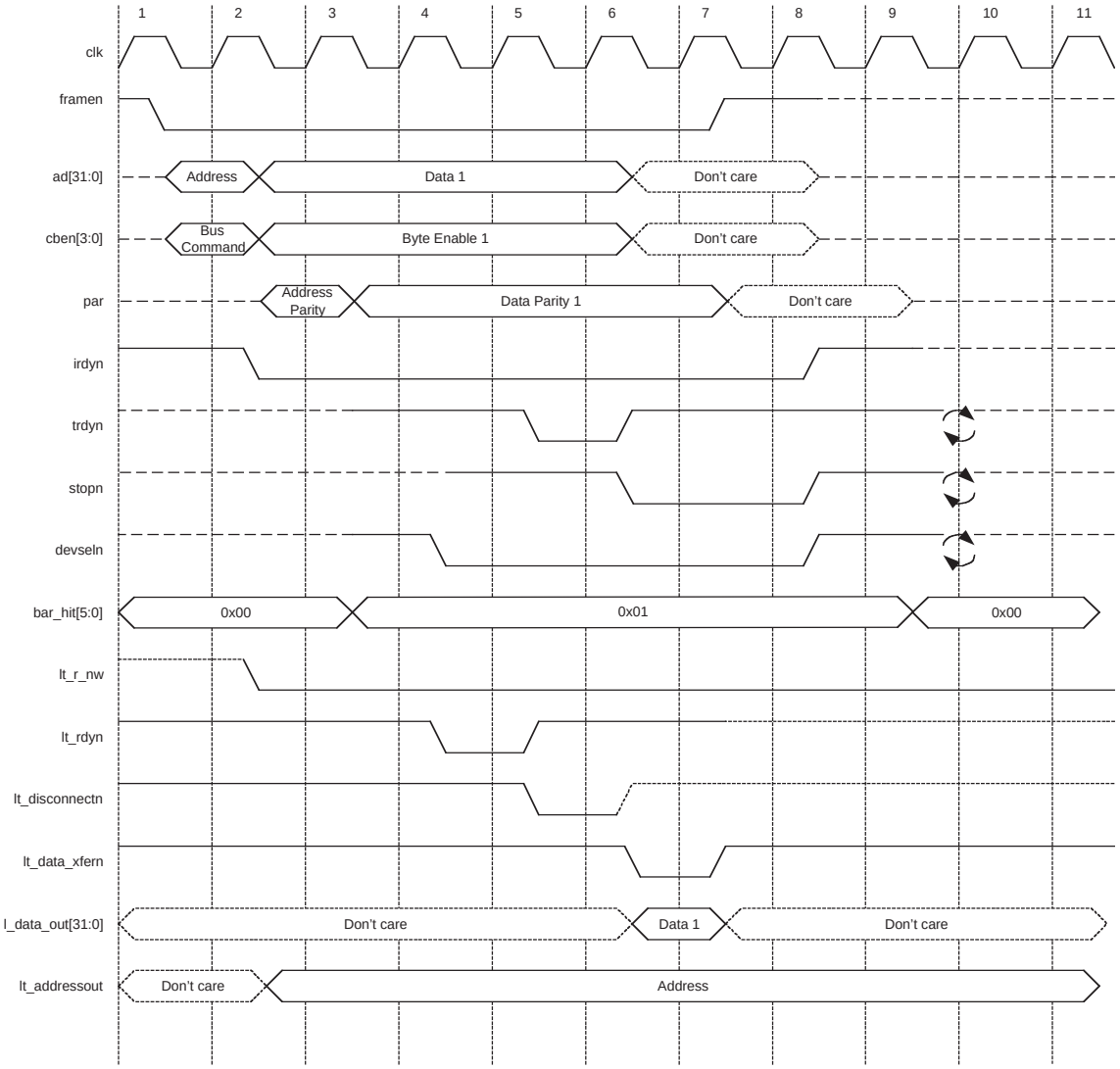



***Table 2-53. 32-bit Target Disconnect Without Data for Write Transaction***

| CLK | Description |
|---|---|
| 4 | The devseln signal is driven low to indicate that the PCI IP core is selected for the transaction. The lt_rdyn signal is driven low to indicate that the back-end application is ready to receive data. |
| 5 | The trdyn signal is driven low because the lt_rdyn signal was driven low on the previous clock cycle.<br>Data 1 is presented to the PCI bus via ad[31:0].<br>Because the target can not complete any more PCI data phases the lt_rdyn signal is driven high and lt_disconnectn signals are driven low. |
| 6 | The target asserts lt_data_xfern to the back-end to signify Data 1 is available on the lt_data_out.<br>The trdyn signal is de-asserted since the lt_rdyn signal was driven high the previous cycle. And the stopn signal is asserted since the lt_disconnectn signal was driven low the previous cycle. |
| 7 | The PCI master de-asserts the framen to acknowledge the disconnection initiated by the target. |
| 8 | The PCI IP core disconnects from the PCI bus by de-asserting the devseln and stopn. |
| 9 | Idle |

## Retry

A Retry may be necessary if the PCI IP core cannot assert the `trdyn` signal within the maximum number of clock cycles defined by the *PCI Local Bus Specification, Revision 3.0*. A Retry occurs if `lt_rdyn` is not asserted before `lt_disconnectn` is asserted. A Retry can also occur if the PCI IP core does not assert `lt_rdyn` within 16 clocks after the assertion of `framen`. Figure 2-48 and Table 2-54 show a Retry on a read transaction.

Below is a list of the reasons for a Retry.

- Target is very slow to respond to complete first data phase
- Snoop hit occurs on modified cache line
- Resource is busy

***Figure 2-48. 32-bit Target Retry for Read Transaction***

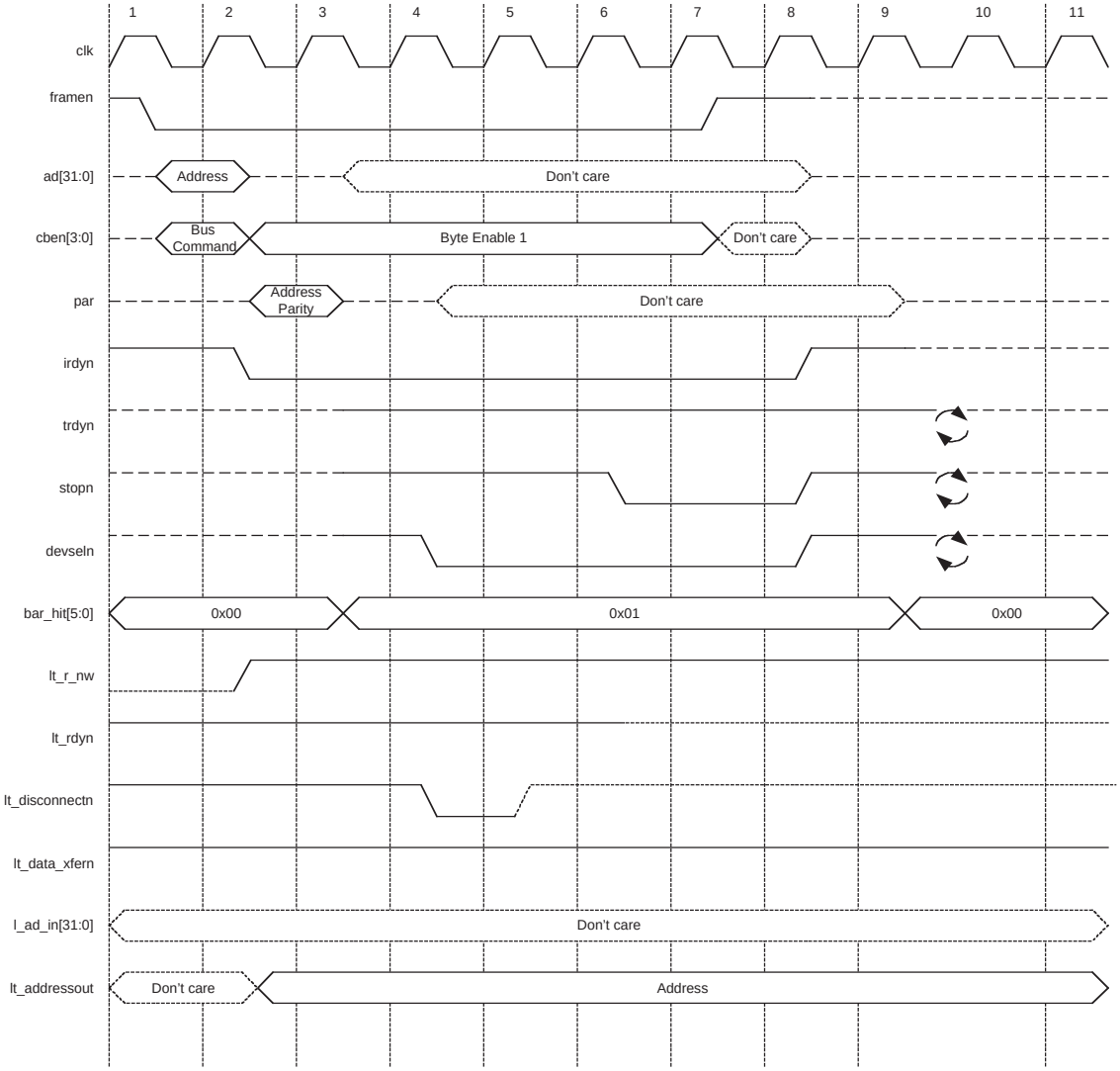


***Table 2-54. 32-bit Target Retry for Read Transaction***

| CLK | Description |
|---|---|
| 4 | The devseln signal is driven low to indicate that the PCI IP core is selected for the transaction. The lt_rdyn signal remains high to indicate that the back-end application is not ready to provide data. Because the target can not complete any PCI data phases, the lt_rdyn signal remains high and the lt_disconnectn signal is driven low. |
| 5 | The lt_data_xfern signal remains high because the lt_rdyn signal was high during the previous cycle. |
| 6 | The stopn signal is driven low on the PCI bus as the lt_disconnectn signal was driven low for the previous two clock cycles. |
| 7 | The PCI master de-asserts the framen to acknowledge the retry initiated by the target. |
| 8 | The PCI master terminates the transaction by de-asserting the irdyn. The PCI IP core de-asserts the devseln and stopn. |
| 9 | Idle |

Figure 2-49 and Table 2-55 show a Retry on a write transaction.

***Figure 2-49. 32-bit Target Retry for Write Transaction***

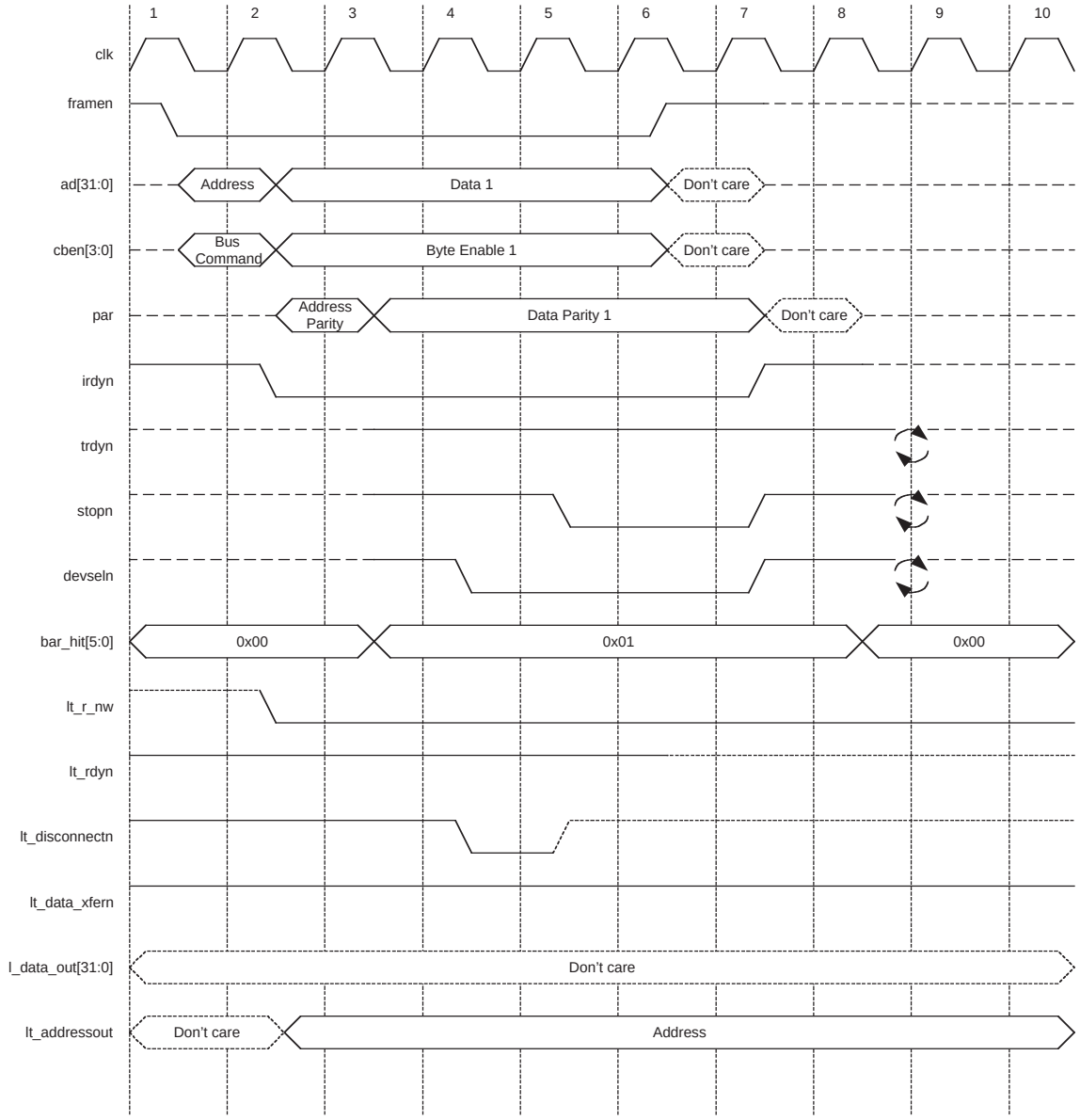



***Table 2-55. 32-bit Target Retry for Write Transaction***

| CLK | Description |
|---|---|
| 4 | The devseln signal is driven low to indicate that the PCI IP core is selected for the transaction. The lt_rdyn signal remains high to indicate that the back-end application is not ready to provide data. Because the target can not complete any PCI data phases, the lt_rdyn signal remains high and the lt_disconnectn signal is driven low. |
| 5 | The lt_data_xfern signal remains high because the lt_rdyn signal was high during the previous cycle. |
| 6 | The stopn signal is driven low on the PCI bus as the lt_disconnectn signal was driven low for the previous two clock cycles. |
| 7 | The PCI master de-asserts the framen to acknowledge the retry initiated by the target. |

*Table 2-55. 32-bit Target Retry for Write Transaction (Continued)*

| CLK | Description |
|---|---|
| 8 | The PCI master terminates the transaction by de-asserting the `irdyn`. The PCI IP core de-asserts the `devseln` and `stopn`. |
| 9 | Idle |

## Target Abort

Unlike the other types of disconnects, the state of `irdyn` does not have any effect on termination during a Target Abort. Figure 2-50 illustrates a Target Abort during a read transaction.

Some possible reasons for a target abort are:

- Broken target
- I/O addressing error
- Address phase parity error

Figure 2-50 and Table 2-56 show a target abort on a read transaction.

*Figure 2-50. 32-bit Target Abort for Read Transaction*

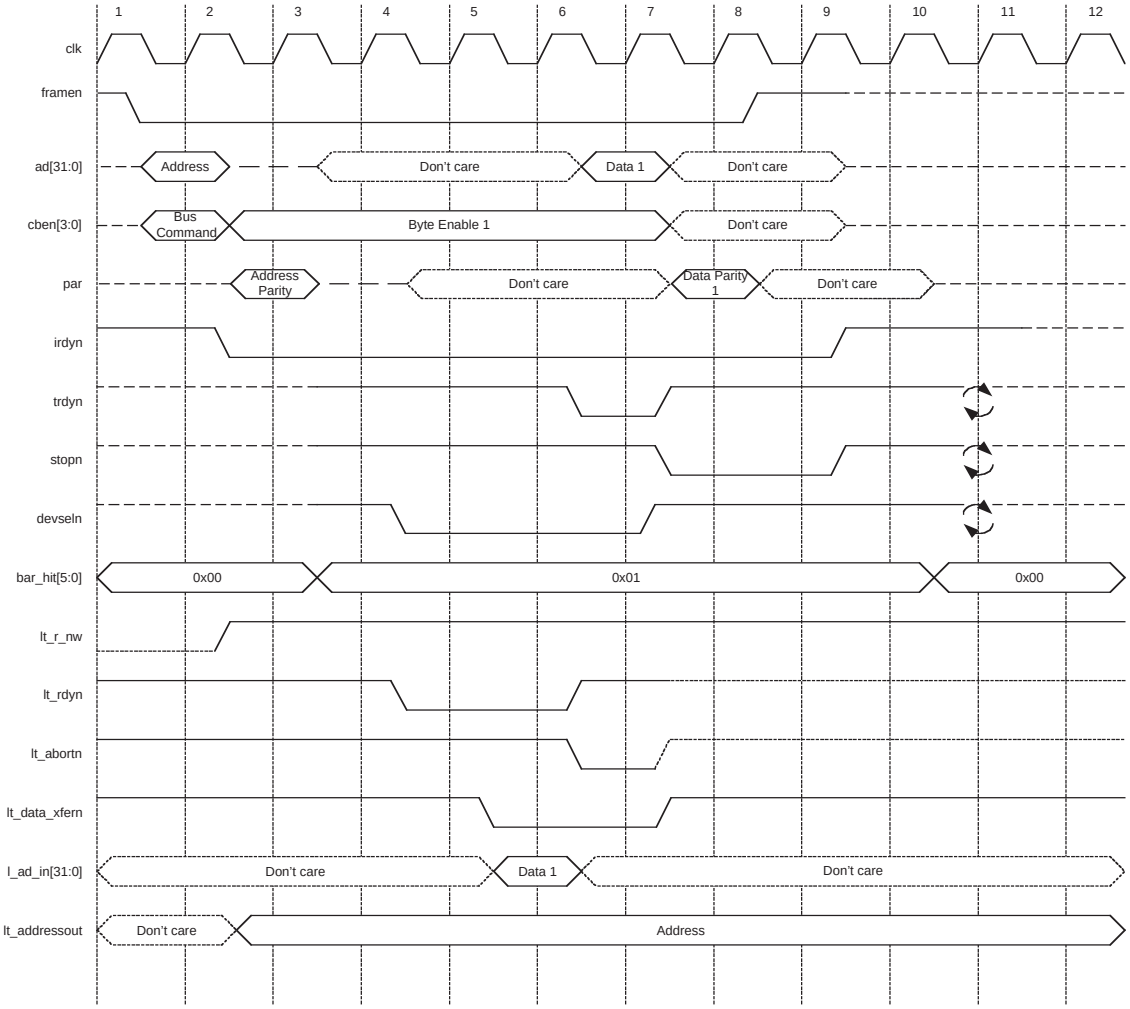


***Table 2-56. 32-bit Target Abort for Read Transaction***

| CLK | Description |
|---|---|
| 4 | The devseln signal is driven low to indicate that the PCI IP core is selected for the transaction. The lt_rdyn signal is driven low to indicate that the back-end application is ready to put data out on the next cycle. |
| 5 | The lt_data_xfern signal is driven low by the PCI IP core to the back-end to indicate that data on l_ad_in is being read.The lt_rdyn signal is driven low to indicate the back-end is ready to provide the next data. |
| 6 | The trdyn signal is driven low because the lt_rdyn signal was driven low two clock cycles before. The lt_data_xfern signal is asserted because the lt_rdyn signal was asserted in the previous cycle.<br>Data 1 is presented to the PCI bus via ad[31:0].<br>Because the back-end wants to abort the transaction the lt_abortn signal is driven low. The data on l_ad_in is also invalid. |
| 7 | A target abort is requested as the devseln and the trdyn signals are de-asserted and the stopn signal is asserted. |
| 8 | The PCI master de-asserts the framen to acknowledge the target abort. |
| 9 | The PCI master terminates the transaction by de-asserting the irdyn. The PCI IP core de-asserts the stopn. |

Figure 2-51 and Table 2-57 show a target abort on a write transaction.

***Figure 2-51. 32-bit Target Abort for Write Transaction***

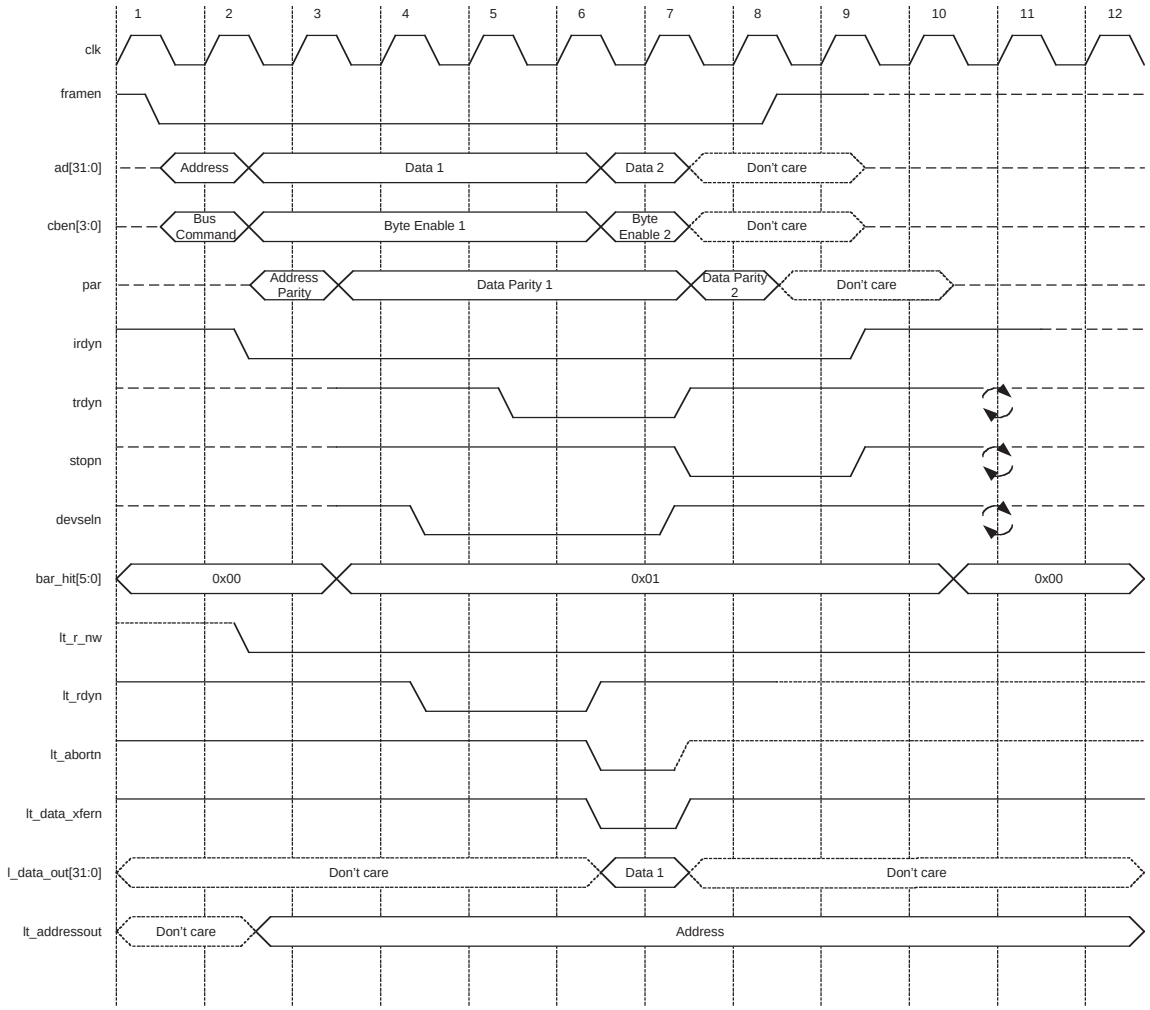


***Table 2-57. 32-bit Target Abort for Write Transaction***

| CLK | Description |
|---|---|
| 4 | The devseln signal is driven low to indicate that the PCI IP core is selected for the transaction. The lt_rdyn signal is driven low to indicate that the back-end application is ready to receive data on the next two cycles. |
| 5 | The target asserts trdyn because lt_rdyn was asserted on previous cycle. The first data phase is completed. |
| 6 | The trdyn signal remains low because the lt_rdyn signal was driven low two clock cycles before. The lt_data_xfern signal is asserted because the lt_rdyn signal was de-asserted in the previous cycle.<br>The target transfers data 1 on lt_data_out[31:0].<br>Because the back-end wants to abort the transaction the lt_abortn signal is driven low. |
| 7 | A target abort is requested as the devseln and the trdyn signals are de-asserted and the stopn signal is asserted. |
| 8 | The PCI master de-asserts the framen to acknowledge the target abort. |
| 9 | The PCI master terminates the transaction by de-asserting the irdyn. The PCI IP core de-asserts the stopn. |

# Parameter Settings

The IPexpress tool is used to create IP and architectural modules in the Diamond and ispLEVER software. Refer to "IP Core Generation" on page 143 for a description on how to generate the IP.

Table 3-1 provides the list of user configurable parameters for the PCI IP core. The parameter settings are specified using the PCI IP core Configuration GUI in IPexpress. The numerous PCI Express parameter options are partitioned across multiple GUI tabs as shown in this chapter.

***Table 3-1. Parameter Descriptions***

| Parameter | Range | Default |
|---|---|---|
| **Bus** | | |
| PCI Data Bus Size | 32, 64 | 32 |
| Local Master Data Bus Size[1] | 32, 64 | 32 |
| Local Target Data Bus Size[1] | 32, 64 | 32 |
| Local Data Bus Size[2] | 32, 64 | 32 |
| Local Address Bus Width | 32, 64 | 32 |
| Bus Speed | 33MHz, 66MHz | — |
| **Identification** | | |
| Vendor ID [15:0] | 0x 0000-FFFF | 0x 1573 |
| Device ID [15:0] | 0x 0000-FFFF | 0x 0000 |
| Subsystem Vendor ID [15:0] | 0x 0000-FFFF | 0x 0000 |
| Subsystem ID [15:0] | 0x 0000-FFFF | 0x 0000 |
| Revision ID [7:0] | 0x 00-FF | 0x 01 |
| Class Code (Base Class, Sub Class Interface) | 0x 00-FF<br>0x 00-FF<br>0x 00-FF | 0x 05<br>0x 00<br>0x 00 |
| **Options** | | |
| Timing of DEVSEL | slow | slow |
| Expansion ROM | Yes, No | No |
| Address Space Size | None, 2k, 4k, 8k, ... , 16M | None |
| Capabilities Pointer [7:0] | {Yes, No} 0x 00-FF | Yes 0x 40 |
| CardBus CIS Pointer [31:0] | 0x 00000000 - 0x FFFFFFFF | 0x 00000000 |
| Fast Back to Back | Enable, Disable | Enable |
| Interrupt Acknowledge | Yes, No | Yes |
| Interrupt Pin | None, INTAN | INTAN |
| **PCI Master**[1] | | |
| Read Only Latency Timer[1] | Yes, No | No |
| MIN_GNT[1] | 0x 00-FF | 0x 00 |
| MAX_LAT[1] | 0x 00-FF | 0x 00 |
| **BARs** | | |
| Number of BARs | 0 - 6 | 3 |
| BAR0 | 0x 00000000 - 0x FFFFFFFF | 0x FFFFFFFD |
| BAR1 | 0x 00000000 - 0x FFFFFFFF | 0x FFFFFFF0 |

*Table 3-1. Parameter Descriptions (Continued)*

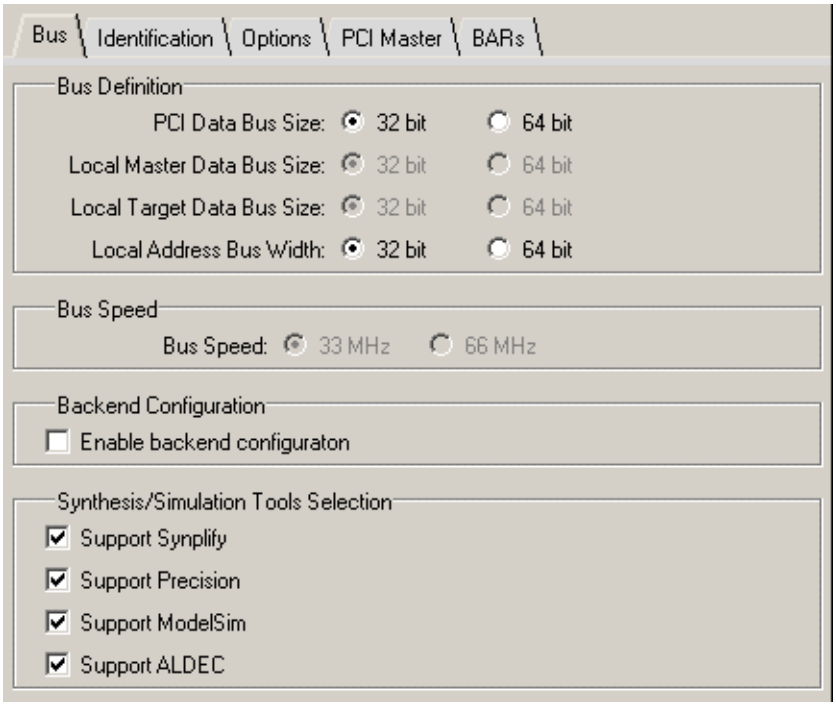
| Parameter | Range | Default |
|---|---|---|
| BAR2 | 0x 00000000 - 0x FFFFFFFF | 0x FFFFFFF0 |
| BAR3 - BAR5 | 0x 00000000 - 0x FFFFFFFF | 0x 00000000 |
| **BAR0 to BAR5 Configuration Options** | | |
| BAR width | 32, 64 | 32 |
| BAR Type | Memory, I/O | I/O for BAR0, Memory for BAR1, BAR2 |
| Address Space Size | None, 4 bytes, 8 bytes, ... , 8G | 4 bytes for BAR0 16 bytes for BAR1, BAR2 |
| Prefetching Enable | Yes, No | No |

1. Only for PCI Master/Target Core.
2. Only for PCI Target Core.

# Bus Tab

Figure 3-1 shows the contents of the Bus tab. This example shows the PCI Master/Target 33.

***Figure 3-1. Bus Tab***




## Bus Definition

### PCI Data Bus Size

The address and data width on the PCI side.

### Local Master Data Bus Size (Master/Target cores only)

The data width for Local Master read/write transactions, must be the same as the PCI Data Bus Size.

### Local Target Data Bus Size (Master/Target cores only)

The data width for Local Target read/write transactions, must be the same as the PCI Data Bus Size.

**Local Data Bus Size (Target cores only)**
The data width for Local side Target read/write transactions, must be the same as the PCI Data Bus Size.

**Local Address Bus Width**
The address width for Local Master and Target read/write transactions, must be the same as the PCI Data Bus Size.

**Bus Speed**
PCI bus operation frequency. A clock frequency on the PCI side. A fixed value that depends on the PCI core being used.

## Backend Configuration

### Enable Backend Configuration

When this option is selected, the core works independently by configuring in the backend. The PCI core will provide a backend interface named self_cfg. The self_cfg interface can directly configure the PCI core after power on instead of another PCI master on PCI bus. The self_cfg interface can read/write the PCI core configuration space., The core takes the read/write command same as PCI config command(cben=h02/h03).

The self_cfg interface signals are listed in Table 3-2.

***Table 3-2. self_cfg Interface Signals***

| Port Name | Type | Corresponding PCI signals | Description |
|---|---|---|---|
| self_cfg_en | In | | Self configuration enable signals, when it's 1'b1, pci bus will be blocked and replaced by self_cfg interface. |
| self_cfg_addr | In | ad (address cycle) | Address of configuration space |
| self_cfg_data_in | In | ad (data cycle of config write command) | Data write to configuration space |
| self_cfg_data_out | Out | ad (data cycle of config read command) | Data read from configuration space |
| self_cfg_rd_wrn | In | cben(0) | Specify the Read/write command. '1' define a read command, '0' define a write command. |
| self_cfg_rdy | Out | !delseln & !trdyn | Only valid for read command, '1' indicate that self_cfg_data_out is valid. |

The backend asserst self_cfn_en to '1' and then starts configuring the PCI core. After configuration is finished, self_cfg_en is deasserted to '0'.

## Synthesis/Simulation Tools Selection

**Support Synplify**
If selected, IPexpress generates evaluation scripts and other associated files required to synthesize the top-level design using the Synplify synthesis tool.

**Support Precision**
If selected, IPexpress generates evaluation script and other associated files required to synthesize the top-level design using the Precision synthesis tool.

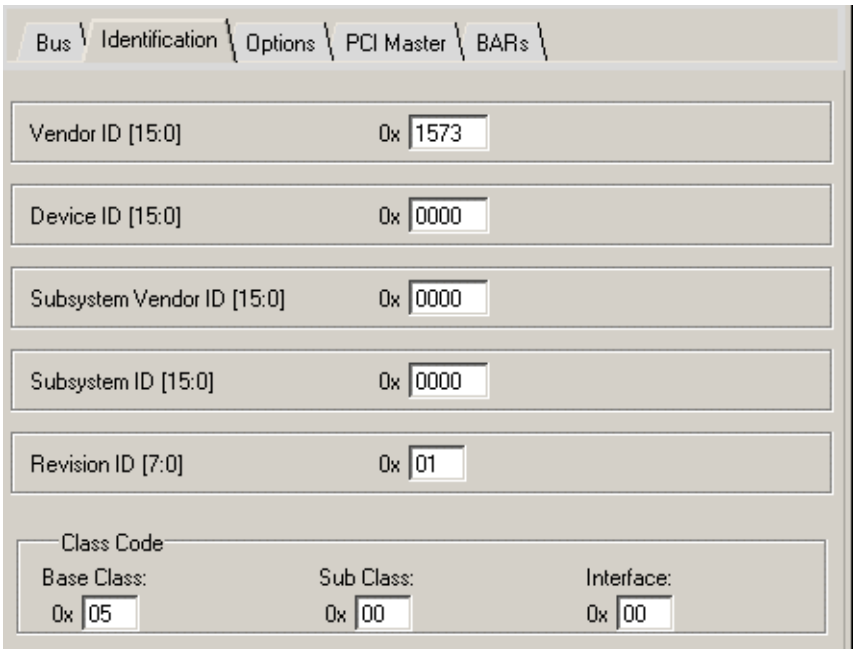
**Support ModelSim**
If selected, IPexpress generates evaluation script and other associated files required to synthesize the top-level design using the Modelsim simulator.

**Support ALDEC**
If selected, IPexpress generates evaluation script and other associated files required to synthesize the top-level design using the ALDEC simulator.

## Identification Tab

Figure 3-2 shows the contents of the Identification tab. This example shows the PCI Master/Target 33.

***Figure 3-2. Identification Tab***




### Vendor ID [15:0]

The Vendor ID is a 16-bit parameter used to identify the manufacturer of the product. The Vendor ID is assigned by the PCI SIG to ensure uniqueness.

### Device ID [15:0]

The Device ID is a 16-bit parameter defined by the manufacturer to uniquely identify a particular product or model.

### Subsystem Vendor ID [15:0]

The Subsystem Vendor ID is a 16-bit parameter used to further identify the manufacturer of the expansion board or subsystem.

### Subsystem ID [15:0]

The Subsystem ID is a 16-bit parameter used to further identify the particular device. This field is defined by the manufacturer to uniquely identify products or models.

### Revision ID [15:0]

The Revision ID is an 8-bit parameter used by the manufacturer and should be viewed as an extension of the Device ID to distinguish between different functional versions of a PCI product.

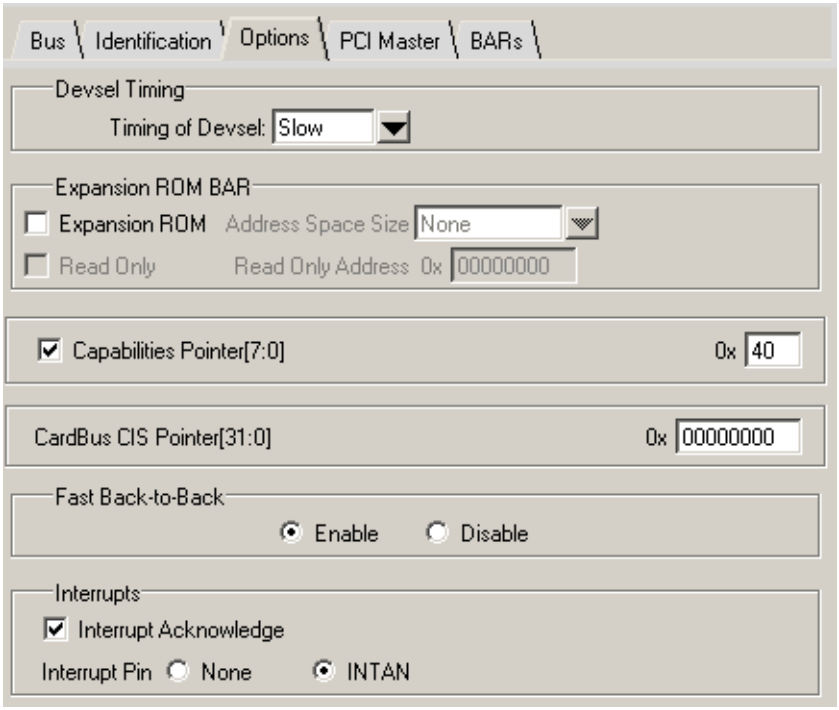
### Class Code (Base Class, Bus Class, Interface)

The Class Code is broken into three byte-size fields. The base class code broadly classifies the type of function the device performs. The sub-class code identifies more specifically the function of the device. The interface byte identifies a specific register-level programming interface.

# Options Tab

Figure 3-3 shows the contents of the Options tab. This example shows the PCI Master/Target 33.

***Figure 3-3. Options Tab***




## Devsel Timing

### Timing of Devsel

The slowest time for a device to assert the devseln signal for all accesses except the configuration accesses. The PCI Core supports only the slow decode setting.

## Expansion ROM BAR

### Expansion ROM

When selected, includes support for the Expansion ROM option.

### Address Space Size

Specifies the Expansion ROM address space size.

### Read Only and Read Only Address

When Read Only is selected, the Expansion ROM base address is specified by the Read Only Address parameter and can only be read by other PCI devices. When Read Only is not selected, the Expansion ROM base address can be specified by another PCI master device via the PCI bus.

### Capabilities Pointer

The Capabilities Pointer indicates the starting location of the Capabilities List.

### CardBus CIS Pointer

The CardBus CIS Pointer is 32-bit register at location 28h in the Configuration Space. For more information on setting this register, refer to the CardBus specification.

### Fast Back-to-Back

This option determines if the master Core supports two or more complete PCI transactions without an idle state between them.

### Interrupts

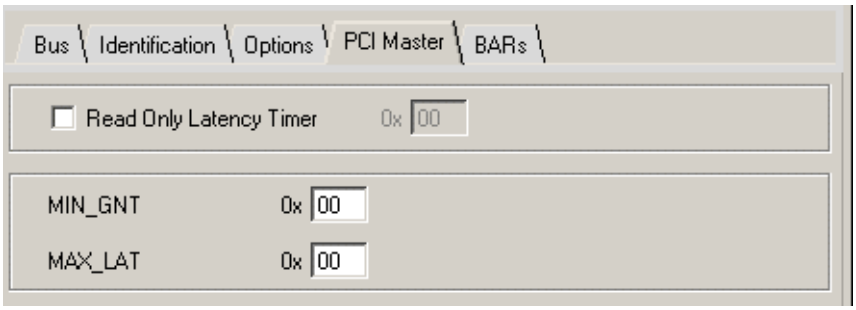
**Interrupt Acknowledge**
This option determines if the PCI core supports Interrupt Acknowledge.

**Interrupt Pin**
Indicates which Interrupt Pin will be used by the PCI Core.

## PCI Master Tab (PCI Master/Target Cores Only)

Figure 3-4 shows the contents of the PCI Master tab. This example shows the PCI Master/Target 33.

***Figure 3-4. PCI Master Tab***




### Read Only Latency Timer

A mechanism for ensuring that a bus master does not extend the access latency of other masters beyond a specified value.

### MIN_GNT

An 8-bit parameter used to specify the length of time in microseconds for the Master to control the PCI bus.

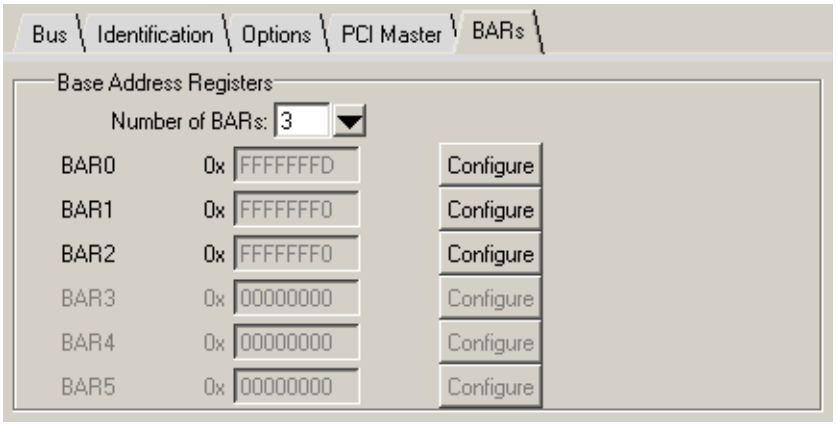
### MAX_LAT

An 8-bit parameter used to specify how often the PCI Core possess the bus.

## BARs Tab

Figure 3-5 shows the contents of the BARs tab. This example shows the PCI Master/Target 33.

***Figure 3-5. BARs Tab***



## Base Address Registers

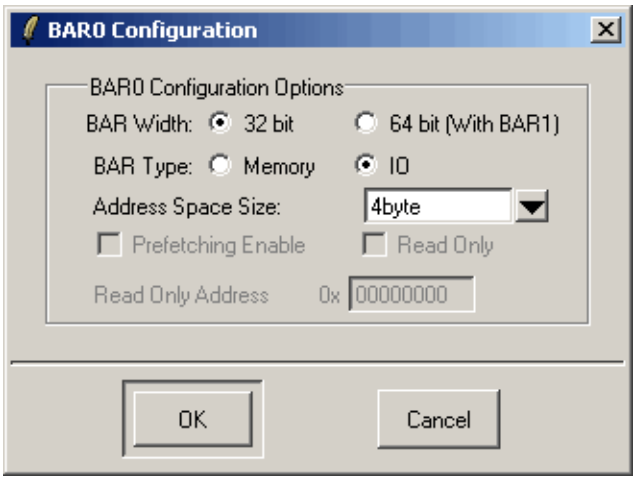
**Number of BARs**
The number of Base Address Registers configured by the user.

**BAR0 - BAR5**
The Base Address value used to map memory or I/O address space.

# BAR Configuration Options

Figure 3-6 shows the BAR Configuration dialog box, which is displayed when the Configure button is pressed in the BARs tab.

***Figure 3-6. BAR Configuration Options***




## BAR Width

The width of the Base Address Register. When using 64-bit width, the current BAR and next BAR will combine as the 64-bit Base Address.

## BAR Type

Used to map memory or I/O space.

## Address Space Size

The parameter is the size of the address range mapped to memory or I/O space.

## Prefetching Enable

This option determines if the memory mapped by this BAR support prefetching operation.

# IP Core Generation

This chapter provides information on how to generate the PCI IP core using the Diamond or ispLEVER software IPexpress tool, and how to include the core in a top-level design.

## Licensing the IP Core

An IP core- and device-specific license is required to enable full, unrestricted use of the PCI IP core in a complete, top-level design. Instructions on how to obtain licenses for Lattice IP cores are given at:

http://www.latticesemi.com/products/intellectualproperty/aboutip/isplevercoreonlinepurchas.cfm

Users may download and generate the PCI IP core and fully evaluate the core through functional simulation and implementation (synthesis, map, place and route) without an IP license. The PCI IP core also supports Lattice's IP hardware evaluation capability, which makes it possible to create versions of the IP core that operate in hardware for a limited time (approximately four hours) without requiring an IP license. See "Hardware Evaluation" on page 148 for further details. However, a license is required to enable timing simulation, to open the design in the Diamond or ispLEVER EPIC tool, and to generate bitstreams that do not include the hardware evaluation timeout limitation.

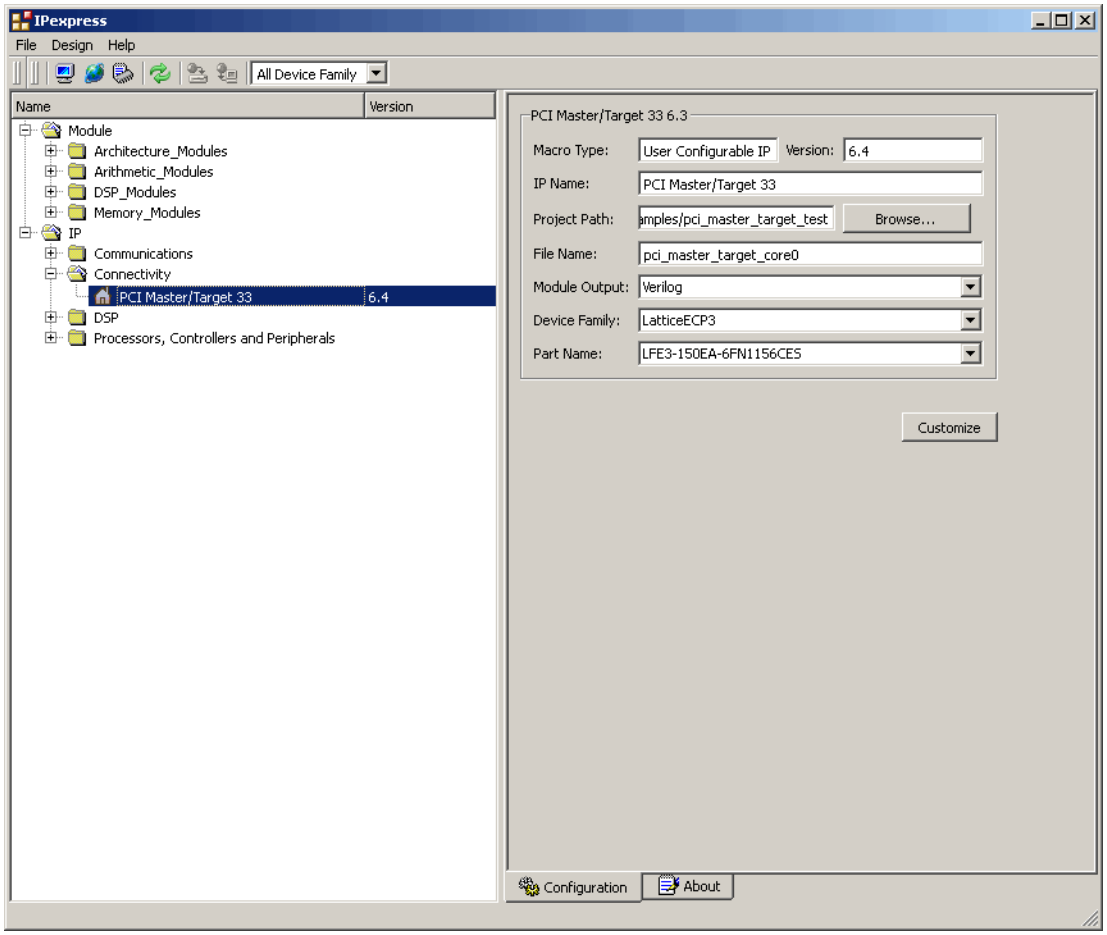
## Getting Started

The PCI IP core is available for download from the Lattice IP Server using the IPexpress tool. The IP files are automatically installed using ispUPDATE technology in any customer-specified directory. After the IP core has been installed, the IP core will be available in the IPexpress GUI dialog box shown in Figure 4-1.

The IPexpress tool GUI dialog box for the PCI IP core is shown in Figure 4-1. To generate a specific IP core configuration the user specifies:

- **Project Path** – Path to the directory where the generated IP files will be located.
- **File Name** – "username" designation given to the generated IP core and corresponding folders and files.
- **(Diamond) Module Output** – Verilog or VHDL.
- **(ispLEVER) Design Entry Type** – Verilog HDL or VHDL.
- **Device Family** – Device family to which IP is to be targeted (e.g. LatticeSCM, Lattice ECP2M, LatticeECP3, etc.). Only families that support the particular IP core are listed.
- **Part Name** – Specific targeted part within the selected device family.

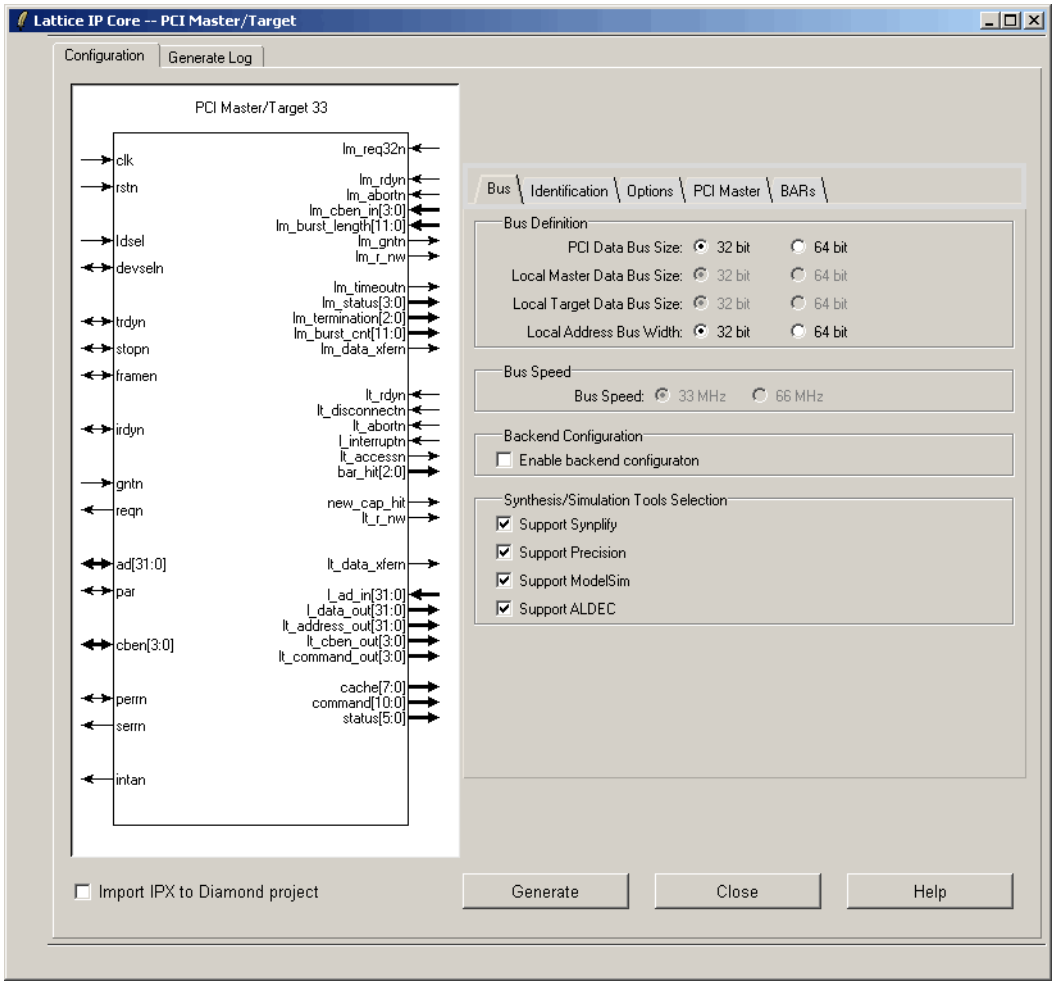
***Figure 4-1. IPexpress Dialog Box (Diamond Version)***




Note that if the IPexpress tool is called from within an existing project, Project Path, Module Output (Design Entry in ispLEVER), Device Family and Part Name default to the specified project parameters. Refer to the IPexpress tool online help for further information.

To create a custom configuration, the user clicks the **Customize** button in the IPexpress tool dialog box to display the PCI IP core Configuration GUI, as shown in Figure 4-2. From this dialog box, the user can select the IP parameter options specific to their application. Refer to "Parameter Settings" on page 136 for more information on the PCI IP core parameter settings.

***Figure 4-2. Configuration GUI (Diamond Version)***



## IPexpress-Created Files and Top Level Directory Structure

When the user clicks the **Generate** button in the IP Configuration dialog box, the IP core and supporting files are generated in the specified "Project Path" directory. The directory structure of the generated files is shown in Figure 4-3. This example shows the directory structure generated with the PCI Master/Target 33 for LatticeECP3 device.

***Figure 4-3. PCI IP Core Directory Structure***

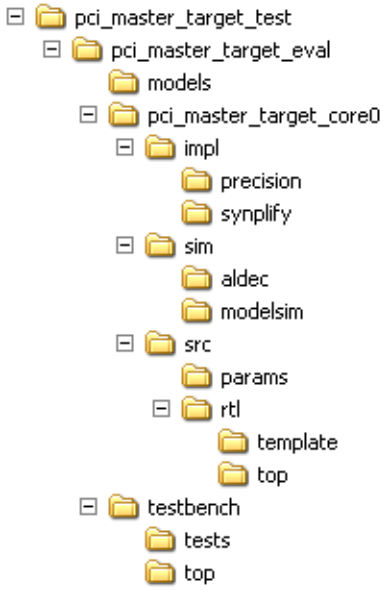



Table 4-1 provides a list of key files and directories created by the IPexpress tool and how they are used. The IPexpress tool creates several files that are used throughout the design cycle. The names of most of the created files are customized to the user's module name specified in the IPexpress tool.

***Table 4-1. File List***

| File | Description |
|---|---|
| <*username*>.lpc | This file contains the IPexpress tool options used to recreate or modify the core in the IPexpress tool. |
| <*username*>.ipx | The IPX file holds references to all of the elements of an IP or Module after it is generated from the IPexpress tool (Diamond version only). The file is used to bring in the appropriate files during the design implementation and analysis. It is also used to re-load parameter settings into the IP/Module generation GUI when an IP/Module is being re-generated. |
| <*username*>.ngo | This file provides the synthesized IP core. |
| <*username*>_bb.v/.vhd | This file provides the synthesis black box for the user's synthesis. |
| <*username*>_inst.v/.vhd | This file provides an instance template for the PCI IP core. |
| <*username*>_beh.v/.vhd | This file provides the front-end simulation library for the PCI IP core. |

Table 4-2 provides a list of key additional files providing IP core generation status information and command line generation capability are generated in the user's project directory.

***Table 4-2. Additional Files***

| File | Description |
|---|---|
| <*username*>_generate.tcl | This file is created when the GUI "Generate" button is pushed. This file may be run from command line. |
| <*username*>_generate.log | This is the synthesis and map log file. |

***Table 4-2. Additional Files (Continued)***

| | |
|---|---|
| <*username*>_gen.log | This is the IPexpress IP generation log file |

## Instantiating the Core

The generated PCI IP core package includes black-box (<*username*>_bb.v/vhd) and instance (<*userame*>_inst.v/vhd) templates (Verilog or VHDL) that can be used to instantiate the core in a top-level design. An example RTL top-level reference source file that can be used as an instantiation template for the IP core is provided in <*project_dir*>\pci_master_target_eval\<*username*>\src\rtl\top. Users may also use this top-level reference as the starting template for the top-level for their complete design.

## Running Functional Simulation

Simulation support for the PCI IP core is provided for Aldec Active-HDL (Verilog and VHDL) simulator and Mentor Graphics ModelSim (Verilog only) simulator.

The functional simulation includes a PCI bus stimulus module (pci_stim_tb) and a local module (lt_stim_tb), which is instantiated in a top level (pci_testbench_top). Module pci_stim_tb simulates a master PCI to configure PCI core and test the core's basic read/write command.

The generated IP core package includes behavior model (<*username*>_beh.v) for functional simulation in the “Project Path” root directory, which <*username*>_beh.v is instantiated in PCI top model (\<*project_dir*>\pci_master_target_eval\<*username*>\src\rtl\top).

The simulation script supporting ModelSim evaluation simulation is provided in \<*project_dir*>\pci_master_target_eval\<*username*>\sim\modelsim.

The simulation script supporting Aldec evaluation simulation is provided in \<*project_dir*>\pci_master_target_eval\<*username*>\sim\aldec.

The Test Application Design is instantiated in a test-bench provided in \<*project_dir*>\pci_master_target_eval\testbench.

Both ModelSim and Aldec simulation is supported via test bench files provided in \<*project_dir*>\pci_master_target_eval\testbench. Models required for simulation are provided in the corresponding \models folder.

Users may run the Aldec evaluation simulation by doing the following:

1. Open Active-HDL.
2. Under the Tools tab, select **Execute Macro**.
3. Browse to folder \<*project_dir*>\pci_master_target_eval\<*username*>\sim\aldec and execute one of the “do” scripts shown.

Users may run the ModelSim evaluation simulation by doing the following:

1. Open ModelSim.
2. Under the File tab, select **Change Directory** and choose the folder <*project_dir*>\pci_master_target_eval\<*username*>\sim\modelsim.
3. Under the Tools tab, select **Execute Macro** and execute the ModelSim “do” script shown.

***Note:*** *When the simulation completes, displayed on the console are:*

```
<< Simulation complete... >>
```

```
<< Number of errors: 0 >>
```

## Synthesizing and Implementing the Core in a Top-Level Design

Synthesis support for the PCI IP core is provided for Mentor Graphics Precision or Synopsys Synplify. The PCI IP core itself is synthesized and is provided in NGO format when the core is generated in IPexpress. Users may synthesize the core in their own top-level design by instantiating the core in their top-level as described previously and then synthesizing the entire design with either Synplify or Precision RTL synthesis.

The top-level file <*username*>_eval_top.v provided in
\<*project_dir*>\pci_master_target_eval\<*username*>\src\top
supports the ability to implement the PCI Express core in isolation. Push-button implementation of this top-level design with either Synplify or Precision RTL Synthesis is supported via the project files
<username>_eval.ldf (Diamond) or .syn (ispLEVER) located in the
\<*project_dir*>\pci_master_target_eval\<*username*>\impl\synplify
and the
\<*project_dir*>\pci_master_target_eval\<*username*>\impl\precision directories, respectively.

*To use this project file in Diamond:*

1. Choose **File > Open > Project**.
2. Browse to \<*project_dir*>\pci_master_target_eval\<*username*>\impl\(synplify or precision) in the Open Project dialog box.
3. Select and open <*username*>.ldf. At this point, all of the files needed to support top-level synthesis and implementation will be imported to the project.
4. Select the **Process** tab in the left-hand GUI window.
5. Implement the complete design via the standard Diamond GUI flow.

*To use this project file in ispLEVER:*

1. Choose **File > Open Project.**
2. Browse to \<*project_dir*>\pci_master_target_eval\<*username*>\impl\(synplify or precision) in the Open Project dialog box.
3. Select and open <*username*>.syn. At this point, all of the files needed to support top-level synthesis and implementation will be imported to the project.
4. Select the device top-level entry in the left-hand GUI window.
5. Implement the complete design via the standard ispLEVER GUI flow.

## Hardware Evaluation

The PCI IP core supports Lattice's IP hardware evaluation capability, which makes it possible to create versions of the IP core that operate in hardware for a limited period of time (approximately four hours) without requiring the purchase of an IP license. It may also be used to evaluate the core in hardware in user-defined designs.

### Enabling Hardware Evaluation in Diamond

Choose **Project > Active Strategy > Translate Design Settings**. The hardware evaluation capability may be enabled/disabled in the Strategy dialog box. It is enabled by default.

### Enabling Hardware Evaluation in ispLEVER

In the Processes for Current Source pane, right-click the **Build Database** process and choose **Properties** from the dropdown menu. The hardware evaluation capability may be enabled/disabled in the Properties dialog box. It is enabled by default.

## Updating/Regenerating the IP Core

By regenerating an IP core with the IPexpress tool, you can modify any of its settings including device type, design entry method, and any of the options specific to the IP core. Regenerating can be done to modify an existing IP core or to create a new but similar one.

### Regenerating an IP Core in Diamond

*To regenerate an IP core in Diamond:*

1. In IPexpress, click the **Regenerate** button.
2. In the Regenerate view of IPexpress, choose the IPX source file of the module or IP you wish to regenerate.
3. IPexpress shows the current settings for the module or IP in the Source box. Make your new settings in the T**arget** box.
4. If you want to generate a new set of files in a new location, set the new location in the **IPX Target File** box. The base of the file name will be the base of all the new file names. The IPX Target File must end with an .ipx extension.
5. Click **Regenerate.** The module's dialog box opens showing the current option settings.
6. In the dialog box, choose the desired options. To get information about the options, click **Help**. Also, check the About tab in IPexpress for links to technical notes and user guides. IP may come with additional information. As the options change, the schematic diagram of the module changes to show the I/O and the device resources the module will need.
7. To import the module into your project, if it's not already there, select **Import IPX to Diamond Project** (not available in stand-alone mode).
8. Click **Generate**.
9. Check the Generate Log tab to check for warnings and error messages.
10. Click **Close**.

The IPexpress package file (.ipx) supported by Diamond holds references to all of the elements of the generated IP core required to support simulation, synthesis and implementation. The IP core may be included in a user's design by importing the .ipx file to the associated Diamond project. To change the option settings of a module or IP that is already in a design project, double-click the module's .ipx file in the File List view. This opens IPexpress and the module's dialog box showing the current option settings. Then go to step 6 above.

### Regenerating an IP Core in ispLEVER

*To regenerate an IP core in ispLEVER:*

1. In the IPexpress tool, choose **Tools > Regenerate IP/Module**.
2. In the Select a Parameter File dialog box, choose the Lattice Parameter Configuration (.lpc) file of the IP core you wish to regenerate, and click **Open**.
3. The Select Target Core Version, Design Entry, and Device dialog box shows the current settings for the IP core in the Source Value box. Make your new settings in the Target Value box.

4. If you want to generate a new set of files in a new location, set the location in the LPC Target File box. The base of the .lpc file name will be the base of all the new file names. The LPC Target File must end with an .lpc extension.
5. Click **Next**. The IP core's dialog box opens showing the current option settings.
6. In the dialog box, choose desired options. To get information about the options, click **Help**. Also, check the About tab in the IPexpress tool for links to technical notes and user guides. The IP core might come with additional information. As the options change, the schematic diagram of the IP core changes to show the I/O and the device resources the IP core will need.
7. Click **Generate**.
8. Click the **Generate Log** tab to check for warnings and error messages.

# Support Resources

This chapter contains information about Lattice Technical Support, additional references, and document revision history.

## Lattice Technical Support

There are a number of ways to receive technical support.

### Online Forums

The first place to look is Lattice Forums (http://www.latticesemi.com/support/forums.cfm). Lattice Forums contain a wealth of knowledge and are actively monitored by Lattice Applications Engineers.

### Telephone Support Hotline

Receive direct technical support for all Lattice products by calling Lattice Applications from 5:30 a.m. to 6 p.m. Pacific Time.

- For USA & Canada: 1-800-LATTICE (528-8423)
- For other locations: +1 503 268 8001

In Asia, call Lattice Applications from 8:30 a.m. to 5:30 p.m. Beijing Time (CST), +0800 UTC. Chinese and English language only.

- For Asia: +86 21 52989090

### E-mail Support

- techsupport@latticesemi.com
- techsupport-asia@latticesemi.com

### Local Support

Contact your nearest Lattice Sales Office.

### Internet

www.latticesemi.com

### PCI-SIG Website

The Peripheral Component Interconnect Special Interest Group (PCI-SIG) website contains specifications and documents referred to in this user's guide. The PCi-SIG URL is:

http://www.pcisig.com.

## References

### LatticeEC/ECP

- HB1000, *LatticeEC/ECP Family Handbook*

### LatticeECP2M

- HB1003, *LatticeECP2M Family Handbook*

### LatticeECP3

- HB1009, *LatticeECP3 Family Handbook*

### LatticeSC/M

- DS1004, *LatticeSC/M Family Data Sheet*

### LatticeXP

- HB1001, *LatticeXP Family Handbook*

### LatticeXP2

- DS1009, *Lattice XP2 Datasheet*

### MachXO

- DS1002, *MachXO Family Datasheet*

### MachXO2

- DS1035*, MachXO2 Family Datasheet*

## Revision History

| Date | Document Version | IP Version | Change Summary |
|---|---|---|---|
| — | — | 5.2 | Previous Lattice releases. |
| August 2006 | 08.3 | 5.2 | Core version 5.2 - IPexpress User-Configurable flow supported for LatticeECP/EC, LatticeECP2, LatticeSC, LatticeXP, and MachXO only. |
| September 2006 | 08.4 | 5.2 | Added Parameter Descriptions section. |
| December 2006 | 08.5 | 5.2 | Updated appendices. Added LatticeECP2M appendix. |
| April 2007 | 08.6 | 5.2 | Updated references to PCI Local Bus Specification from revision 2.2 to revision 3.0. |
| | | | Updated Command Register figure and table. Replaced “stepping control” with “reserved bit”. |
| May 2007 | 08.7 | 5.2 | Updated BAR Mapped to Memory Space section. |
| | | | Updated command9:0 signal description in the Local Interface Signals table. |
| | | | Added support for LatticeXP2 FPGA family. |
| November 2007 | 08.8 | 5.2 | Updated appendices. |
| July 2008 | 08.9 | 6.1 | Updated appendices. |
| July 2009 | 09.0 | 6.1 | Added support for LatticeECP3 FPGA family. |
| July 2010 | 09.1 | 6.2 | Divided document into chapters. Added table of contents. |
| | | | Added Quick Facts table in Chapter 1, “Introduction.” |
| | | | Added new content in Chapter 4, “IP Core Generation.” |
| November 2010 | 09.2 | 6.4 | Added support for Diamond software throughout. |
| | | | Added support for MachXO2 device family throughout. |

*Appendix A:*

# Resource Utilization

This appendix gives resource utilization information for Lattice FPGAs using the PCI IP core.

IPexpress is the Lattice IP configuration utility, and is included as a standard feature of the Diamond and ispLEVER design tools. Details regarding the usage of IPexpress can be found in the IPexpress and Diamond or ispLEVER help system. For more information on the Diamond or ispLEVER design tools, visit the Lattice web site at: www.latticesemi.com/software.

## LatticeECP and LatticeEC FPGAs

***Table A-1. Performance and Resource Utilization[1]***

| IPexpress User-Configurable Mode | SLICEs | LUTs | Registers | sysMEM™ EBRs | External Pins (PCI Interface) | $f_{MAX}$ |
|---|---|---|---|---|---|---|
| Target 33 MHz, 32-bit PCI/Local/Address bus width | 586 | 703 | 472 | 0 | 48 | 33 |
| Target 33 MHz, 64-bit PCI/Local/Address bus width | 715 | 913 | 594 | 0 | 87 | 33 |
| Target 66 MHz, 32-bit PCI/Local/Address bus width | 606 | 966 | 493 | 0 | 48 | 66 |
| Target 66 MHz, 64-bit PCI/Local/Address bus width | 832 | 1344 | 614 | 0 | 87 | 66 |
| Master/Target 33 MHz, 32-bit PCI/Local/Address bus width | 846 | 1060 | 642 | 0 | 50 | 33 |
| Master/Target 33 MHz, 64-bit PCI/Local/Address bus width | 1153 | 1549 | 849 | 0 | 89 | 33 |
| Master/Target 66 MHz, 32-bit PCI/Local/Address bus width | 1083 | 1690 | 663 | 0 | 50 | 66 |
| Master/Target 66 MHz, 64-bit PCI/Local/Address bus width | 1599 | 2569 | 869 | 0 | 89 | 66 |

1. Performance and utilization data are generated using an LFEC33E-5F672C device with Lattice Diamond 1.0 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeECP/EC family.

### Ordering Part Number

Table A-2 lists the Ordering Part Number (OPNs) for each mode of operation supported by the PCI IP core for LatticeECP/EC.

***Table A-2. OPN for LatticeECP/EC PCI IP Core***

| Speed | PCI Bus | Type | OPN |
|---|---|---|---|
| 33 MHz | 32-bit | Target | PCI-T32-E2-U6 |
| 33 MHz | 64-bit | Target | PCI-T64-E2-U6 |
| 66 MHz | 32-bit | Target | PCI-T32-E2-U6 |
| 66 MHz | 64-bit | Target | PCI-T64-E2-U6 |
| 33 MHz | 32-bit | Master/Target | PCI-MT32-E2-U6 |
| 33 MHz | 64-bit | Master/Target | PCI-MT64-E2-U6 |
| 66 MHz | 32-bit | Master/Target | PCI-MT32-E2-U6 |
| 66 MHz | 64-bit | Master/Target | PCI-MT64-E2-U6 |

## LatticeECP2 FPGAs

***Table A-3. Performance and Resource Utilization[1]***

| IPexpress User-Configurable Mode | SLICEs | LUTs | Registers | sysMEM EBRs | External Pins (PCI Interface) | $f_{MAX}$ |
|---|---|---|---|---|---|---|
| Target 33 MHz, 32-bit PCI/ Local/Address bus width | 593 | 717 | 472 | 0 | 48 | 33 |
| Target 33 MHz, 64-bit PCI/ Local/Address bus width | 722 | 927 | 594 | 0 | 87 | 33 |
| Target 66 MHz, 32-bit PCI/ Local/Address bus width | 606 | 972 | 493 | 0 | 48 | 66 |
| Target 66 MHz, 64-bit PCI/ Local/Address bus width | 832 | 1350 | 614 | 0 | 87 | 66 |
| Master/Target 33 MHz, 32-bit PCI/ Local/Address bus width | 856 | 1068 | 642 | 0 | 50 | 33 |
| Master/Target 33 MHz, 64-bit PCI/ Local/Address bus width | 1168 | 1561 | 849 | 0 | 89 | 33 |
| Master/Target 66 MHz, 32-bit PCI/ Local/Address bus width | 1086 | 1700 | 663 | 0 | 50 | 66 |
| Master/Target 66 MHz, 64-bit PCI/ Local/Address bus width | 1598 | 2580 | 869 | 0 | 89 | 66 |

1. Performance and utilization data are generated using an LFE2-20E-6F672C device with Lattice Diamond 1.0 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeECP2 family.

### Ordering Part Number

Table A-4 lists the Ordering Part Number (OPNs) for each mode of operation supported by the PCI IP core for LatticeECP2.

***Table A-4. OPN for LatticeECP2 PCI IP Core***

| Speed | PCI Bus | Type | OPN |
|---|---|---|---|
| 33 MHz | 32-bit | Target | PCI-T32-P2-U6 |
| 33 MHz | 64-bit | Target | PCI-T64-P2-U6 |
| 66 MHz | 32-bit | Target | PCI-T32-P2-U6 |
| 66 MHz | 64-bit | Target | PCI-T64-P2-U6 |
| 33 MHz | 32-bit | Master/Target | PCI-MT32-P2-U6 |
| 33 MHz | 64-bit | Master/Target | PCI-MT64-P2-U6 |
| 66 MHz | 32-bit | Master/Target | PCI-MT32-P2-U6 |
| 66 MHz | 64-bit | Master/Target | PCI-MT64-P2-U6 |

## LatticeECP2M FPGAs

***Table A-5. Performance and Resource Utilization[1]***

| IPexpress User-Configurable Mode | SLICEs | LUTs | Registers | sysMEM EBRs | External Pins (PCI Interface) | $f_{MAX}$ |
|---|---|---|---|---|---|---|
| Target 33 MHz, 32-bit PCI/ Local/Address bus width | 593 | 717 | 472 | 0 | 48 | 33 |
| Target 33 MHz, 64-bit PCI/ Local/Address bus width | 722 | 927 | 594 | 0 | 87 | 33 |
| Target 66 MHz, 32-bit PCI/ Local/Address bus width | 606 | 972 | 493 | 0 | 48 | 66 |
| Target 66 MHz, 64-bit PCI/ Local/Address bus width | 832 | 1350 | 614 | 0 | 87 | 66 |
| Master/Target 33 MHz, 32-bit PCI/ Local/Address bus width | 856 | 1068 | 642 | 0 | 50 | 33 |
| Master/Target 33 MHz, 64-bit PCI/ Local/Address bus width | 1168 | 1561 | 849 | 0 | 89 | 33 |
| Master/Target 66 MHz, 32-bit PCI/ Local/Address bus width | 1086 | 1700 | 663 | 0 | 50 | 66 |
| Master/Target 66 MHz, 64-bit PCI/ Local/Address bus width | 1598 | 2580 | 869 | 0 | 89 | 66 |

1. Performance and utilization data are generated using an LFE2M-35E-6F672C device with Lattice Diamond 1.0 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeECP2M family.

### Ordering Part Number

Table A-6 lists the Ordering Part Number (OPNs) for each mode of operation supported by the PCI IP core for LatticeECP2M.

***Table A-6. OPN for LatticeECP2M PCI IP Core***

| Speed | PCI Bus | Type | OPN |
|---|---|---|---|
| 33 MHz | 32-bit | Target | PCI-T32-PM-U6 |
| 33 MHz | 64-bit | Target | PCI-T64-PM-U6 |
| 66 MHz | 32-bit | Target | PCI-T32-PM-U6 |
| 66 MHz | 64-bit | Target | PCI-T64-PM-U6 |
| 33 MHz | 32-bit | Master/Target | PCI-MT32-PM-U6 |
| 33 MHz | 64-bit | Master/Target | PCI-MT64-PM-U6 |
| 66 MHz | 32-bit | Master/Target | PCI-MT32-PM-U6 |
| 66 MHz | 64-bit | Master/Target | PCI-MT64-PM-U6 |

## LatticeECP3 FPGAs

***Table A-7. Performance and Resource Utilization[1]***

| IPexpress User-Configurable Mode | SLICEs | LUTs | Registers | sysMEM EBRs | External Pins (PCI Interface) | $f_{MAX}$ |
|---|---|---|---|---|---|---|
| Target 33 MHz, 32-bit PCI/Local/Address bus width | 483 | 706 | 470 | 0 | 48 | 33 |
| Target 33 MHz, 64-bit PCI/Local/Address bus width | 612 | 918 | 592 | 0 | 87 | 33 |
| Target 66 MHz, 32-bit PCI/Local/Address bus width | 589 | 963 | 491 | 0 | 48 | 66 |
| Target 66 MHz, 64-bit PCI/Local/Address bus width | 809 | 1341 | 612 | 0 | 87 | 66 |
| Master/Target 33 MHz, 32-bit PCI/Local/Address bus width | 683 | 1059 | 640 | 0 | 50 | 33 |
| Master/Target 33 MHz, 64-bit PCI/Local/Address bus width | 1005 | 1552 | 847 | 0 | 89 | 33 |
| Master/Target 66 MHz, 32-bit PCI/Local/Address bus width | 1076 | 1691 | 661 | 0 | 50 | 66 |
| Master/Target 66 MHz, 64-bit PCI/Local/Address bus width | 1550 | 2570 | 867 | 0 | 89 | 66 |

1. Performance and utilization data are generated using an LFE3-95EA-7FN1156CES device with Lattice Diamond 1.0 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeECP3 family.

### Ordering Part Number

Table A-8 lists the Ordering Part Number (OPNs) for each mode of operation supported by the PCI IP core for LatticeECP3.

***Table A-8. OPN for LatticeECP3 PCI IP Core***

| Speed | PCI Bus | Type | OPN |
|---|---|---|---|
| 33 MHz | 32-bit | Target | PCI-T32-E3-U6 |
| 33 MHz | 64-bit | Target | PCI-T64-E3-U6 |
| 66 MHz | 32-bit | Target | PCI-T32-E3-U6 |
| 66 MHz | 64-bit | Target | PCI-T64-E3-U6 |
| 33 MHz | 32-bit | Master/Target | PCI-MT32-E3-U6 |
| 33 MHz | 64-bit | Master/Target | PCI-MT64-E3-U6 |
| 66 MHz | 32-bit | Master/Target | PCI-MT32-E3-U6 |
| 66 MHz | 64-bit | Master/Target | PCI-MT64-E3-U6 |

# LatticeXP FPGAs

***Table A-9. Performance and Resource Utilization[1]***

| IPexpress User-Configurable Mode | SLICEs | LUTs | Registers | sysMEM EBRs | External Pins (PCI Interface) | $f_{MAX}$ |
|---|---|---|---|---|---|---|
| Target 33 MHz, 32-bit PCI/ Local/Address bus width | 586 | 703 | 472 | 0 | 48 | 33 |
| Target 33 MHz, 64-bit PCI/ Local/Address bus width | 715 | 913 | 594 | 0 | 87 | 33 |
| Target 66 MHz, 32-bit PCI/ Local/Address bus width | 606 | 966 | 493 | 0 | 48 | 66 |
| Target 66 MHz, 64-bit PCI/ Local/Address bus width | 832 | 1344 | 614 | 0 | 87 | 66 |
| Master/Target 33 MHz, 32-bit PCI/ Local/Address bus width | 846 | 1060 | 642 | 0 | 50 | 33 |
| Master/Target 33 MHz, 64-bit PCI/ Local/Address bus width | 1090 | 1549 | 849 | 0 | 89 | 33 |
| Master/Target 66 MHz, 32-bit PCI/ Local/Address bus width | 1083 | 1690 | 663 | 0 | 50 | 66 |

1. Performance and utilization data are generated using an LFXP20C-5F484C device with Lattice Diamond 1.0 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeXP family.

## Ordering Part Number

Table A-10 lists the Ordering Part Number (OPNs) for each mode of operation supported by the PCI IP core for LatticeXP.

***Table A-10. OPN for LatticeXP PCI IP Core***

| Speed | PCI Bus | Type | OPN |
|---|---|---|---|
| 33 MHz | 32-bit | Target | PCI-T32-XM-U6 |
| 33 MHz | 64-bit | Target | PCI-T64-XM-U6 |
| 66 MHz | 32-bit | Target | PCI-T32-XM-U6 |
| 66 MHz | 64-bit | Target | PCI-T64-XM-U6 |
| 33 MHz | 32-bit | Master/Target | PCI-MT32-XM-U6 |
| 33 MHz | 64-bit | Master/Target | PCI-MT64-XM-U6 |
| 66 MHz | 32-bit | Master/Target | PCI-MT32-XM-U6 |
| 66 MHz | 64-bit | Master/Target | PCI-MT64-XM-U6 |

## LatticeXP2 FPGAs

***Table A-11. Performance and Resource Utilization[1]***

| IPexpress User-Configurable Mode | SLICEs | LUTs | Registers | sysMEM EBRs | External Pins (PCI Interface) | $f_{MAX}$ |
|---|---|---|---|---|---|---|
| Target 33 MHz, 32-bit PCI/ Local/Address bus width | 588 | 709 | 470 | 0 | 48 | 33 |
| Target 33 MHz, 64-bit PCI/ Local/Address bus width | 707 | 919 | 592 | 0 | 87 | 33 |
| Target 66 MHz, 32-bit PCI/ Local/Address bus width | 601 | 964 | 491 | 0 | 48 | 66 |
| Target 66 MHz, 64-bit PCI/ Local/Address bus width | 827 | 1342 | 612 | 0 | 87 | 66 |
| Master/Target 33 MHz, 32-bit PCI/ Local/Address bus width | 851 | 1060 | 640 | 0 | 50 | 33 |
| Master/Target 33 MHz, 64-bit PCI/ Local/Address bus width | 1100 | 1553 | 847 | 0 | 89 | 33 |
| Master/Target 66 MHz, 32-bit PCI/ Local/Address bus width | 1081 | 1692 | 661 | 0 | 50 | 66 |
| Master/Target 66 MHz, 64-bit PCI/ Local/Address bus width | 1530 | 2572 | 867 | 0 | 89 | 66 |

1. Performance and utilization data are generated using an LFXP2-17E-6F484C device with Lattice Diamond 1.0 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeXP2 family.

### Ordering Part Number

Table A-12 lists the Ordering Part Number (OPNs) for each mode of operation supported by the PCI IP core for LatticeXP2.

***Table A-12. OPN for LatticeXP2 PCI IP Core***

| Speed | PCI Bus | Type | OPN |
|---|---|---|---|
| 33 MHz | 32-bit | Target | PCI-T32-X2-U6 |
| 33 MHz | 64-bit | Target | PCI-T64-X2-U6 |
| 66 MHz | 32-bit | Target | PCI-T32-X2-U6 |
| 66 MHz | 64-bit | Target | PCI-T64-X2-U6 |
| 33 MHz | 32-bit | Master/Target | PCI-MT32-X2-U6 |
| 33 MHz | 64-bit | Master/Target | PCI-MT64-X2-U6 |
| 66 MHz | 32-bit | Master/Target | PCI-MT32-X2-U6 |
| 66 MHz | 64-bit | Master/Target | PCI-MT64-X2-U6 |

# MachXO FPGAs

***Table A-13. Performance and Resource Utilization[1]***

| IPexpress User-Configurable Mode | SLICEs | LUTs | Registers | sysMEM EBRs | External Pins (PCI Interface) | $f_{MAX}$ |
|---|---|---|---|---|---|---|
| Target 33 MHz, 32-bit PCI/ Local/Address bus width | 359 | 703 | 472 | 0 | 48 | 33 |
| Target 66 MHz, 32-bit PCI/ Local/Address bus width | 517 | 966 | 493 | 0 | 48 | 66 |
| Master/Target 33 MHz, 32-bit PCI/ Local/Address bus width | 542 | 1060 | 642 | 0 | 50 | 33 |

1. Performance and utilization data are generated using an LCMXO2280C-5FT324C device with Lattice Diamond 1.0 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the MachXO family.

## Ordering Part Number

Table A-14 lists the Ordering Part Number (OPNs) for each mode of operation supported by the PCI IP core for MachXO.

***Table A-14. MachXO OPN for PCI IP Core***

| Speed | PCI Bus | Type | OPN |
|---|---|---|---|
| 33 MHz | 32-bit | Target | PCI-T32-XO-U6 |
| 66 MHz | 32-bit | Target | PCI-T32-XO-U6 |
| 33 MHz | 32-bit | Master/Target | PCI-MT32-XO-U6 |

# MachXO2 FPGAs

***Table A-15. Performance and Resource Utilization[1]***

| IPexpress User-Configurable Mode | SLICEs | LUTs | Registers | sysMEM EBRs | External Pins (PCI Interface) | $f_{MAX}$ |
|---|---|---|---|---|---|---|
| Target 33 MHz, 32-bit PCI/ Local/Address bus width | 304 | 601 | 422 | 0 | 48 | 33 |
| Master/Target 33 MHz, 32-bit PCI/ Local/Address bus width | 406 | 803 | 582 | 0 | 50 | 33 |

1. Performance and utilization data are generated using an LCMXO2-1200HC-6TG144CES device with Lattice Diamond 1.0 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the MachXO2 family.

## Ordering Part Number

Table A-16 lists the Ordering Part Number (OPNs) for each mode of operation supported by the PCI IP core for MachXO2.

***Table A-16. OPN for MachXO2 PCI IP Core***

| Speed | PCI Bus | Type | OPN |
|---|---|---|---|
| 33 MHz | 32-bit | Target | PCI-T32-M2-U1 |
| 33 MHz | 32-bit | Master/Target | PCI-MT32-M2-U1 |

## LatticeSC FPGAs

***Table A-17. Performance and Resource Utilization[1]***

| IPexpress User-Configurable Mode | SLICEs | LUTs | Registers | sysMEM EBRs | External Pins (PCI Interface) | $f_{MAX}$ |
|---|---|---|---|---|---|---|
| Target 33 MHz, 32-bit PCI/ Local/Address bus width | 488 | 679 | 470 | 0 | 48 | 33 |
| Target 33 MHz, 64-bit PCI/ Local/Address bus width | 621 | 893 | 594 | 0 | 87 | 33 |
| Target 66 MHz, 32-bit PCI/ Local/Address bus width | 618 | 990 | 493 | 0 | 48 | 66 |
| Target 66 MHz, 64-bit PCI/ Local/Address bus width | 845 | 1391 | 622 | 0 | 87 | 66 |
| Master/Target 33 MHz, 32-bit PCI/ Local/Address bus width | 724 | 1050 | 640 | 0 | 50 | 33 |
| Master/Target 33 MHz, 64-bit PCI/ Local/Address bus width | 986 | 1529 | 850 | 0 | 89 | 33 |
| Master/Target 66 MHz, 32-bit PCI/ Local/Address bus width | 1085 | 1722 | 663 | 0 | 50 | 66 |
| Master/Target 66 MHz, 64-bit PCI/ Local/Address bus width | 1513 | 2631 | 871 | 0 | 89 | 66 |

1. Performance and utilization data are generated using an LFSC3GA25E-6F900C device with Lattice Diamond 1.0 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeSC family.

### Ordering Part Number

Table A-18 lists the Ordering Part Number (OPNs) for each mode of operation supported by the PCI IP core for LatticeSC.

***Table A-18. OPN for LatticeSC PCI IP Core***

| Speed | PCI Bus | Type | OPN |
|---|---|---|---|
| 33 MHz | 32-bit | Target | PCI-T32-SC-U6 |
| 33 MHz | 64-bit | Target | PCI-T64-SC-U6 |
| 66 MHz | 32-bit | Target | PCI-T32-SC-U6 |
| 66 MHz | 64-bit | Target | PCI-T64-SC-U6 |
| 33 MHz | 32-bit | Master/Target | PCI-MT32-SC-U6 |
| 33 MHz | 64-bit | Master/Target | PCI-MT64-SC-U6 |
| 66 MHz | 32-bit | Master/Target | PCI-MT32-SC-U6 |
| 66 MHz | 64-bit | Master/Target | PCI-MT64-SC-U6 |

# Pin Assignment Considerations for LatticeECP and LatticeEC Devices

## PCI Pin Assignments for Master/Target 33MHz 64-Bit Bus

The PCI Master/Target 33MHz 64-bit core is optimized for LFEC33E-5F672C. An example assignment, optimized for best performance, is given in Table B-1. In the IPexpress user-configurable design flow, actual pin assignment is contained with the .lpf preference file. Refer to the readme file included with the core package for further information.

***Table B-1. PCI Pins Assignments***

| Signal Name | Pin/Bank | Buffer Type |
|---|---|---|
| **PCI System Pins** | | |
| clk | W1/6 | LVCMOS33_IN |
| rstn | Y8/5 | PCI33_IN |
| **PCI Address and Data** | | |
| ad[0] | AB13/5 | PCI33_BIDI |
| ad[1] | AC10/5 | PCI33_BIDI |
| ad[2] | AD10/5 | PCI33_BIDI |
| ad[3] | AA9/5 | PCI33_BIDI |
| ad[4] | AB9/5 | PCI33_BIDI |
| ad[5] | AC9/5 | PCI33_BIDI |
| ad[6] | AD9/5 | PCI33_BIDI |
| ad[7] | AA8/5 | PCI33_BIDI |
| ad[8] | AB8/5 | PCI33_BIDI |
| ad[9] | AC8/5 | PCI33_BIDI |
| ad[10] | AD8/5 | PCI33_BIDI |
| ad[11] | AE8/5 | PCI33_BIDI |
| ad[12] | AF8/5 | PCI33_BIDI |
| ad[13] | AA7/5 | PCI33_BIDI |
| ad[14] | AB7/5 | PCI33_BIDI |
| ad[15] | AC7/5 | PCI33_BIDI |
| ad[16] | AD7/5 | PCI33_BIDI |
| ad[17] | AE7/5 | PCI33_BIDI |
| ad[18] | AF7/5 | PCI33_BIDI |
| ad[19] | AC6/5 | PCI33_BIDI |
| ad[20] | AD6/5 | PCI33_BIDI |
| ad[21] | AE6/5 | PCI33_BIDI |
| ad[22] | AF6/5 | PCI33_BIDI |
| ad[23] | AC5/5 | PCI33_BIDI |
| ad[24] | AD5/5 | PCI33_BIDI |
| ad[25] | AE5/5 | PCI33_BIDI |
| ad[26] | AF5/5 | PCI33_BIDI |
| ad[27] | AE4/5 | PCI33_BIDI |
| ad[28] | AF4/5 | PCI33_BIDI |
| ad[29] | AE3/5 | PCI33_BIDI |

*Table B-1. PCI Pins Assignments (Continued)*

| Signal Name | Pin/Bank | Buffer Type |
|---|---|---|
| ad[30] | AF3/5 | PCI33_BIDI |
| ad[31] | AE2/5 | PCI33_BIDI |
| ad[32] | AF24/4 | PCI33_BIDI |
| ad[33] | AF23/4 | PCI33_BIDI |
| ad[34] | AE23/4 | PCI33_BIDI |
| ad[35] | AF22/4 | PCI33_BIDI |
| ad[36] | AE22/4 | PCI33_BIDI |
| ad[37] | AF21/4 | PCI33_BIDI |
| ad[38] | AE21/4 | PCI33_BIDI |
| ad[39] | AF20/4 | PCI33_BIDI |
| ad[40] | AE20/4 | PCI33_BIDI |
| ad[41] | AF19/4 | PCI33_BIDI |
| ad[42] | AE19/4 | PCI33_BIDI |
| ad[43] | AF18/4 | PCI33_BIDI |
| ad[44] | AE18/4 | PCI33_BIDI |
| ad[45] | AD18/4 | PCI33_BIDI |
| ad[46] | AC18/4 | PCI33_BIDI |
| ad[47] | AB18/4 | PCI33_BIDI |
| ad[48] | AA18/4 | PCI33_BIDI |
| ad[49] | AF17/4 | PCI33_BIDI |
| ad[50] | AD17/4 | PCI33_BIDI |
| ad[51] | AC17/4 | PCI33_BIDI |
| ad[52] | AB17/4 | PCI33_BIDI |
| ad[53] | AA17/4 | PCI33_BIDI |
| ad[54] | AF16/4 | PCI33_BIDI |
| ad[55] | AE16/4 | PCI33_BIDI |
| ad[56] | AD16/4 | PCI33_BIDI |
| ad[57] | AC16/4 | PCI33_BIDI |
| ad[58] | AB16/4 | PCI33_BIDI |
| ad[59] | AA16/4 | PCI33_BIDI |
| ad[60] | Y16/4 | PCI33_BIDI |
| ad[61] | AD15/4 | PCI33_BIDI |
| ad[62] | AA15/4 | PCI33_BIDI |
| ad[63] | Y15/4 | PCI33_BIDI |
| cben[0] | AA10/5 | PCI33_BIDI |
| cben[1] | AC11/5 | PCI33_BIDI |
| cben[2] | AD11/5 | PCI33_BIDI |
| cben[3] | AF9/5 | PCI33_BIDI |
| cben[4] | AF12/5 | PCI33_BIDI |
| cben[5] | AF11/5 | PCI33_BIDI |
| cben[6] | AE11/5 | PCI33_BIDI |
| cben[7] | AA11/5 | PCI33_BIDI |
| par | AF10/5 | PCI33_BIDI |
| par64 | AB11/5 | PCI33_BIDI |

*Table B-1. PCI Pins Assignments (Continued)*

| Signal Name | Pin/Bank | Buffer Type |
|---|---|---|
| **PCI Interface Controls** | | |
| Framen | AA12/5 | PCI33_BIDI |
| irdyn | AB12/5 | PCI33_BIDI |
| trdyn | AC12/5 | PCI33_BIDI |
| stopn | AE12/5 | PCI33_BIDI |
| Idsel | AB6/5 | PCI33_IN |
| devseln | AD12/5 | PCI33_BIDI |
| perrn | AE9/5 | PCI33_BIDI |
| serrn | AE10/5 | PCI33_BIDI |
| ack64n | AE13/5 | PCI33_BIDI |
| req64n | AF13/5 | PCI33_BIDI |
| **PCI Interrupts** | | |
| intan | AA6/5 | PCI33_OUT |
| **PCI Bus Arbitration** | | |
| gntn | AF2/5 | PCI33_IN |
| reqn | AD4/5 | PCI33_OUT |

## PCI Pin Assignments for Target 66MHz 64-Bit Bus

The PCI Target 33MHz 64-bit core is optimized for LFEC33E-5F672C. An example pin assignment, optimized for best performance, is given in Table B-2. Refer to the readme file included with the core package for further information.

*Table B-2. PCI Pins Assignments*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| **PCI System Pins** | | |
| clk | W1/6 | LVCMOS33_IN |
| rstn | Y8/5 | PCI33_IN |
| **PCI Address and Data** | | |
| ad[0] | AB10/5 | PCI33_BIDI |
| ad[1] | AC10/5 | PCI33_BIDI |
| ad[2] | AD10/5 | PCI33_BIDI |
| ad[3] | AA9/5 | PCI33_BIDI |
| ad[4] | AB9/5 | PCI33_BIDI |
| ad[5] | AC9/5 | PCI33_BIDI |
| ad[6] | AD9/5 | PCI33_BIDI |
| ad[7] | AA8/5 | PCI33_BIDI |
| ad[8] | AB8/5 | PCI33_BIDI |
| ad[9] | AC8/5 | PCI33_BIDI |
| ad[10] | AD8/5 | PCI33_BIDI |
| ad[11] | AE8/5 | PCI33_BIDI |
| ad[12] | AF8/5 | PCI33_BIDI |
| ad[13] | AA7/5 | PCI33_BIDI |
| ad[14] | AB7/5 | PCI33_BIDI |
| ad[15] | AC7/5 | PCI33_BIDI |
| ad[16] | AD7/5 | PCI33_BIDI |

*Table B-2. PCI Pins Assignments (Continued)*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| ad[17] | AE7/5 | PCI33_BIDI |
| ad[18] | AF7/5 | PCI33_BIDI |
| ad[19] | AC6/5 | PCI33_BIDI |
| ad[20] | AD6/5 | PCI33_BIDI |
| ad[21] | AE6/5 | PCI33_BIDI |
| ad[22] | AF6/5 | PCI33_BIDI |
| ad[23] | AC5/5 | PCI33_BIDI |
| ad[24] | AD5/5 | PCI33_BIDI |
| ad[25] | AE5/5 | PCI33_BIDI |
| ad[26] | AF5/5 | PCI33_BIDI |
| ad[27] | AE4/5 | PCI33_BIDI |
| ad[28] | AF4/5 | PCI33_BIDI |
| ad[29] | AE3/5 | PCI33_BIDI |
| ad[30] | AF3/5 | PCI33_BIDI |
| ad[31] | AE2/5 | PCI33_BIDI |
| ad[32] | AF24/4 | PCI33_BIDI |
| ad[33] | AF23/4 | PCI33_BIDI |
| ad[34] | AE23/4 | PCI33_BIDI |
| ad[35] | AF22/4 | PCI33_BIDI |
| ad[36] | AE22/4 | PCI33_BIDI |
| ad[37] | AF21/4 | PCI33_BIDI |
| ad[38] | AE21/4 | PCI33_BIDI |
| ad[39] | AF20/4 | PCI33_BIDI |
| ad[40] | AE20/4 | PCI33_BIDI |
| ad[41] | AF19/4 | PCI33_BIDI |
| ad[42] | AE19/4 | PCI33_BIDI |
| ad[43] | AF18/4 | PCI33_BIDI |
| ad[44] | AE18/4 | PCI33_BIDI |
| ad[45] | AD18/4 | PCI33_BIDI |
| ad[46] | AC18/4 | PCI33_BIDI |
| ad[47] | AB18/4 | PCI33_BIDI |
| ad[48] | AA18/4 | PCI33_BIDI |
| ad[49] | AF17/4 | PCI33_BIDI |
| ad[50] | AD17/4 | PCI33_BIDI |
| ad[51] | AC17/4 | PCI33_BIDI |
| ad[52] | AB17/4 | PCI33_BIDI |
| ad[53] | AA17/4 | PCI33_BIDI |
| ad[54] | AF16/4 | PCI33_BIDI |
| ad[55] | AE16/4 | PCI33_BIDI |
| ad[56] | AD16/4 | PCI33_BIDI |
| ad[57] | AC16/4 | PCI33_BIDI |
| ad[58] | AB16/4 | PCI33_BIDI |
| ad[59] | AA16/4 | PCI33_BIDI |
| ad[60] | Y16/4 | PCI33_BIDI |

*Table B-2. PCI Pins Assignments (Continued)*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| ad[61] | AD15/4 | PCI33_BIDI |
| ad[62] | AA15/4 | PCI33_BIDI |
| ad[63] | Y15/4 | PCI33_BIDI |
| cben[0] | AA10/5 | PCI33_IN |
| cben[1] | AC11/5 | PCI33_IN |
| cben[2] | AD11/5 | PCI33_IN |
| cben[3] | AF9/5 | PCI33_IN |
| cben[4] | AF12/5 | PCI33_IN |
| cben[5] | AF11/5 | PCI33_IN |
| cben[6] | AE11/5 | PCI33_IN |
| cben[7] | AA11/5 | PCI33_IN |
| par64 | AB11/5 | PCI33_BIDI |
| par | AF10/5 | PCI33_BIDI |
| **PCI Interface Controls** | | |
| Framen | AA12/5 | PCI33_IN |
| irdyn | AB12/5 | PCI33_IN |
| trdyn | AC12/5 | PCI33_OUT |
| Stopn | AE12/5 | PCI33_OUT |
| Idsel | AB6/5 | PCI33_IN |
| devseln | AD12/5 | PCI33_OUT |
| perrn | AE9/5 | PCI33_OUT |
| serrn | AE10/5 | PCI33_OUT |
| ack64n | AE13/5 | PCI33_OUT |
| req64n | AF13/5 | PCI33_IN |
| **PCI Interrupts** | | |
| intan | AA6/5 | PCI33_OUT |

## PCI Pin Assignments for Master/Target 33MHz 32-Bit Bus

The PCI Master/Target 33MHz 32-bit core is optimized for LFEC33E-5F672C. An example pin assignment, optimized for best performance, is given in Table B-3. Refer to the readme file included with the core package for further information.

*Table B-3. PCI Pins Assignments*

| Signal Name | Pin/Bank | Buffer Type |
|---|---|---|
| **PCI System Pins** | | |
| clk | W1/6 | LVCMOS33_IN |
| rstn | Y8/5 | PCI33_IN |
| **PCI Address and Data** | | |
| ad[0] | AB10/5 | PCI33_BIDI |
| ad[1] | AC10/5 | PCI33_BIDI |
| ad[2] | AD10/5 | PCI33_BIDI |
| ad[3] | AA9/5 | PCI33_BIDI |
| ad[4] | AB9/5 | PCI33_BIDI |
| ad[5] | AC9/5 | PCI33_BIDI |
| ad[6] | AD9/5 | PCI33_BIDI |

*Table B-3. PCI Pins Assignments (Continued)*

| Signal Name | Pin/Bank | Buffer Type |
|---|---|---|
| ad[7] | AA8/5 | PCI33_BIDI |
| ad[8] | AB8/5 | PCI33_BIDI |
| ad[9] | AC8/5 | PCI33_BIDI |
| ad[10] | AD8/5 | PCI33_BIDI |
| ad[11] | AE8/5 | PCI33_BIDI |
| ad[12] | AF8/5 | PCI33_BIDI |
| ad[13] | AA7/5 | PCI33_BIDI |
| ad[14] | AB7/5 | PCI33_BIDI |
| ad[15] | AC7/5 | PCI33_BIDI |
| ad[16] | AD7/5 | PCI33_BIDI |
| ad[17] | AE7/5 | PCI33_BIDI |
| ad[18] | AF7/5 | PCI33_BIDI |
| ad[19] | AC6/5 | PCI33_BIDI |
| ad[20] | AD6/5 | PCI33_BIDI |
| ad[21] | AE6/5 | PCI33_BIDI |
| ad[22] | AF6/5 | PCI33_BIDI |
| ad[23] | AC5/5 | PCI33_BIDI |
| ad[24] | AD5/5 | PCI33_BIDI |
| ad[25] | AE5/5 | PCI33_BIDI |
| ad[26] | AF5/5 | PCI33_BIDI |
| ad[27] | AE4/5 | PCI33_BIDI |
| ad[28] | AF4/5 | PCI33_BIDI |
| ad[29] | AE3/5 | PCI33_BIDI |
| ad[30] | AF3/5 | PCI33_BIDI |
| ad[31] | AE2/5 | PCI33_BIDI |
| cben[0] | AA10/5 | PCI33_BIDI |
| cben[1] | AC11/5 | PCI33_BIDI |
| cben[2] | AD11/5 | PCI33_BIDI |
| cben[3] | AF9/5 | PCI33_BIDI |
| Par | AF10/5 | PCI33_BIDI |
| **PCI Interface Controls** | | |
| Framen | AA12/5 | PCI33_BIDI |
| irdyn | AB12/5 | PCI33_BIDI |
| trdyn | AC12/5 | PCI33_BIDI |
| stopn | AE12/5 | PCI33_BIDI |
| Idsel | AB6/5 | PCI33_IN |
| devseln | AD12/5 | PCI33_BIDI |
| perrn | AE9/5 | PCI33_BIDI |
| serrn | AE10/5 | PCI33_BIDI |
| **PCI Interrupts** | | |
| intan | AA6/5 | PCI33_OUT |
| **PCI Bus Arbitration** | | |
| gntn | AF2/5 | PCI33_IN |
| reqn | AD4/5 | PCI33_OUT |

## PCI Pin Assignments for Target 33MHz 32-Bit Bus

The PCI Target 33MHz 32-bit core is optimized for LFEC33E-5F672C. An example pin assignment, optimized for best performance, is given in Table B-4. Refer to the readme file included with the core package for further information.

***Table B-4. PCI Pins Assignments***

| Signal Name | Pin/ Bank | Buffer Type |
|---|---|---|
| **PCI System Pins** | | |
| clk | W1/6 | LVCMOS33_IN |
| rstn | Y8/5 | PCI33_IN |
| **PCI Address and Data** | | |
| ad[0] | AB10/5 | PCI33_BIDI |
| ad[1] | AC10/5 | PCI33_BIDI |
| ad[2] | AD10/5 | PCI33_BIDI |
| ad[3] | AA9/5 | PCI33_BIDI |
| ad[4] | AB9/5 | PCI33_BIDI |
| ad[5] | AC9/5 | PCI33_BIDI |
| ad[6] | AD9/5 | PCI33_BIDI |
| ad[7] | AA8/5 | PCI33_BIDI |
| ad[8] | AB8/5 | PCI33_BIDI |
| ad[9] | AC8/5 | PCI33_BIDI |
| ad[10] | AD8/5 | PCI33_BIDI |
| ad[11] | AE8/5 | PCI33_BIDI |
| ad[12] | AF8/5 | PCI33_BIDI |
| ad[13] | AA7/5 | PCI33_BIDI |
| ad[14] | AB7/5 | PCI33_BIDI |
| ad[15] | AC7/5 | PCI33_BIDI |
| ad[16] | AD7/5 | PCI33_BIDI |
| ad[17] | AE7/5 | PCI33_BIDI |
| ad[18] | AF7/5 | PCI33_BIDI |
| ad[19] | AC6/5 | PCI33_BIDI |
| ad[20] | AD6/5 | PCI33_BIDI |
| ad[21] | AE6/5 | PCI33_BIDI |
| ad[22] | AF6/5 | PCI33_BIDI |
| ad[23] | AC5/5 | PCI33_BIDI |
| ad[24] | AD5/5 | PCI33_BIDI |
| ad[25] | AE5/5 | PCI33_BIDI |
| ad[26] | AF5/5 | PCI33_BIDI |
| ad[27] | AE4/5 | PCI33_BIDI |
| ad[28] | AF4/5 | PCI33_BIDI |
| ad[29] | AE3/5 | PCI33_BIDI |
| ad[30] | AF3/5 | PCI33_BIDI |
| ad[31] | AE2/5 | PCI33_BIDI |
| cben[0] | AA10/5 | PCI33_BIDI |
| cben[1] | AC11/5 | PCI33_BIDI |
| cben[2] | AD11/5 | PCI33_BIDI |

*Table B-4. PCI Pins Assignments (Continued)*

| Signal Name | Pin/ Bank | Buffer Type |
|---|---|---|
| cben[3] | AF9/5 | PCI33_BIDI |
| Par | AF10/5 | PCI33_BIDI |
| **PCI Interface Controls** | | |
| Framen | AA12/5 | PCI33_BIDI |
| irdyn | AB12/5 | PCI33_BIDI |
| trdyn | AC12/5 | PCI33_BIDI |
| stopn | AE12/5 | PCI33_BIDI |
| Idsel | AB6/5 | PCI33_IN |
| devseln | AD12/5 | PCI33_BIDI |
| perrn | AE9/5 | PCI33_BIDI |
| serrn | AE10/5 | PCI33_BIDI |
| **PCI Interrupts** | | |
| intan | AA6/5 | PCI33_OUT |

# Pin Assignment Considerations for LatticeXP Devices

## PCI Pin Assignments for Master/Target 33MHz 32-Bit Bus

The PCI Master/Target 33MHz 32-bit core is optimized for LFXP10-4F388C. An example pin assignment, optimized for best performance, is given in Table B-5. Refer to the readme file included with the core package for further information.

*Table B-5. PCI Pins Assignments*

| Signal Name | Pin/Bank | BufferType |
|---|---|---|
| **PCI System Pins** | | |
| clk | U1/6 | LVCMOS33_IN |
| rstn | AA4/5 | PCI33_IN |
| **PCI Address and Data** | | |
| ad[0] | AB18/4 | PCI33_BIDI |
| ad[1] | AA18/4 | PCI33_BIDI |
| ad[2] | Y18/4 | PCI33_BIDI |
| ad[3] | AB17/4 | PCI33_BIDI |
| ad[4] | Y14/4 | PCI33_BIDI |
| ad[5] | Y13/4 | PCI33_BIDI |
| ad[6] | AA17/4 | PCI33_BIDI |
| ad[7] | Y17/4 | PCI33_BIDI |
| ad[8] | AB16/4 | PCI33_BIDI |
| ad[9] | AA16/4 | PCI33_BIDI |
| ad[10] | AB15/4 | PCI33_BIDI |
| ad[11] | AA15/4 | PCI33_BIDI |
| ad[12] | W13/4 | PCI33_BIDI |
| ad[13] | W12/4 | PCI33_BIDI |
| ad[14] | AB14/4 | PCI33_BIDI |
| ad[15] | AA14/4 | PCI33_BIDI |
| ad[16] | AA13/4 | PCI33_BIDI |
| ad[17] | AA10/5 | PCI33_BIDI |

*Table B-5. PCI Pins Assignments (Continued)*

| Signal Name | Pin/Bank | BufferType |
|---|---|---|
| ad[18] | Y8/5 | PCI33_BIDI |
| ad[19] | AB8/5 | PCI33_BIDI |
| ad[20] | AA8/5 | PCI33_BIDI |
| ad[21] | Y7/5 | PCI33_BIDI |
| ad[22] | AB7/5 | PCI33_BIDI |
| ad[23] | AA7/5 | PCI33_BIDI |
| ad[24] | Y10/5 | PCI33_BIDI |
| ad[25] | Y9/5 | PCI33_BIDI |
| ad[26] | AB6/5 | PCI33_BIDI |
| ad[27] | AA6/5 | PCI33_BIDI |
| ad[28] | AB5/5 | PCI33_BIDI |
| ad[29] | AA5/5 | PCI33_BIDI |
| ad[30] | AB4/5 | PCI33_BIDI |
| ad[31] | W9/5 | PCI33_BIDI |
| cben[0] | AA19/4 | PCI33_BIDI |
| cben[1] | Y20/4 | PCI33_BIDI |
| cben[2] | W14/4 | PCI33_BIDI |
| cben[3] | W15/4 | PCI33_BIDI |
| **PCI Interface Controls** | | |
| par | W11/5 | PCI33_BIDI |
| serrn | AB9/5 | PCI33_BIDI |
| perrn | W10/5 | PCI33_BIDI |
| devseln | AB10/5 | PCI33_BIDI |
| framen | AB11/5 | PCI33_BIDI |
| irdyn | Y11/5 | PCI33_BIDI |
| stopn | AA11/5 | PCI33_BIDI |
| trdyn | Y12/5 | PCI33_BIDI |
| gntn | AA12/4 | PCI33_BIDI |
| reqn | AB12/4 | PCI33_BIDI |
| **PCI Interrupts** | | |
| idsel | AB19/4 | PCI33_IN |
| intan | W6/5 | PCI33_OUT |

## PCI Pin Assignments for Target 33MHz 32-Bit Bus

The PCI Target 33MHz 32-bit core is optimized for LFXP10-4F388C. An example pin assignment, optimized for best performance, is given in Table 63. Refer to the readme file included with the core package for further information.

*Table B-6. PCI Pins Assignments*

| Signal Name | Pin/Bank | Buffer Type |
|---|---|---|
| **PCI System Pins** | | |
| clk | U1/6 | LVCMOS33_IN |
| rstn | AA4/5 | PCI33_IN |
| **PCI Address and Data** | | |
| ad[0] | AB18/4 | PCI33_BIDI |

*Table B-6. PCI Pins Assignments (Continued)*

| Signal Name | Pin/Bank | Buffer Type |
|---|---|---|
| ad[1] | AA18/4 | PCI33_BIDI |
| ad[2] | Y18/4 | PCI33_BIDI |
| ad[3] | AB17/4 | PCI33_BIDI |
| ad[4] | Y14/4 | PCI33_BIDI |
| ad[5] | Y13/4 | PCI33_BIDI |
| ad[6] | AA17/4 | PCI33_BIDI |
| ad[7] | Y17/4 | PCI33_BIDI |
| ad[8] | AB16/4 | PCI33_BIDI |
| ad[9] | AA16/4 | PCI33_BIDI |
| ad[10] | AB15/4 | PCI33_BIDI |
| ad[11] | AA15/4 | PCI33_BIDI |
| ad[12] | W13/4 | PCI33_BIDI |
| ad[13] | W12/4 | PCI33_BIDI |
| ad[14] | AB14/4 | PCI33_BIDI |
| ad[15] | AA14/4 | PCI33_BIDI |
| ad[16] | AA13/4 | PCI33_BIDI |
| ad[17] | AA10/5 | PCI33_BIDI |
| ad[18] | Y8/5 | PCI33_BIDI |
| ad[19] | AB8/5 | PCI33_BIDI |
| ad[20] | AA8/5 | PCI33_BIDI |
| ad[21] | Y7/5 | PCI33_BIDI |
| ad[22] | AB7/5 | PCI33_BIDI |
| ad[23] | AA7/5 | PCI33_BIDI |
| ad[24] | Y10/5 | PCI33_BIDI |
| ad[25] | Y9/5 | PCI33_BIDI |
| ad[26] | AB6/5 | PCI33_BIDI |
| ad[27] | AA6/5 | PCI33_BIDI |
| ad[28] | AB5/5 | PCI33_BIDI |
| ad[29] | AA5/5 | PCI33_BIDI |
| ad[30] | AB4/5 | PCI33_BIDI |
| ad[31] | W9/5 | PCI33_BIDI |
| cben[0] | AA19/4 | PCI33_BIDI |
| cben[1] | Y20/4 | PCI33_BIDI |
| cben[2] | W14/4 | PCI33_BIDI |
| cben[3] | W15/4 | PCI33_BIDI |
| par | W11/5 | PCI33_BIDI |
| **PCI Interface Controls** | | |
| serrn | AB9/5 | PCI33_BIDI |
| perrn | W10/5 | PCI33_BIDI |
| devseln | AB10/5 | PCI33_BIDI |
| framen | AB11/5 | PCI33_BIDI |
| irdyn | Y11/5 | PCI33_BIDI |
| stopn | AA11/5 | PCI33_BIDI |
| trdyn | Y12/5 | PCI33_BIDI |

*Table B-6. PCI Pins Assignments (Continued)*

| Signal Name | Pin/Bank | Buffer Type |
|---|---|---|
| idsel | AB19/4 | PCI33_IN |
| **PCI Interrupts** | | |
| intan | W6/5 | PCI33_OUT |

## PCI Pin Assignments for Master/Target 33MHz 64-Bit Bus

The PCI Master/Target 33MHz 64-bit core is optimized for LFXP20C-4F484C. An example pin assignment, optimized for best performance, is given in Table B-7. Refer to the readme file included with the core package for further information.

*Table B-7. PCI Pin Assignments*

| Signal Name | Pin/Bank | Buffer Type |
|---|---|---|
| **PCI System Pins** | | |
| clk | U2 / 6 | LVCMOS33_IN |
| rstn | Y5 / 5 | PCI33_IN |
| **PCI Address and Data** | | |
| ad(0) | W13 / 4 | PCI33_BIDI |
| ad(1) | Y13 / 4 | PCI33_BIDI |
| ad(2) | AA13 / 4 | PCI33_BIDI |
| ad(3) | AB13 / 4 | PCI33_BIDI |
| ad(4) | V12 / 4 | PCI33_BIDI |
| ad(5) | W12 / 4 | PCI33_BIDI |
| ad(6) | Y12 / 4 | PCI33_BIDI |
| ad(7) | AA12 / 4 | PCI33_BIDI |
| ad(8) | V11 / 5 | PCI33_BIDI |
| ad(9) | W11 / 5 | PCI33_BIDI |
| ad(10) | Y11 / 5 | PCI33_BIDI |
| ad(11) | AA11 / 5 | PCI33_BIDI |
| ad(12) | AB11 / 5 | PCI33_BIDI |
| ad(13) | V10 / 5 | PCI33_BIDI |
| ad(14) | W10 / 5 | PCI33_BIDI |
| ad(15) | Y10 / 5 | PCI33_BIDI |
| ad(16) | W8 / 5 | PCI33_BIDI |
| ad(17) | Y8 / 5 | PCI33_BIDI |
| ad(18) | AA8 / 5 | PCI33_BIDI |
| ad(19) | AB8 / 5 | PCI33_BIDI |
| ad(20) | U7 / 5 | PCI33_BIDI |
| ad(21) | V7 / 5 | PCI33_BIDI |
| ad(22) | W7 / 5 | PCI33_BIDI |
| ad(23) | Y7 / 5 | PCI33_BIDI |
| ad(24) | Y6 / 5 | PCI33_BIDI |
| ad(25) | AA6 / 5 | PCI33_BIDI |
| ad(26) | AB6 / 5 | PCI33_BIDI |
| ad(27) | AA5 / 5 | PCI33_BIDI |

*Table B-7. PCI Pin Assignments (Continued)*

| Signal Name | Pin/Bank | Buffer Type |
|---|---|---|
| ad(28) | AB5 / 5 | PCI33_BIDI |
| ad(29) | AA4 / 5 | PCI33_BIDI |
| ad(30) | AB4 / 5 | PCI33_BIDI |
| ad(31) | AA3 / 5 | PCI33_BIDI |
| ad(32) | AA21 / 4 | PCI33_BIDI |
| ad(33) | AB20 / 4 | PCI33_BIDI |
| ad(34) | AA20 / 4 | PCI33_BIDI |
| ad(35) | Y20 / 4 | PCI33_BIDI |
| ad(36) | AB19 / 4 | PCI33_BIDI |
| ad(37) | AA19 / 4 | PCI33_BIDI |
| ad(38) | Y19 / 4 | PCI33_BIDI |
| ad(39) | W19 / 4 | PCI33_BIDI |
| ad(40) | AB18 / 4 | PCI33_BIDI |
| ad(41) | AA18 / 4 | PCI33_BIDI |
| ad(42) | Y18 / 4 | PCI33_BIDI |
| ad(43) | W18 / 4 | PCI33_BIDI |
| ad(44) | V18 / 4 | PCI33_BIDI |
| ad(45) | U18 / 4 | PCI33_BIDI |
| ad(46) | AB17 / 4 | PCI33_BIDI |
| ad(47) | AA17 / 4 | PCI33_BIDI |
| ad(48) | Y17 / 4 | PCI33_BIDI |
| ad(49) | W17 / 4 | PCI33_BIDI |
| ad(50) | V17 / 4 | PCI33_BIDI |
| ad(51) | U17 / 4 | PCI33_BIDI |
| ad(52) | AB16 / 4 | PCI33_BIDI |
| ad(53) | AA16 / 4 | PCI33_BIDI |
| ad(54) | Y16 / 4 | PCI33_BIDI |
| ad(55) | W16 / 4 | PCI33_BIDI |
| ad(56) | V16 / 4 | PCI33_BIDI |
| ad(57) | U16 / 4 | PCI33_BIDI |
| ad(58) | AB15 / 4 | PCI33_BIDI |
| ad(59) | AA15 / 4 | PCI33_BIDI |
| ad(60) | Y15 / 4 | PCI33_BIDI |
| ad(61) | W15 / 4 | PCI33_BIDI |
| ad(62) | V15 / 4 | PCI33_BIDI |
| ad(63) | U15 / 4 | PCI33_BIDI |
| cben(0) | AB12 / 5 | PCI33_BIDI |
| cben(1) | AA10 / 5 | PCI33_BIDI |
| cben(2) | V8 / 5 | PCI33_BIDI |
| cben(3) | AB7 / 5 | PCI33_BIDI |
| cben(4) | AA14 / 4 | PCI33_BIDI |

*Table B-7. PCI Pin Assignments (Continued)*

| Signal Name | Pin/Bank | Buffer Type |
|---|---|---|
| cben(5) | Y14 / 4 | PCI33_BIDI |
| cben(6) | W14 / 4 | PCI33_BIDI |
| cben(7) | V14 / 4 | PCI33_BIDI |
| par | AB10 / 5 | PCI33_BIDI |
| par64 | AB14 / 4 | PCI33_BIDI |
| **PCI Interface Controls** | | |
| Framen | U8 / 5 | PCI33_BIDI |
| irdyn | AB9 / 5 | PCI33_BIDI |
| trdyn | AA9 / 5 | PCI33_BIDI |
| stopn | W9 / 5 | PCI33_BIDI |
| idsel | AA7 / 5 | PCI33_IN |
| devseln | Y9 / 5 | PCI33_BIDI |
| perrn | V9 / 5 | PCI33_BIDI |
| serrn | U9 / 5 | PCI33_BIDI |
| ack64n | V13 / 4 | PCI33_BIDI |
| req64n | U14 / 4 | PCI33_BIDI |
| **PCI Interrupts** | | |
| intan | W6 / 5 | PCI33_OUT |

# Pin Assignment Considerations for MachXO Devices

## PCI Pin Assignments for Target 33MHz 32-Bit Bus

The PCI Target 33MHz 32-bit core is optimized for LCMXO1200C-4FT256C. An example pin assignment, optimized for best performance, is given in Table B-8. Refer to the readme file included with the core package for further information.

*Table B-8. PCI Pin Assignments*

| Signal Name | Pin/Bank | Buffer Type |
|---|---|---|
| **PCI System Pins** | | |
| clk | M5 / 6 | LVCMOS33_IN |
| rstn | G2 / 7 | LVCMOS33_IN |
| **PCI Address and Data** | | |
| ad(0) | B9 / 1 | PCI33_BIDI |
| ad(1) | D10 / 1 | PCI33_BIDI |
| ad(2) | D9 / 1 | PCI33_BIDI |
| ad(3) | C10 / 1 | PCI33_BIDI |
| ad(4) | C9 / 1 | PCI33_BIDI |
| ad(5) | A9 / 1 | PCI33_BIDI |
| ad(6) | A10 / 1 | PCI33_BIDI |
| ad(7) | E9 / 1 | PCI33_BIDI |
| ad(8) | D7 / 0 | PCI33_BIDI |
| ad(9) | D8 / 0 | PCI33_BIDI |
| ad(10) | C8 / 0 | PCI33_BIDI |
| ad(11) | B8 / 0 | PCI33_BIDI |

*Table B-8. PCI Pin Assignments (Continued)*

| Signal Name | Pin/Bank | Buffer Type |
|---|---|---|
| ad(12) | A7 / 0 | PCI33_BIDI |
| ad(13) | A6 / 0 | PCI33_BIDI |
| ad(14) | B7 / 0 | PCI33_BIDI |
| ad(15) | B6 / 0 | PCI33_BIDI |
| ad(16) | C6 / 0 | PCI33_BIDI |
| ad(17) | A4 / 0 | PCI33_BIDI |
| ad(18) | A5 / 0 | PCI33_BIDI |
| ad(19) | E6 / 0 | PCI33_BIDI |
| ad(20) | E7 / 0 | PCI33_BIDI |
| ad(21) | C5 / 0 | PCI33_BIDI |
| ad(22) | C4 / 0 | PCI33_BIDI |
| ad(23) | B5 / 0 | PCI33_BIDI |
| ad(24) | D5 / 0 | PCI33_BIDI |
| ad(25) | D6 / 0 | PCI33_BIDI |
| ad(26) | A3 / 0 | PCI33_BIDI |
| ad(27) | A2 / 0 | PCI33_BIDI |
| ad(28) | D4 / 0 | PCI33_BIDI |
| ad(29) | D3 / 0 | PCI33_BIDI |
| ad(30) | B3 / 0 | PCI33_BIDI |
| ad(31) | B2 / 0 | PCI33_BIDI |
| cben(0) | A11 / 1 | PCI33_BIDI |
| cben(1) | E8 / 1 | PCI33_BIDI |
| cben(2) | C7 / 0 | PCI33_BIDI |
| cben(3) | B4 / 0 | PCI33_BIDI |
| par | D11 / 1 | PCI33_BIDI |
| **PCI Interface Controls** | | |
| Framen | B11 / 1 | PCI33_BIDI |
| irdyn | B12 / 1 | PCI33_BIDI |
| trdyn | C12 / 1 | PCI33_BIDI |
| stopn | C11 / 1 | PCI33_BIDI |
| idsel | A13 / 1 | PCI33_IN |
| devseln | A12 / 1 | PCI33_BIDI |
| perrn | E10 / 1 | PCI33_BIDI |
| serrn | D12 / 1 | PCI33_BIDI |
| **PCI Interrupts** | | |
| intan | A14 / 1 | PCI33_OUT |

## PCI Pin Assignments for Target 66MHz 32-Bit Bus

The PCI Target 66MHz 32-bit core is optimized for LCMXO1200C-4FT256C. An example pin assignment, optimized for best performance, is given in Table B-9. Refer to the readme file included with the core package for further information.

***Table B-9. PCI Pin Assignments***

| Signal Name | Pin/Bank | Buffer Type |
|---|---|---|
| **PCI System Pins** | | |
| clk | M5 / 6 | LVCMOS33_IN |
| rstn | G2 / 7 | LVCMOS33_IN |
| **PCI Address and Data** | | |
| ad(0) | A3 / 0 | PCI33_BIDI |
| ad(1) | D6 / 0 | PCI33_BIDI |
| ad(2) | D5 / 0 | PCI33_BIDI |
| ad(3) | B4 / 0 | PCI33_BIDI |
| ad(4) | B5 / 0 | PCI33_BIDI |
| ad(5) | C4 / 0 | PCI33_BIDI |
| ad(6) | C5 / 0 | PCI33_BIDI |
| ad(7) | E7 / 0 | PCI33_BIDI |
| ad(8) | E6 / 0 | PCI33_BIDI |
| ad(9) | A5 / 0 | PCI33_BIDI |
| ad(10) | A4 / 0 | PCI33_BIDI |
| ad(11) | C6 / 0 | PCI33_BIDI |
| ad(12) | C7 / 0 | PCI33_BIDI |
| ad(13) | B6 / 0 | PCI33_BIDI |
| ad(14) | B7 / 0 | PCI33_BIDI |
| ad(15) | A6 / 0 | PCI33_BIDI |
| ad(16) | E9 / 1 | PCI33_BIDI |
| ad(17) | E8 / 1 | PCI33_BIDI |
| ad(18) | A9 / 1 | PCI33_BIDI |
| ad(19) | A10 / 1 | PCI33_BIDI |
| ad(20) | C10 / 1 | PCI33_BIDI |
| ad(21) | C9 / 1 | PCI33_BIDI |
| ad(22) | D10 / 1 | PCI33_BIDI |
| ad(23) | D9 / 1 | PCI33_BIDI |
| ad(24) | B10 / 1 | PCI33_BIDI |
| ad(25) | B9 / 1 | PCI33_BIDI |
| ad(26) | A12 / 1 | PCI33_BIDI |
| ad(27) | A11 / 1 | PCI33_BIDI |
| ad(28) | B12 / 1 | PCI33_BIDI |
| ad(29) | B11 / 1 | PCI33_BIDI |
| ad(30) | C12 / 1 | PCI33_BIDI |
| ad(31) | C11 / 1 | PCI33_BIDI |
| cben(0) | A14 / 1 | PCI33_BIDI |

*Table B-9. PCI Pin Assignments (Continued)*

| Signal Name | Pin/Bank | Buffer Type |
|---|---|---|
| cben(1) | E10 / 1 | PCI33_BIDI |
| cben(2) | D12 / 1 | PCI33_BIDI |
| cben(3) | B13 / 1 | PCI33_BIDI |
| par | D11 / 1 | PCI33_BIDI |
| **PCI Interface Controls** | | |
| Framen | D7 / 0 | PCI33_BIDI |
| irdyn | C8 / 0 | PCI33_BIDI |
| trdyn | A7 / 0 | PCI33_BIDI |
| stopn | B8 / 0 | PCI33_BIDI |
| idsel | C14 / 1 | PCI33_IN |
| devseln | D8 / 0 | PCI33_BIDI |
| perrn | B14 / 1 | PCI33_BIDI |
| serrn | E11 / 1 | PCI33_BIDI |
| **PCI Interrupts** | | |
| intan | A15 / 1 | PCI33_OUT |

## PCI Pin Assignments for Master/Target 33MHz 32-Bit Bus

The PCI Master/Target 33MHz 32-bit core is optimized for LCMXO2280C-5FT324C. An example pin assignment, optimized for best performance, is given in Table B-10. Refer to the readme file included with the core package for further information.

*Table B-10. PCI Pin Assignments*

| Signal Name | Pin/Bank | Buffer Type |
|---|---|---|
| **PCI System Pins** | | |
| clk | R3 / 6 | LVCMOS33_IN |
| **PCI Address and Data** | | |
| ad(0) | B5 / 0 | PCI33_BIDI |
| ad(1) | C6 / 0 | PCI33_BIDI |
| ad(2) | A5 / 0 | PCI33_BIDI |
| ad(3) | E7 / 0 | PCI33_BIDI |
| ad(4) | D7 / 0 | PCI33_BIDI |
| ad(5) | E8 / 0 | PCI33_BIDI |
| ad(6) | C7 / 0 | PCI33_BIDI |
| ad(7) | F8 / 0 | PCI33_BIDI |
| ad(8) | D8 / 0 | PCI33_BIDI |
| ad(9) | B6 / 0 | PCI33_BIDI |
| ad(10) | A6 / 0 | PCI33_BIDI |
| ad(11) | B7 / 0 | PCI33_BIDI |
| ad(12) | A7 / 0 | PCI33_BIDI |
| ad(13) | C8 / 0 | PCI33_BIDI |
| ad(14) | B8 / 0 | PCI33_BIDI |
| ad(15) | A8 / 0 | PCI33_BIDI |
| ad(16) | D9 / 1 | PCI33_BIDI |

*Table B-10. PCI Pin Assignments (Continued)*

| Signal Name | Pin/Bank | Buffer Type |
|---|---|---|
| ad(17) | E9 / 1 | PCI33_BIDI |
| ad(18) | E10 / 1 | PCI33_BIDI |
| ad(19) | C10 / 1 | PCI33_BIDI |
| ad(20) | B11 / 1 | PCI33_BIDI |
| ad(21) | A11 / 1 | PCI33_BIDI |
| ad(22) | F10 / 1 | PCI33_BIDI |
| ad(23) | D10 / 1 | PCI33_BIDI |
| ad(24) | C11 / 1 | PCI33_BIDI |
| ad(25) | A12 / 1 | PCI33_BIDI |
| ad(26) | E11 / 1 | PCI33_BIDI |
| ad(27) | D11 / 1 | PCI33_BIDI |
| ad(28) | C12 / 1 | PCI33_BIDI |
| ad(29) | B12 / 1 | PCI33_BIDI |
| ad(30) | B13 / 1 | PCI33_BIDI |
| ad(31) | A13 / 1 | PCI33_BIDI |
| cben(0) | D12 / 1 | PCI33_BIDI |
| cben(1) | A15 / 1 | PCI33_BIDI |
| cben(2) | B14 / 1 | PCI33_BIDI |
| cben(3) | B16 / 1 | PCI33_BIDI |
| par | A14 / 1 | PCI33_BIDI |
| **PCI Interface Controls** | | |
| Framen | B10 / 0 | PCI33_BIDI |
| irdyn | A9 / 0 | PCI33_BIDI |
| trdyn | B9 / 0 | PCI33_BIDI |
| stopn | C9 / 0 | PCI33_BIDI |
| idsel | C14 / 1 | PCI33_IN |
| devseln | A10 / 0 | PCI33_BIDI |
| perrn | E12 / 1 | PCI33_BIDI |
| serrn | B15 / 1 | PCI33_BIDI |
| **PCI Interrupts** | | |
| intan | F12 / 1 | PCI33_OUT |
| **PCI Bus Arbitration** | | |
| gntn | F11 / 1 | PCI33_IN |
| reqn | C13 / 1 | PCI33_OUT |

# PCI Assignment Considerations for LatticeSC Devices

## PCI Pin Assignments for Master/Target 33 MHz 32-bit Bus

The PCI Master/Target 33 MHz 32-bit core is optimized for LFSC3GA25E-5F900C. An example pin assignment, optimized for best performance, is given in Table B-11. Refer to the readme file included with the core package for further information.

***Table B-11. PCI Pin Assignments***

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| **PCI System Pins** | | |
| clk | AH1/5 | PCI33_IN |
| rstn | AJ11/4 | PCI33_IN |
| **PCI Address and Data** | | |
| ad[0] | AJ8/5 | PCI33_BIDI |
| ad[1] | AJ7/5 | PCI33_BIDI |
| ad[2] | AJ6/5 | PCI33_BIDI |
| ad[3] | AJ5/5 | PCI33_BIDI |
| ad[4] | AK5/5 | PCI33_BIDI |
| ad[5] | AK4/5 | PCI33_BIDI |
| ad[6] | AE13/5 | PCI33_BIDI |
| ad[7] | AE12/5 | PCI33_BIDI |
| ad[8] | AH8/5 | PCI33_BIDI |
| ad[9] | AH7/5 | PCI33_BIDI |
| ad[10] | AF10/5 | PCI33_BIDI |
| ad[11] | AE11/5 | PCI33_BIDI |
| ad[12] | AJ4/5 | PCI33_BIDI |
| ad[13] | AK3/5 | PCI33_BIDI |
| ad[14] | AE10/5 | PCI33_BIDI |
| ad[15] | AF9/5 | PCI33_BIDI |
| ad[16] | AJ3/5 | PCI33_BIDI |
| ad[17] | AH3/5 | PCI33_BIDI |
| ad[18] | AG8/5 | PCI33_BIDI |
| ad[19] | AF8/5 | PCI33_BIDI |
| ad[20] | AG5/5 | PCI33_BIDI |
| ad[21] | AH4/5 | PCI33_BIDI |
| ad[22] | AF6/5 | PCI33_BIDI |
| ad[23] | AF7/5 | PCI33_BIDI |
| ad[24] | AD8/5 | PCI33_BIDI |
| ad[25] | AD7/5 | PCI33_BIDI |
| ad[26] | AK2/5 | PCI33_BIDI |
| ad[27] | AJ2/5 | PCI33_BIDI |
| ad[28] | AD6/5 | PCI33_BIDI |
| ad[29] | AH2/5 | PCI33_BIDI |
| ad[30] | AG3/5 | PCI33_BIDI |
| ad[31] | AE5/5 | PCI33_BIDI |
| cben[0] | AH10/5 | PCI33_BIDI |
| cben[1] | AH11/5 | PCI33_BIDI |

*Table B-11. PCI Pin Assignments (Continued)*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| cben[2] | AF13/5 | PCI33_BIDI |
| cben[3] | AE14/5 | PCI33_BIDI |
| par | AG14/5 | PCI33_BIDI |
| **PCI Interface Controls** | | |
| framen | AF15/5 | PCI33_BIDI |
| irdyn | AK7/5 | PCI33_BIDI |
| trdyn | AK6/5 | PCI33_BIDI |
| stopn | AH13/5 | PCI33_BIDI |
| idsel | AG13/5 | PCI33_IN |
| devseln | AK8/5 | PCI33_BIDI |
| perrn | AK9/5 | PCI33_BIDI |
| serrn | AH14/5 | PCI33_BIDI |
| **PCI Interrupts** | | |
| intan | AJ12/5 | PCI33_OUT |
| **PCI Bus Arbitration** | | |
| gntn | AF4/5 | PCI33_IN |
| reqn | AJ1/5 | PCI33_OUT |

## PCI Pin Assignments for Master/Target 33 MHz 64-bit Bus

The PCI Master/Target 33 MHz 64-bit core is optimized for LFSC3GA25E-5F900C. An example pin assignment, optimized for best performance, is given in Table B-12. Refer to the readme file included with the core package for further information.

*Table B-12. PCI Pin Assignments*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| **PCI System Pins** | | |
| clk | AH1/5 | PCI33_IN |
| rstn | AJ11/5 | PCI33_IN |
| **PCI Address and Data** | | |
| ad[0] | AJ8/5 | PCI33_BIDI |
| ad[1] | AJ7/5 | PCI33_BIDI |
| ad[2] | AJ6/5 | PCI33_BIDI |
| ad[3] | AJ5/5 | PCI33_BIDI |
| ad[4] | AK5/5 | PCI33_BIDI |
| ad[5] | AK4/5 | PCI33_BIDI |
| ad[6] | AE13/5 | PCI33_BIDI |
| ad[7] | AE12/5 | PCI33_BIDI |
| ad[8] | AH8/5 | PCI33_BIDI |
| ad[9] | AH7/5 | PCI33_BIDI |
| ad[10] | AF10/5 | PCI33_BIDI |
| ad[11] | AE11/5 | PCI33_BIDI |
| ad[12] | AJ4/5 | PCI33_BIDI |
| ad[13] | AK3/5 | PCI33_BIDI |

*Table B-12. PCI Pin Assignments (Continued)*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| ad[14] | AE10/5 | PCI33_BIDI |
| ad[15] | AF9/5 | PCI33_BIDI |
| ad[16] | AJ3/5 | PCI33_BIDI |
| ad[17] | AH3/5 | PCI33_BIDI |
| ad[18] | AG8/5 | PCI33_BIDI |
| ad[19] | AF8/5 | PCI33_BIDI |
| ad[20] | AG5/5 | PCI33_BIDI |
| ad[21] | AH4/5 | PCI33_BIDI |
| ad[22] | AF6/5 | PCI33_BIDI |
| ad[23] | AF7/5 | PCI33_BIDI |
| ad[24] | AD8/5 | PCI33_BIDI |
| ad[25] | AD7/5 | PCI33_BIDI |
| ad[26] | AK2/5 | PCI33_BIDI |
| ad[27] | AJ2/5 | PCI33_BIDI |
| ad[28] | AD6/5 | PCI33_BIDI |
| ad[29] | AH2/5 | PCI33_BIDI |
| ad[30] | AG3/5 | PCI33_BIDI |
| ad[31] | AE5/5 | PCI33_BIDI |
| ad[32] | AK23/4 | PCI33_BIDI |
| ad[33] | AK22/4 | PCI33_BIDI |
| ad[34] | AF19/4 | PCI33_BIDI |
| ad[35] | AG19/4 | PCI33_BIDI |
| ad[36] | AJ21/4 | PCI33_BIDI |
| ad[37] | AJ20/4 | PCI33_BIDI |
| ad[38] | AG18/4 | PCI33_BIDI |
| ad[39] | AF18/4 | PCI33_BIDI |
| ad[40] | AK20/4 | PCI33_BIDI |
| ad[41] | AJ19/4 | PCI33_BIDI |
| ad[42] | AJ18/4 | PCI33_BIDI |
| ad[43] | AG17/4 | PCI33_BIDI |
| ad[44] | AF17/4 | PCI33_BIDI |
| ad[45] | AH18/4 | PCI33_BIDI |
| ad[46] | AH17/4 | PCI33_BIDI |
| ad[47] | AK19/4 | PCI33_BIDI |
| ad[48] | AK18/4 | PCI33_BIDI |
| ad[49] | AG16/4 | PCI33_BIDI |
| ad[50] | AH16/4 | PCI33_BIDI |
| ad[51] | AF16/4 | PCI33_BIDI |
| ad[52] | AE16/4 | PCI33_BIDI |
| ad[53] | AJ17/4 | PCI33_BIDI |
| ad[54] | AJ16/4 | PCI33_BIDI |

*Table B-12. PCI Pin Assignments (Continued)*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| ad[55] | AK17/4 | PCI33_BIDI |
| ad[56] | AK16/4 | PCI33_BIDI |
| ad[57] | AK15/5 | PCI33_BIDI |
| ad[58] | AK14/5 | PCI33_BIDI |
| ad[59] | AJ15/5 | PCI33_BIDI |
| ad[60] | AJ14/5 | PCI33_BIDI |
| ad[61] | AK13/5 | PCI33_BIDI |
| ad[62] | AK12/5 | PCI33_BIDI |
| ad[63] | AE15/5 | PCI33_BIDI |
| cben[0] | AH10/5 | PCI33_BIDI |
| cben[1] | AH11/5 | PCI33_BIDI |
| cben[2] | AF13/5 | PCI33_BIDI |
| cben[3] | AE14/5 | PCI33_BIDI |
| cben[4] | AG15/5 | PCI33_BIDI |
| cben[5] | AH12/5 | PCI33_BIDI |
| cben[6] | AJ13/5 | PCI33_BIDI |
| cben[7] | AD15/5 | PCI33_BIDI |
| par | AG14/5 | PCI33_BIDI |
| par64 | AK10/5 | PCI33_BIDI |
| **PCI Interface Controls** | | |
| framen | AF15/5 | PCI33_BIDI |
| irdyn | AK7/5 | PCI33_BIDI |
| trdyn | AK6/5 | PCI33_BIDI |
| stopn | AH13/5 | PCI33_BIDI |
| idsel | AG13/5 | PCI33_IN |
| devseln | AK8/5 | PCI33_BIDI |
| perrn | AK9/5 | PCI33_BIDI |
| serrn | AH14/5 | PCI33_BIDI |
| ack64n | AH15/5 | PCI33_BIDI |
| req64n | K11/5 | PCI33_BIDI |
| **PCI Interrupts** | | |
| intan | AJ12/5 | PCI33_OUT |
| **PCI Bus Arbitration** | | |
| gntn | AF4/5 | PCI33_IN |
| reqn | AJ1/5 | PCI33_OUT |

## PCI Pin Assignments for Target 33 MHz 32-bit Bus

The PCI Target 33 MHz 32-bit core is optimized for LFSC3GA25E-5F900C. An example pin assignment, optimized for best performance, is given in Table B-13. Refer to the readme file included with the core package for further information.

***Table B-13. PCI Pin Assignments***

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| **PCI System Pins** | | |
| clk | AH1/5 | PCI33_IN |
| rstn | AJ11/4 | PCI33_IN |
| **PCI Address and Data** | | |
| ad[0] | AJ8/5 | PCI33_BIDI |
| ad[1] | AJ7/5 | PCI33_BIDI |
| ad[2] | AJ6/5 | PCI33_BIDI |
| ad[3] | AJ5/5 | PCI33_BIDI |
| ad[4] | AK5/5 | PCI33_BIDI |
| ad[5] | AK4/5 | PCI33_BIDI |
| ad[6] | AE13/5 | PCI33_BIDI |
| ad[7] | AE12/5 | PCI33_BIDI |
| ad[8] | AH8/5 | PCI33_BIDI |
| ad[9] | AH7/5 | PCI33_BIDI |
| ad[10] | AF10/5 | PCI33_BIDI |
| ad[11] | AE11/5 | PCI33_BIDI |
| ad[12] | AJ4/5 | PCI33_BIDI |
| ad[13] | AK3/5 | PCI33_BIDI |
| ad[14] | AE10/5 | PCI33_BIDI |
| ad[15] | AF9/5 | PCI33_BIDI |
| ad[16] | AJ3/5 | PCI33_BIDI |
| ad[17] | AH3/5 | PCI33_BIDI |
| ad[18] | AG8/5 | PCI33_BIDI |
| ad[19] | AF8/5 | PCI33_BIDI |
| ad[20] | AG5/5 | PCI33_BIDI |
| ad[21] | AH4/5 | PCI33_BIDI |
| ad[22] | AF6/5 | PCI33_BIDI |
| ad[23] | AF7/5 | PCI33_BIDI |
| ad[24] | AD8/5 | PCI33_BIDI |
| ad[25] | AD7/5 | PCI33_BIDI |
| ad[26] | AK2/5 | PCI33_BIDI |
| ad[27] | AJ2/5 | PCI33_BIDI |
| ad[28] | AD6/5 | PCI33_BIDI |
| ad[29] | AH2/5 | PCI33_BIDI |
| ad[30] | AG3/5 | PCI33_BIDI |
| ad[31] | AE5/5 | PCI33_BIDI |
| cben[0] | AH10/5 | PCI33_IN |
| cben[1] | AH11/5 | PCI33_IN |
| cben[2] | AF13/5 | PCI33_IN |

*Table B-13. PCI Pin Assignments (Continued)*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| cben[3] | AE14/5 | PCI33_IN |
| par | AG14/5 | PCI33_BIDI |
| **PCI Interface Controls** | | |
| framen | AF15/5 | PCI33_IN |
| irdyn | AK7/5 | PCI33_IN |
| trdyn | AK6/5 | PCI33_OUT |
| stopn | AH13/5 | PCI33_OUT |
| idsel | AG13/5 | PCI33_IN |
| devseln | AK8/5 | PCI33_OUT |
| perrn | AK9/5 | PCI33_OUT |
| serrn | AH14/5 | PCI33_OUT |
| **PCI Interrupts** | | |
| intan | AJ12/5 | PCI33_OUT |

## PCI Pin Assignments for Target 33 MHz 64-bit Bus

The PCI Target 33 MHz 64-bit core is optimized for LFSC3GA25E-5F900C. An example pin assignment, optimized for best performance, is given in Table B-14. Refer to the readme file included with the core package for further information.

*Table B-14. PCI Pin Assignments*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| **PCI System Pins** | | |
| clk | AH1/5 | PCI33_IN |
| rstn | AJ11/5 | PCI33_IN |
| **PCI Address and Data** | | |
| ad[0] | AJ8/5 | PCI33_BIDI |
| ad[1] | AJ7/5 | PCI33_BIDI |
| ad[2] | AJ6/5 | PCI33_BIDI |
| ad[3] | AJ5/5 | PCI33_BIDI |
| ad[4] | AK5/5 | PCI33_BIDI |
| ad[5] | AK4/5 | PCI33_BIDI |
| ad[6] | AE13/5 | PCI33_BIDI |
| ad[7] | AE12/5 | PCI33_BIDI |
| ad[8] | AH8/5 | PCI33_BIDI |
| ad[9] | AH7/5 | PCI33_BIDI |
| ad[10] | AF10/5 | PCI33_BIDI |
| ad[11] | AE11/5 | PCI33_BIDI |
| ad[12] | AJ4/5 | PCI33_BIDI |
| ad[13] | AK3/5 | PCI33_BIDI |
| ad[14] | AE10/5 | PCI33_BIDI |
| ad[15] | AF9/5 | PCI33_BIDI |
| ad[16] | AJ3/5 | PCI33_BIDI |
| ad[17] | AH3/5 | PCI33_BIDI |
| ad[18] | AG8/5 | PCI33_BIDI |
| ad[19] | AF8/5 | PCI33_BIDI |

*Table B-14. PCI Pin Assignments (Continued)*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| ad[20] | AG5/5 | PCI33_BIDI |
| ad[21] | AH4/5 | PCI33_BIDI |
| ad[22] | AF6/5 | PCI33_BIDI |
| ad[23] | AF7/5 | PCI33_BIDI |
| ad[24] | AD8/5 | PCI33_BIDI |
| ad[25] | AD7/5 | PCI33_BIDI |
| ad[26] | AK2/5 | PCI33_BIDI |
| ad[27] | AJ2/5 | PCI33_BIDI |
| ad[28] | AD6/5 | PCI33_BIDI |
| ad[29] | AH2/5 | PCI33_BIDI |
| ad[30] | AG3/5 | PCI33_BIDI |
| ad[31] | AE5/5 | PCI33_BIDI |
| ad[32] | AK23/4 | PCI33_BIDI |
| ad[33] | AK22/4 | PCI33_BIDI |
| ad[34] | AF19/4 | PCI33_BIDI |
| ad[35] | AG19/4 | PCI33_BIDI |
| ad[36] | AJ21/4 | PCI33_BIDI |
| ad[37] | AJ20/4 | PCI33_BIDI |
| ad[38] | AG18/4 | PCI33_BIDI |
| ad[39] | AF18/4 | PCI33_BIDI |
| ad[40] | AK20/4 | PCI33_BIDI |
| ad[41] | AJ19/4 | PCI33_BIDI |
| ad[42] | AJ18/4 | PCI33_BIDI |
| ad[43] | AG17/4 | PCI33_BIDI |
| ad[44] | AF17/4 | PCI33_BIDI |
| ad[45] | AH18/4 | PCI33_BIDI |
| ad[46] | AH17/4 | PCI33_BIDI |
| ad[47] | AK19/4 | PCI33_BIDI |
| ad[48] | AK18/4 | PCI33_BIDI |
| ad[49] | AG16/4 | PCI33_BIDI |
| ad[50] | AH16/4 | PCI33_BIDI |
| ad[51] | AF16/4 | PCI33_BIDI |
| ad[52] | AE16/4 | PCI33_BIDI |
| ad[53] | AJ17/4 | PCI33_BIDI |
| ad[54] | AJ16/4 | PCI33_BIDI |
| ad[55] | AK17/4 | PCI33_BIDI |
| ad[56] | AK16/4 | PCI33_BIDI |
| ad[57] | AK15/5 | PCI33_BIDI |
| ad[58] | AK14/5 | PCI33_BIDI |
| ad[59] | AJ15/5 | PCI33_BIDI |
| ad[60] | AJ14/5 | PCI33_BIDI |
| ad[61] | AK13/5 | PCI33_BIDI |
| ad[62] | AK12/5 | PCI33_BIDI |
| ad[63] | AE15/5 | PCI33_BIDI |

*Table B-14. PCI Pin Assignments (Continued)*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| cben[0] | AH10/5 | PCI33_IN |
| cben[1] | AH11/5 | PCI33_IN |
| cben[2] | AF13/5 | PCI33_IN |
| cben[3] | AE14/5 | PCI33_IN |
| cben[4] | AG15/5 | PCI33_IN |
| cben[5] | AH12/5 | PCI33_IN |
| cben[6] | AJ13/5 | PCI33_IN |
| cben[7] | AD15/5 | PCI33_IN |
| par | AG14/5 | PCI33_BIDI |
| par64 | AK10/5 | PCI33_BIDI |
| **PCI Interface Controls** | | |
| framen | AF15/5 | PCI33_IN |
| irdyn | AK7/5 | PCI33_IN |
| trdyn | AK6/5 | PCI33_OUT |
| stopn | AH13 | PCI33_OUT |
| idsel | AG13/5 | PCI33_IN |
| devseln | AK8/5 | PCI33_OUT |
| perrn | AK9/5 | PCI33_OUT |
| serrn | AH14/5 | PCI33_OUT |
| ack64n | AH15/5 | PCI33_OUT |
| req64n | K11/5 | PCI33_IN |
| **PCI Interrupts** | | |
| intan | AJ12/5 | PCI33_OUT |

## PCI Pin Assignments for Master/Target 66 MHz 32-bit Bus

The PCI Master/Target 66 MHz 32-bit core is optimized for LFSC3GA25E-5F900C. An example pin assignment, optimized for best performance, is given in Table B-15. Refer to the readme file included with the core package for further information.

*Table B-15. PCI Pin Assignments*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| **PCI System Pins** | | |
| clk | AH1/5 | PCI33_IN |
| rstn | AJ11/5 | PCI33_IN |
| **PCI Address and Data** | | |
| ad[0] | AJ8/5 | PCI33_BIDI |
| ad[1] | AJ7/5 | PCI33_BIDI |
| ad[2] | AJ6/5 | PCI33_BIDI |
| ad[3] | AJ5/5 | PCI33_BIDI |
| ad[4] | AK5/5 | PCI33_BIDI |
| ad[5] | AK4/5 | PCI33_BIDI |
| ad[6] | AE13/5 | PCI33_BIDI |
| ad[7] | AE12/5 | PCI33_BIDI |
| ad[8] | AH8/5 | PCI33_BIDI |
| ad[9] | AH7/5 | PCI33_BIDI |

*Table B-15. PCI Pin Assignments (Continued)*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| ad[10] | AF10/5 | PCI33_BIDI |
| ad[11] | AE11/5 | PCI33_BIDI |
| ad[12] | AJ4/5 | PCI33_BIDI |
| ad[13] | AK3/5 | PCI33_BIDI |
| ad[14] | AE10/5 | PCI33_BIDI |
| ad[15] | AF9/5 | PCI33_BIDI |
| ad[16] | AJ3/5 | PCI33_BIDI |
| ad[17] | AH3/5 | PCI33_BIDI |
| ad[18] | AG8/5 | PCI33_BIDI |
| ad[19] | AF8/5 | PCI33_BIDI |
| ad[20] | AG5/5 | PCI33_BIDI |
| ad[21] | AH4/5 | PCI33_BIDI |
| ad[22] | AF6/5 | PCI33_BIDI |
| ad[23] | AF7/5 | PCI33_BIDI |
| ad[24] | AD8/5 | PCI33_BIDI |
| ad[25] | AD7/5 | PCI33_BIDI |
| ad[26] | AK2/5 | PCI33_BIDI |
| ad[27] | AJ2/5 | PCI33_BIDI |
| ad[28] | AD6/5 | PCI33_BIDI |
| ad[29] | AH2/5 | PCI33_BIDI |
| ad[30] | AG3/5 | PCI33_BIDI |
| ad[31] | AE5/5 | PCI33_BIDI |
| cben[0] | AH10/5 | PCI33_BIDI |
| cben[1] | AH11/5 | PCI33_BIDI |
| cben[2] | AF13/5 | PCI33_BIDI |
| cben[3] | AE14/5 | PCI33_BIDI |
| par | AG14/5 | PCI33_BIDI |
| **PCI Interface Controls** | | |
| framen | AF15/5 | PCI33_BIDI |
| irdyn | AK7/5 | PCI33_BIDI |
| trdyn | AK6/5 | PCI33_BIDI |
| stopn | AH13/5 | PCI33_BIDI |
| idsel | AG13/5 | PCI33_IN |
| devseln | AK8/5 | PCI33_BIDI |
| perrn | AK9/5 | PCI33_BIDI |
| serrn | AH14/5 | PCI33_BIDI |
| **PCI Interrupts** | | |
| intan | AJ12/5 | PCI33_OUT |
| **PCI Bus Arbitration** | | |
| reqn | AJ1/5 | PCI33_IN |
| gntn | AF4/5 | PCI33_OUT |

## PCI Pin Assignments for Master/Target 66 MHz 64-bit Bus

The PCI Master/Target 66 MHz 64-bit core is optimized for LFSC3GA25E-5F900C. An example pin assignment, optimized for best performance, is given in Table B-16. Refer to the readme file included with the core package for further information.

***Table B-16. PCI Pin Assignments***

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| **PCI System Pins** | | |
| clk | AH1/5 | PCI33_IN |
| rstn | AJ11/5 | PCI33_IN |
| **PCI Address and Data** | | |
| ad[0] | AJ8/5 | PCI33_BIDI |
| ad[1] | AJ7/5 | PCI33_BIDI |
| ad[2] | AJ6/5 | PCI33_BIDI |
| ad[3] | AJ5/5 | PCI33_BIDI |
| ad[4] | AK5/5 | PCI33_BIDI |
| ad[5] | AK4/5 | PCI33_BIDI |
| ad[6] | AE13/5 | PCI33_BIDI |
| ad[7] | AE12/5 | PCI33_BIDI |
| ad[8] | AH8/5 | PCI33_BIDI |
| ad[9] | AH7/5 | PCI33_BIDI |
| ad[10] | AF10/5 | PCI33_BIDI |
| ad[11] | AE11/5 | PCI33_BIDI |
| ad[12] | AJ4/5 | PCI33_BIDI |
| ad[13] | AK3/5 | PCI33_BIDI |
| ad[14] | AE10/5 | PCI33_BIDI |
| ad[15] | AF9/5 | PCI33_BIDI |
| ad[16] | AJ3/5 | PCI33_BIDI |
| ad[17] | AH3/5 | PCI33_BIDI |
| ad[18] | AG8/5 | PCI33_BIDI |
| ad[19] | AF8/5 | PCI33_BIDI |
| ad[20] | AG5/5 | PCI33_BIDI |
| ad[21] | AH4/5 | PCI33_BIDI |
| ad[22] | AF6/5 | PCI33_BIDI |
| ad[23] | AF7/5 | PCI33_BIDI |
| ad[24] | AD8/5 | PCI33_BIDI |
| ad[25] | AD7/5 | PCI33_BIDI |
| ad[26] | AK2/5 | PCI33_BIDI |
| ad[27] | AJ2/5 | PCI33_BIDI |
| ad[28] | AD6/5 | PCI33_BIDI |
| ad[29] | AH2/5 | PCI33_BIDI |
| ad[30] | AG3/5 | PCI33_BIDI |
| ad[31] | AE5/5 | PCI33_BIDI |
| ad[32] | AK23/4 | PCI33_BIDI |
| ad[33] | AK22/4 | PCI33_BIDI |
| ad[34] | AF19/4 | PCI33_BIDI |

*Table B-16. PCI Pin Assignments (Continued)*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| ad[35] | AG19/4 | PCI33_BIDI |
| ad[36] | AJ21/4 | PCI33_BIDI |
| ad[37] | AJ20/4 | PCI33_BIDI |
| ad[38] | AG18/4 | PCI33_BIDI |
| ad[39] | AF18/4 | PCI33_BIDI |
| ad[40] | AK20/4 | PCI33_BIDI |
| ad[41] | AJ19/4 | PCI33_BIDI |
| ad[42] | AJ18/4 | PCI33_BIDI |
| ad[43] | AG17/4 | PCI33_BIDI |
| ad[44] | AF17/4 | PCI33_BIDI |
| ad[45] | AH18/4 | PCI33_BIDI |
| ad[46] | AH17/4 | PCI33_BIDI |
| ad[47] | AK19/4 | PCI33_BIDI |
| ad[48] | AK18/4 | PCI33_BIDI |
| ad[49] | AG16/4 | PCI33_BIDI |
| ad[50] | AH16/4 | PCI33_BIDI |
| ad[51] | AF16/4 | PCI33_BIDI |
| ad[52] | AE16/4 | PCI33_BIDI |
| ad[53] | AJ17/4 | PCI33_BIDI |
| ad[54] | AJ16/4 | PCI33_BIDI |
| ad[55] | AK17/4 | PCI33_BIDI |
| ad[56] | AK16/4 | PCI33_BIDI |
| ad[57] | AK15/5 | PCI33_BIDI |
| ad[58] | AK14/5 | PCI33_BIDI |
| ad[59] | AJ15/5 | PCI33_BIDI |
| ad[60] | AJ14/5 | PCI33_BIDI |
| ad[61] | AK13/5 | PCI33_BIDI |
| ad[62] | AK12/5 | PCI33_BIDI |
| ad[63] | AE15/5 | PCI33_BIDI |
| cben[0] | AH10/5 | PCI33_BIDI |
| cben[1] | AH11/5 | PCI33_BIDI |
| cben[2] | AF13/5 | PCI33_BIDI |
| cben[3] | AE14/5 | PCI33_BIDI |
| cben[4] | AG15/5 | PCI33_BIDI |
| cben[5] | AH12/5 | PCI33_BIDI |
| cben[6] | AJ13/5 | PCI33_BIDI |
| cben[7] | AD15/5 | PCI33_BIDI |
| par64 | AK10/5 | PCI33_BIDI |
| par | AG14/5 | PCI33_BIDI |
| **PCI Interface Controls** | | |
| framen | AF15/5 | PCI33_BIDI |
| irdyn | AK7/5 | PCI33_BIDI |
| trdyn | AK6/5 | PCI33_BIDI |
| stopn | AH13/5 | PCI33_BIDI |

*Table B-16. PCI Pin Assignments (Continued)*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| idsel | AG13/5 | PCI33_IN |
| devseln | AK8/5 | PCI33_BIDI |
| perrn | AK9/5 | PCI33_BIDI |
| serrn | AH14/5 | PCI33_BIDI |
| ack64n | AH15/5 | PCI33_BIDI |
| req64n | K11/5 | PCI33_BIDI |
| **PCI Interrupts** | | |
| intan | AJ12/5 | PCI33_IN |
| **PCI Bus Arbitration** | | |
| gntn | AF4/5 | PCI33_IN |
| reqn | AJ1/5 | PCI33_OUT |

## PCI Pin Assignments for Target 66 MHz 32-bit Bus

The PCI Target 66 MHz 32-bit core is optimized for LFSC3GA25E-5F900C. An example pin assignment, optimized for best performance, is given in Table B-17. Refer to the readme file included with the core package for further information.

*Table B-17. PCI Pin Assignments*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| **PCI System Pins** | | |
| clk | AH1/5 | PCI33_IN |
| rstn | AJ11/5 | PCI33_IN |
| **PCI Address and Data** | | |
| ad[0] | AJ8/5 | PCI33_BIDI |
| ad[1] | AJ7/5 | PCI33_BIDI |
| ad[2] | AJ6/5 | PCI33_BIDI |
| ad[3] | AJ5/5 | PCI33_BIDI |
| ad[4] | AK5/5 | PCI33_BIDI |
| ad[5] | AK4/5 | PCI33_BIDI |
| ad[6] | AE13/5 | PCI33_BIDI |
| ad[7] | AE12/5 | PCI33_BIDI |
| ad[8] | AH8/5 | PCI33_BIDI |
| ad[9] | AH7/5 | PCI33_BIDI |
| ad[10] | AF10/5 | PCI33_BIDI |
| ad[11] | AE11/5 | PCI33_BIDI |
| ad[12] | AJ4/5 | PCI33_BIDI |
| ad[13] | AK3/5 | PCI33_BIDI |
| ad[14] | AE10/5 | PCI33_BIDI |
| ad[15] | AF9/5 | PCI33_BIDI |
| ad[16] | AJ3/5 | PCI33_BIDI |
| ad[17] | AH3/5 | PCI33_BIDI |
| ad[18] | AG8/5 | PCI33_BIDI |
| ad[19] | AF8/5 | PCI33_BIDI |
| ad[20] | AG5/5 | PCI33_BIDI |
| ad[21] | AH4/5 | PCI33_BIDI |

*Table B-17. PCI Pin Assignments (Continued)*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| ad[22] | AF6/5 | PCI33_BIDI |
| ad[23] | AF7/5 | PCI33_BIDI |
| ad[24] | AD8/5 | PCI33_BIDI |
| ad[25] | AD7/5 | PCI33_BIDI |
| ad[26] | AK2/5 | PCI33_BIDI |
| ad[27] | AJ2/5 | PCI33_BIDI |
| ad[28] | AD6/5 | PCI33_BIDI |
| ad[29] | AH2/5 | PCI33_BIDI |
| ad[30] | AG3/5 | PCI33_BIDI |
| ad[31] | AE5/5 | PCI33_BIDI |
| ad[32] | AK23/4 | PCI33_BIDI |
| ad[33] | AK22/4 | PCI33_BIDI |
| ad[34] | AF19/4 | PCI33_BIDI |
| ad[35] | AG19/4 | PCI33_BIDI |
| ad[36] | AJ21/4 | PCI33_BIDI |
| ad[37] | AJ20/4 | PCI33_BIDI |
| ad[38] | AG18/4 | PCI33_BIDI |
| ad[39] | AF18/4 | PCI33_BIDI |
| ad[40] | AK20/4 | PCI33_BIDI |
| ad[41] | AJ19/4 | PCI33_BIDI |
| ad[42] | AJ18/4 | PCI33_BIDI |
| ad[43] | AG17/4 | PCI33_BIDI |
| ad[44] | AF17/4 | PCI33_BIDI |
| ad[45] | AH18/4 | PCI33_BIDI |
| ad[46] | AH17/4 | PCI33_BIDI |
| ad[47] | AK19/4 | PCI33_BIDI |
| ad[48] | AK18/4 | PCI33_BIDI |
| ad[49] | AG16/4 | PCI33_BIDI |
| ad[50] | AH16/4 | PCI33_BIDI |
| ad[51] | AF16/4 | PCI33_BIDI |
| ad[52] | AE16/4 | PCI33_BIDI |
| ad[53] | AJ17/4 | PCI33_BIDI |
| ad[54] | AJ16/4 | PCI33_BIDI |
| ad[55] | AK17/4 | PCI33_BIDI |
| ad[56] | AK16/4 | PCI33_BIDI |
| ad[57] | AK15/5 | PCI33_BIDI |
| ad[58] | AK14/5 | PCI33_BIDI |
| ad[59] | AJ15/5 | PCI33_BIDI |
| ad[60] | AJ14/5 | PCI33_BIDI |
| ad[61] | AK13/5 | PCI33_BIDI |
| ad[62] | AK12/5 | PCI33_BIDI |
| ad[63] | AE15/5 | PCI33_BIDI |
| cben[0] | AH10/5 | PCI33_BIDI |
| cben[1] | AH11/5 | PCI33_BIDI |

*Table B-17. PCI Pin Assignments (Continued)*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| cben[2] | AF13/5 | PCI33_BIDI |
| cben[3] | AE14/5 | PCI33_BIDI |
| cben[4] | AG15/5 | PCI33_BIDI |
| cben[5] | AH12/5 | PCI33_BIDI |
| cben[6] | AJ13/5 | PCI33_BIDI |
| cben[7] | AD15/5 | PCI33_BIDI |
| par64 | AK10/5 | PCI33_BIDI |
| par | AG14/5 | PCI33_BIDI |
| **PCI Interface Controls** | | |
| framen | AF15/5 | PCI33_BIDI |
| irdyn | AK7/5 | PCI33_BIDI |
| trdyn | AK6/5 | PCI33_BIDI |
| stopn | AH13/5 | PCI33_BIDI |
| idsel | AG13/5 | PCI33_IN |
| devseln | AK8/5 | PCI33_BIDI |
| perrn | AK9/5 | PCI33_BIDI |
| serrn | AH14/5 | PCI33_BIDI |
| ack64n | AH15/5 | PCI33_BIDI |
| req64n | K11/5 | PCI33_BIDI |
| **PCI Interrupts** | | |
| intan | AJ12/5 | PCI33_IN |
| **PCI Bus Arbitration** | | |
| gntn | AF4/5 | PCI33_IN |
| reqn | AJ1/5 | PCI33_OUT |

## PCI Pin Assignments for Target 66 MHz 64-bit Bus

The PCI Target 66 MHz 64-bit core is optimized for LFSC3GA25E-5F900C. An example pin assignment, optimized for best performance, is given in Table B-18. Refer to the readme file included with the core package for further information.

*Table B-18. PCI Pin Assignments*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| **PCI System Pins** | | |
| clk | AH1/5 | PCI33_IN |
| rstn | AJ11/5 | PCI33_IN |
| **PCI Address and Data** | | |
| ad[0] | AJ8/5 | PCI33_BIDI |
| ad[1] | AJ7/5 | PCI33_BIDI |
| ad[2] | AJ6/5 | PCI33_BIDI |
| ad[3] | AJ5/5 | PCI33_BIDI |
| ad[4] | AK5/5 | PCI33_BIDI |
| ad[5] | AK4/5 | PCI33_BIDI |
| ad[6] | AE13/5 | PCI33_BIDI |
| ad[7] | AE12/5 | PCI33_BIDI |
| ad[8] | AH8/5 | PCI33_BIDI |

*Table B-18. PCI Pin Assignments (Continued)*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| ad[9] | AH7/5 | PCI33_BIDI |
| ad[10] | AF10/5 | PCI33_BIDI |
| ad[11] | AE11/5 | PCI33_BIDI |
| ad[12] | AJ4/5 | PCI33_BIDI |
| ad[13] | AK3/5 | PCI33_BIDI |
| ad[14] | AE10/5 | PCI33_BIDI |
| ad[15] | AF9/5 | PCI33_BIDI |
| ad[16] | AJ3/5 | PCI33_BIDI |
| ad[17] | AH3/5 | PCI33_BIDI |
| ad[18] | AG8/5 | PCI33_BIDI |
| ad[19] | AF8/5 | PCI33_BIDI |
| ad[20] | AG5/5 | PCI33_BIDI |
| ad[21] | AH4/5 | PCI33_BIDI |
| ad[22] | AF6/5 | PCI33_BIDI |
| ad[23] | AF7/5 | PCI33_BIDI |
| ad[24] | AD8/5 | PCI33_BIDI |
| ad[25] | AD7/5 | PCI33_BIDI |
| ad[26] | AK2/5 | PCI33_BIDI |
| ad[27] | AJ2/5 | PCI33_BIDI |
| ad[28] | AD6/5 | PCI33_BIDI |
| ad[29] | AH2/5 | PCI33_BIDI |
| ad[30] | AG3/5 | PCI33_BIDI |
| ad[31] | AE5/5 | PCI33_BIDI |
| ad[32] | AK23/4 | PCI33_BIDI |
| ad[33] | AK22/4 | PCI33_BIDI |
| ad[34] | AF19/4 | PCI33_BIDI |
| ad[35] | AG19/4 | PCI33_BIDI |
| ad[36] | AJ21/4 | PCI33_BIDI |
| ad[37] | AJ20/4 | PCI33_BIDI |
| ad[38] | AG18/4 | PCI33_BIDI |
| ad[39] | AF18/4 | PCI33_BIDI |
| ad[40] | AK20/4 | PCI33_BIDI |
| ad[41] | AJ19/4 | PCI33_BIDI |
| ad[42] | AJ18/4 | PCI33_BIDI |
| ad[43] | AG17/4 | PCI33_BIDI |
| ad[44] | AF17/4 | PCI33_BIDI |
| ad[45] | AH18/4 | PCI33_BIDI |
| ad[46] | AH17/4 | PCI33_BIDI |
| ad[47] | AK19/4 | PCI33_BIDI |
| ad[48] | AK18/4 | PCI33_BIDI |
| ad[49] | AG16/4 | PCI33_BIDI |
| ad[50] | AH16/4 | PCI33_BIDI |
| ad[51] | AF16/4 | PCI33_BIDI |
| ad[52] | AE16/4 | PCI33_BIDI |

*Table B-18. PCI Pin Assignments (Continued)*

| Signal Name | Pin/Bank | I/O Type |
|---|---|---|
| ad[53] | AJ17/4 | PCI33_BIDI |
| ad[54] | AJ16/4 | PCI33_BIDI |
| ad[55] | AK17/4 | PCI33_BIDI |
| ad[56] | AK16/4 | PCI33_BIDI |
| ad[57] | AK15/5 | PCI33_BIDI |
| ad[58] | AK14/5 | PCI33_BIDI |
| ad[59] | AJ15/5 | PCI33_BIDI |
| ad[60] | AJ14/5 | PCI33_BIDI |
| ad[61] | AK13/5 | PCI33_BIDI |
| ad[62] | AK12/5 | PCI33_BIDI |
| ad[63] | AE15/5 | PCI33_BIDI |
| cben[0] | AH10/5 | PCI33_IN |
| cben[1] | AH11/5 | PCI33_IN |
| cben[2] | AF13/5 | PCI33_IN |
| cben[3] | AE14/5 | PCI33_IN |
| cben[4] | AG15/5 | PCI33_IN |
| cben[5] | AH12/5 | PCI33_IN |
| cben[6] | AJ13/5 | PCI33_IN |
| cben[7] | AD15/5 | PCI33_IN |
| par | AG14/5 | PCI33_BIDI |
| par64 | AK10/5 | PCI33_BIDI |
| **PCI Interface Controls** | | |
| framen | AF15/5 | PCI33_IN |
| irdyn | AK7/5 | PCI33_IN |
| trdyn | AK6/5 | PCI33_OUT |
| stopn | AH13/5 | PCI33_OUT |
| idsel | AG13/5 | PCI33_IN |
| devseln | AK8/5 | PCI33_OUT |
| perrn | AK9/5 | PCI33_OUT |
| serrn | AH14/5 | PCI33_OUT |
| ack64n | AH15/5 | PCI33_OUT |
| req64n | K11/5 | PCI33_IN |
| **PCI Interrupts** | | |
| intan | AJ12/5 | PCI33_OUT |

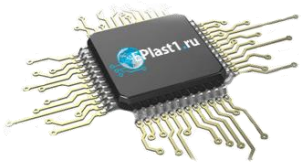
Компания «ЭлектроПласт» предлагает заключение долгосрочных отношений при поставках импортных электронных компонентов на взаимовыгодных условиях!

Наши преимущества:

- Оперативные поставки широкого спектра электронных компонентов отечественного и импортного производства напрямую от производителей и с крупнейших мировых складов;
- Поставка более 17-ти миллионов наименований электронных компонентов;
- Поставка сложных, дефицитных, либо снятых с производства позиций;
- Оперативные сроки поставки под заказ (от 5 рабочих дней);
- Экспресс доставка в любую точку России;
- Техническая поддержка проекта, помощь в подборе аналогов, поставка прототипов;
- Система менеджмента качества сертифицирована по Международному стандарту ISO 9001;
- Лицензия ФСБ на осуществление работ с использованием сведений, составляющих государственную тайну;
- Поставка специализированных компонентов (Xilinx, Altera, Analog Devices, Intersil, Interpoint, Microsemi, Aeroflex, Peregrine, Syfer, Eurofarad, Texas Instrument, Miteq, Cobham, E2V, MA-COM, Hittite, Mini-Circuits,General Dynamics и др.);

Помимо этого, одним из направлений компании «ЭлектроПласт» является направление «Источники питания». Мы предлагаем Вам помощь Конструкторского отдела:

- Подбор оптимального решения, техническое обоснование при выборе компонента;
- Подбор аналогов;
- Консультации по применению компонента;
- Поставка образцов и прототипов;
- Техническая поддержка проекта;
- Защита от снятия компонента с производства.




**Как с нами связаться**

**Телефон:** 8 (812) 309 58 32 (многоканальный)
**Факс:** 8 (812) 320-02-42
**Электронная почта:** org@eplast1.ru
**Адрес:** 198099, г. Санкт-Петербург, ул. Калинина, дом 2, корпус 4, литера А.