PCI Bus Applications

In Altera Devices

April 1995, ver. 1 Application Note 41

Introduction

The peripheral component interconnect (PCI) bus is designed for multiprocessor systems and high-performance peripherals, including audio and video systems, network adapters, graphics accelerator boards, and data storage controllers. PCI compliance requires a high I/O count, high drive requirements, relatively high density, and adherence to demanding timing specifications.

PCI-compliant designs are not required to implement all functions described in the ***PCI Local Bus Specification***; actual PCI designs typically use a subset of these specifications. Because each PCI design is unique, programmable logic devices provide an ideal solution for PCI designs.

Altera offers devices from the FLEX 8000, MAX 7000, and FLASHlogic device families that meet the I/O count, drive requirements, density, and performance specifications of the PCI interface.

This application note should be used with the following specification:

PCI Special Interest Group. ***PCI Local Bus Specification***. Rev. 2.0. Hillsboro, Oregon: PCI Special Interest Group, 1993.

Go to *Application Brief 140 (PCI Compliance of Altera Devices)* for a list of PCI-compliant Altera devices and their electrical specifications.

The PCI bus supports the following features:

- *Automatic configuration:* PCI hosts automatically identify and configure devices, an important feature in Plug and Play applications. Automatic configuration eliminates the need for the configuration switches commonly found on Industry Standard Architecture (ISA) cards.
- *Low power consumption:* The PCI interface supports both 5.0-V and 3.3-V operation for easier migration from 5.0-V to 3.3-V environments. The PCI interface also supports mixed-voltage environments.
- *High-performance data transfer:* PCI offers high-performance 32- or 64-bit data transfer and an optional burst mode that provides accelerated throughput of data across the bus. Devices connected to the PCI interface achieve performance comparable to that of devices directly connected to the processor local bus.

PCI Development Kit

Altera provides a PCI Development Kit to assist designers in implementing a custom PCI interface in Altera devices. The kit includes a diskette containing macrofunctions in Altera Hardware Description Language (AHDL) for FLEX 8000 and MAX 7000 devices and in PALASM2-compatible (**.pds**) files for FLASHlogic devices.

The PCI Development Kit contains two sets of macrofunctions:

- *Architecture-optimized macrofunctions:* These macrofunctions are optimized for each Altera device family; they meet the timing requirements of the ***PCI Local Bus Specification*** and demonstrate techniques that can be used when implementing the interface.
- *Customizable macrofunctions:* These macrofunctions contain generic master, target, and combined master/target functionality, but are not optimized for a specific device family. You can easily copy and customize these macrofunctions to suit your PCI application.

To obtain the PCI Development Kit, call Altera Literature Services at (408) 894-7144.

Masters, Targets & Back-End Devices

The master controls data transfers to and from the target. To initiate a data transfer, the master must assert the required signals and keep them asserted until they are acknowledged. To indicate completion, a target must provide an acknowledge cycle, which can include error or retry signals. The PCI interface automatically inserts any wait states needed to prevent the master from timing out, allowing high-speed PCI transactions to be processed by relatively slow back-end devices.

Table 1 lists the features of the master and target macrofunctions contained in the PCI Development Kit.

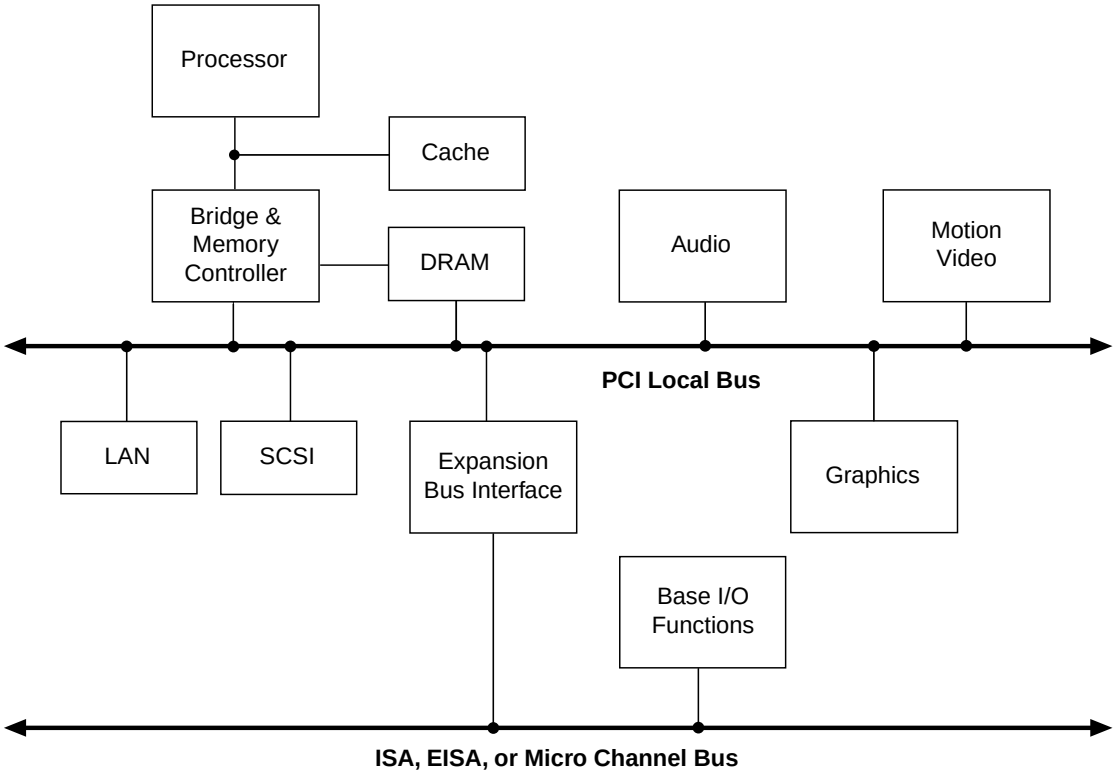
Table 1. Master & Target Macrofunction Features

Master	Target
Performs single-data-phase reads and writes.	Responds to single-data-phase reads and writes.
Can be customized for burst-mode data transfers.	Can be customized for burst-mode data transfers.
Addresses memory or I/O space.	Supports up to 32 bits of I/O address space.
Initiates configuration accesses.	Initiates configuration accesses.
Disconnects burst-mode configuration accesses.	Disconnects burst-mode configuration accesses and performs non-burst-mode I/O transactions.
Times out on master aborts. Recognizes target aborts.	Generates target aborts.
Recognizes retries.	Generates retries.
Generates and checks parity.	Generates and checks parity.
Responds to system resets (nRST).	Responds to system resets (nRST).
Generates parity errors (nPERR) and system errors (nSERR), and reports system errors (nSERR) to the device driver.	Generates parity errors (nPERR) and system errors (nSERR).

A PCI back-end device is defined as any device that stores, sends, or retrieves information using the PCI bus (for example, video cards, memory cards, disk drives, and multimedia cards). In a conventional configuration, the back-end device is considered a target. The target interface logic and memory can either be encapsulated in a controller device or fully or partially integrated into the back-end device. For example, enabling the target to access memory on the back-end device during configuration requests can make more free space available on the controller device.

The back-end device is physically remote from the processor local bus. Back-end devices communicate with the host processor across a PCI bridge (see Figure 1), which serves as a layer of device management between back-end devices and the host, thereby streamlining data transfer. The interface supports bus mastering, which allows intelligent devices to access main memory directly.

Figure 1. PCI System Block Diagram

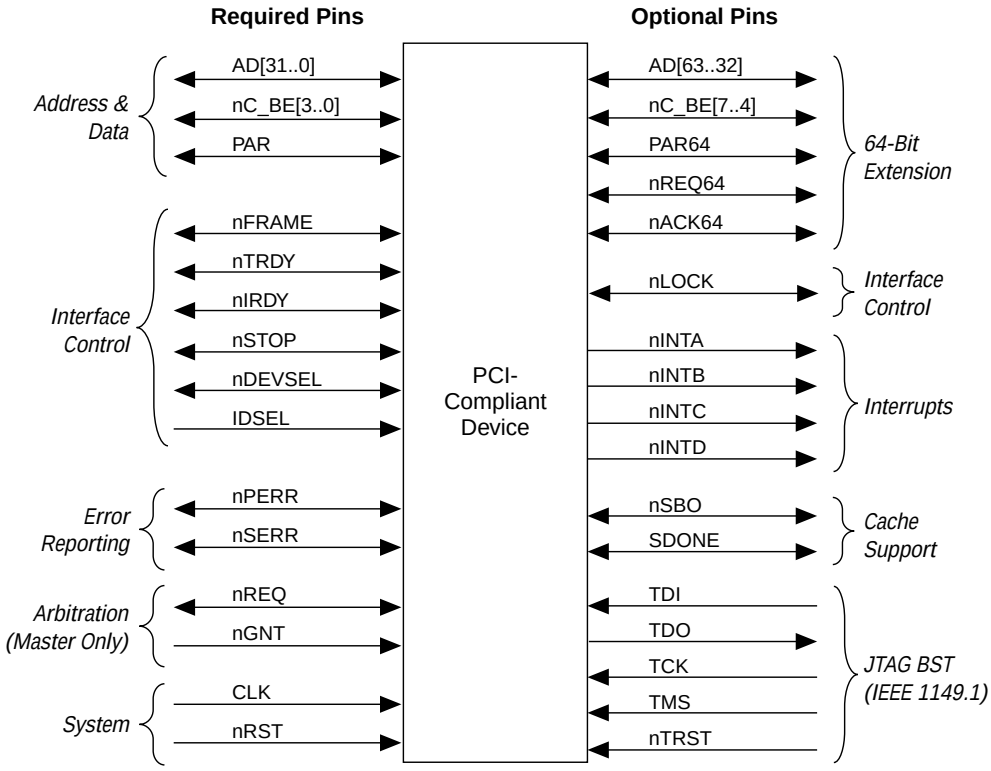


PCI Signals

All signals between the back-end device and the target are synchronized with the PCI Clock, which is typically based on the microprocessor and its support circuitry. The ***PCI Local Bus Specification*** defines an operating Clock rate ranging from 0 to 33 MHz, ensuring maximum flexibility for interfacing back-end devices of various speeds.

All 32-bit PCI buses have 124 pins; 64-bit PCI buses have 188 pins. A target must have at least 47 pins available for the bus interface. A master must have at least 49 pins available for the bus interface; two additional pins—request (`nREQ`) and grant (`nGNT`)—are required for master arbitration. The remaining PCI bus pins are used to implement 64-bit addressing and data transfer, interrupts, caching, and JTAG boundary-scan testing (BST), and supply power and ground connections. Figure 2 shows the functional grouping of required and optional signals defined by the ***PCI Local Bus Specification***.

Figure 2. Required & Optional Pins



The PCI bus uses the following types of pins:

- Inputs are input-only pins.
- Outputs are output-only pins.
- Bidirectionals can act as input pins or output pins.
- Tri-states are bidirectional, tri-state pins. Signals on these pins are in a high-impedance state when deasserted.
- Sustained tri-states are bidirectional tri-state pins. Signals on these pins are always active-low, and must be actively deasserted by driving them high for one cycle prior to being tri-stated. Pull-up resistors on the system board sustain the actively deasserted signal.
- Open-drains allow several devices to share a signal as a wired-OR.

Table 2 lists each signal pin and its signal type.

Table 2. PCI Signal Types

Signal Name	Signal Type
CLK	Input
nRST	Input
AD[31..0]	Tri-state
nC_BE[3..0]	Tri-state
PAR	Tri-state
nDEVSEL	Sustained tri-state
nFRAME	Sustained tri-state
IDSEL	Input
nIRDY	Sustained tri-state
nLOCK	Sustained tri-state
nSTOP	Sustained tri-state
nTRDY	Sustained tri-state
nGNT	Tri-state
nREQ	Tri-state
nPERR	Sustained tri-state
nSERR	Open-drain
nINTA	Open-drain
nINTB	Open-drain
nINTC	Open-drain
nINTD	Open-drain
nSBO	Bidirectional
SDONE	Bidirectional
nACK64	Sustained tri-state
AD[63..32]	Tri-state
nC_BE[7..4]	Tri-state
nPAR64	Tri-state
nREQ64	Sustained tri-state
TCK	Input
TDI	Input
TDO	Output
TMS	Input
nTRST	Input

Table 3 describes the PCI pins.

Table 3. PCI Pins (Part 1 of 2)

Pin Type	Pin Name *(1)*	Master	Target	Description
System pins	CLK	Input	Input	Provides timing for all transactions, including bus arbitration, at frequencies ranging from 0 to 33 MHz.
	nRST	Input	Input	When asserted, initializes all PCI configuration registers, masters, and targets.
Address/ data pins	AD[31..0]	Bidirectional	Bidirectional	Driven by the master during a write transaction or the selected target during a read transaction.
	nC_BE[3..0]	Bidirectional	Input	Issues a command during the address phase and a byte enable during the data phase.
	PAR	Bidirectional	Bidirectional	Ensures even parity across AD[31..0] and nC_BE[3..0].
Interface control pins	nDEVSEL	Input	Output	Asserted by the target when it decodes its address. If not active within 6 CLK cycles after a transfer is initiated, the initiating master aborts the transfer.
	nFRAME	Bidirectional	Input	Driven by the current bus master. Indicates the start and duration of a transaction. The nFRAME signal is deasserted when the master is ready to complete the final data phase in the transaction.
	IDSEL	Input	Input	Used as a chip select during access to the configuration registers.
	nIRDY	Bidirectional	Input	During a write transaction, the current bus master asserts this signal to indicate that valid data is being driven onto the PCI bus. During a read transaction, the current bus master asserts this signal to indicate that the master is ready to accept data from the selected target. During read and write transactions, wait states are inserted until both nIRDY and nTRDY are asserted.
	nLOCK	Bidirectional	Input	When asserted, prevents any other master from accessing the bus.
	nSTOP	Bidirectional	Output	Asserted by target to stop a master transaction.
	nTRDY	Bidirectional	Output	During a read transaction, the target asserts signal to indicate that the target is driving valid data onto the PCI bus. During a write transaction, the target asserts this signal to indicate that the target is ready to accept data. During read and write transactions, wait states are inserted until both nIRDY and nTRDY are asserted.

Table 3. PCI Pins (Part 2 of 2)

Pin Type	Pin Name *(1)*	Master	Target	Description
Arbitration pins (bus masters only)	nGNT	Input	–	Originates from the arbiter. Tells the master that it is granted bus control.
	nREQ	Output	–	Tells the arbiter that a particular master needs dedicated access to the PCI bus.
Error reporting pins	nPERR	Bidirectional	Output	Asserted by the master or target if a parity error is detected on the address or data. Depending on the design, this assertion may cause an abort.
	nSERR	Output	Output	Reports address parity errors and special cycle data parity errors. The master or target asserts this signal only if allowed by the configuration command. This open-drain signal can be generated simultaneously by multiple devices.
Interrupt pins	nINTA	Output	Output	Open-drain signal to the system interrupt controller.
	nINTB	Output	Output	Open-drain signal to the system interrupt controller.
	nINTC	Output	Output	Open-drain signal to the system interrupt controller.
	nINTD	Output	Output	Open-drain signal to the system interrupt controller.
Cache support pins	nSBO	–	Input	Indicates that the PCI memory access in progress is about to read or update the information in memory. This signal must be qualified by the SDONE signal.
	SDONE	–	Input	Asserted by the cache when it finishes snooping a PCI memory access.
64-Bit extension pins *Note (2)*	nACK64	Input	Output	Generated by the selected target in response to the nREQ64 signal from the master.
	AD[63..32]	Bidirectional	Bidirectional	Contains the upper 4 data bytes for a 64-bit bus. These data bytes are used in the addressing phase only if 64-bit addressing is being used.
	nC_BE[7..4]	Output	Input	Additional command and byte enable signals.
	PAR64	Bidirectional	Bidirectional	Even parity bit associated with AD[63..32] and nBE[7..4].
	nREQ64	Bidirectional	Input	Generated by the master to indicate a 64-bit transfer.
JTAG pins *Note (2)*	TCK	Input	Input	Synchronous Clock for JTAG operation.
	TDI	Input	Input	Input pin for all JTAG shift registers, such as instruction registers and peripheral registers.
	TDO	Output	Output	Output pin for all bits being scanned out of the JTAG registers.
	TMS	Input	Input	Controls the state of the Test Access Port (TAP) state machine.
	nTRST	Input	Input	Asynchronous reset to force the TAP into an initialized state.

Notes to table:
(1) An "n" preceding the signal name (e.g., nRST) indicates an active-low signal.
(2) These pins are optional.

Back-End I/O Signals

Table 4 lists describes the back-end signals.

Table 4. Back-End I/O Signals

Signal Name	Signal Function	Signal Description
ADDR[31..0]	Address bus	Carries address information.
APAR	Address parity	Drives even parity from the back-end device onto ADDR[31..0] when the address information is valid.
nBE[3..0]	Byte enable	Identifies bytes within a double word being accessed.
CNFG	Configuration access	When high, indicates a configuration access. When low, indicates an I/O access to the target, or tells the back-end device to stop driving the bus and to prepare for a configuration access to the master.
DATA[31..0]	Data bus	Carries data.
nDEV_REQ	Device request	Tells the target that the back-end device has completed the data transfer to the target, or tells the master that the back-end device has data to transfer to the master.
nDEV_ACK	Device acknowledge	Tells the target that the master has a data request for the back-end device, or tells the back-end device that the data transfer to the master is complete.
DPAR	Data parity	Drives even parity from the back-end device onto DATA[31..0] when the data is valid.
nM_ABORT	Master abort	When asserted, the master tells the back-end device that no target responded to its request.
MEM_IO	Memory I/O	When high, indicates a memory transfer. When low, indicates an I/O transfer.
nRETRY	Retry	Tells the master to try again later because the target is not able to respond at this time.
nRD_WR	Read/write	When high, indicates a read transfer. When low, indicates a write transfer.
nT_ABORT	Target abort	Indicates a fatal error in the target.

Configuration Registers

Upon system power-up, the host identifies and configures each back-end device. To facilitate this process, all back-end devices must provide read/write register space that contains configuration information, as shown in Table 5.

Table 5. Configuration Header Registers

Bytes	Bits			
	31	16	15	0
00h	Device ID		Vendor ID	
04h	Status		Command	
08h	Class Code			Revision ID
0Ch	Built-in Self-Test (BIST)	Header Type	Latency Timer	Cache Line Size
10h	Base Address Register (BAR)			
14h				
18h				
1Ch				
20h				
24h				
28h	Reserved			
2Ch	Reserved			
30h	Expansion ROM Base Address			
34h	Reserved			
38h	Reserved			
3Ch	MAX_LAT	MIN_GNT	Interrupt Pin	Interrupt Line

Table 6 describes the required configuration registers. Both required and optional registers are listed in the ***PCI Local Bus Specification***.

Table 6. Required PCI Configuration Registers

Register	Description
Vendor ID	Indicates the manufacturer of the device. This identifier is allocated to the manufacturer by the PCI Special Interest Group (SIG). The vendor ID for Altera devices is 1172h.
Device ID	Identifies the device. This identifier is allocated to the manufacturer by the PCI SIG.
Command	Determines the device's response to different commands. Configuration software must set the appropriate bits.
Status	Records PCI bus events and errors; also contains target device capability information.
Revision ID	Specifies a device-specific revision. The value is chosen by the vendor.
Class code	Used to identify the generic function of the device. This register is read-only.
Header type	Identifies the layout of bytes 10h through 3Fh and indicates whether the device contains multiple functions.
Base address register (BAR)	Tells the target which addresses are within the address space of the back-end device. This register must be located in the target, rather than in the back-end device, so that the target can perform address decoding.

PCI Controller

Starting with a predefined, generic interface design, you can easily modify the design template provided in the PCI Development Kit to meet your needs. The ***PCI Local Bus Specification*** recommends using the generic PCI controller architecture shown in Figure 3. The generic architecture contains interface logic, controller logic, configuration register space, and optional expansion ROM.

Go to the ***PCI Local Bus Specification*** for information on the expansion ROM.

Figure 3. Generic PCI Controller Architecture

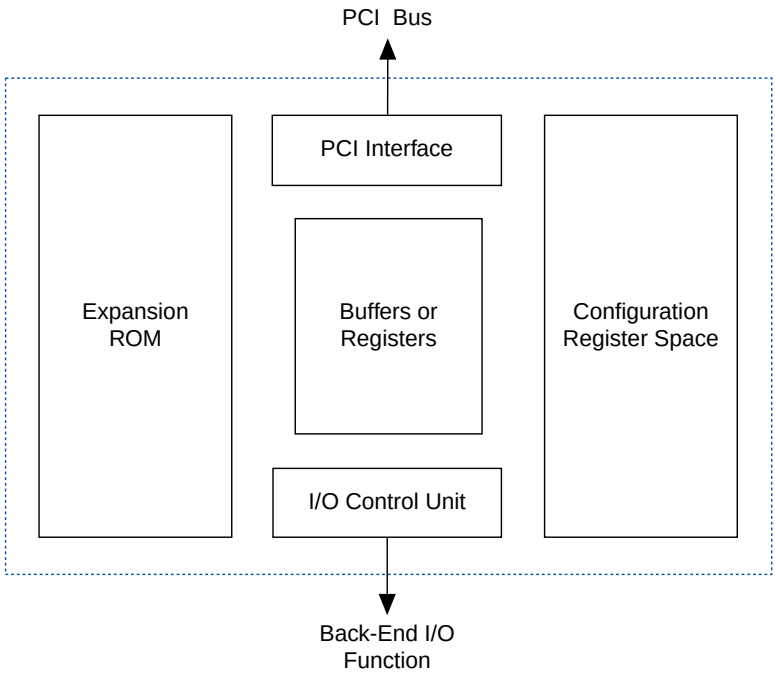
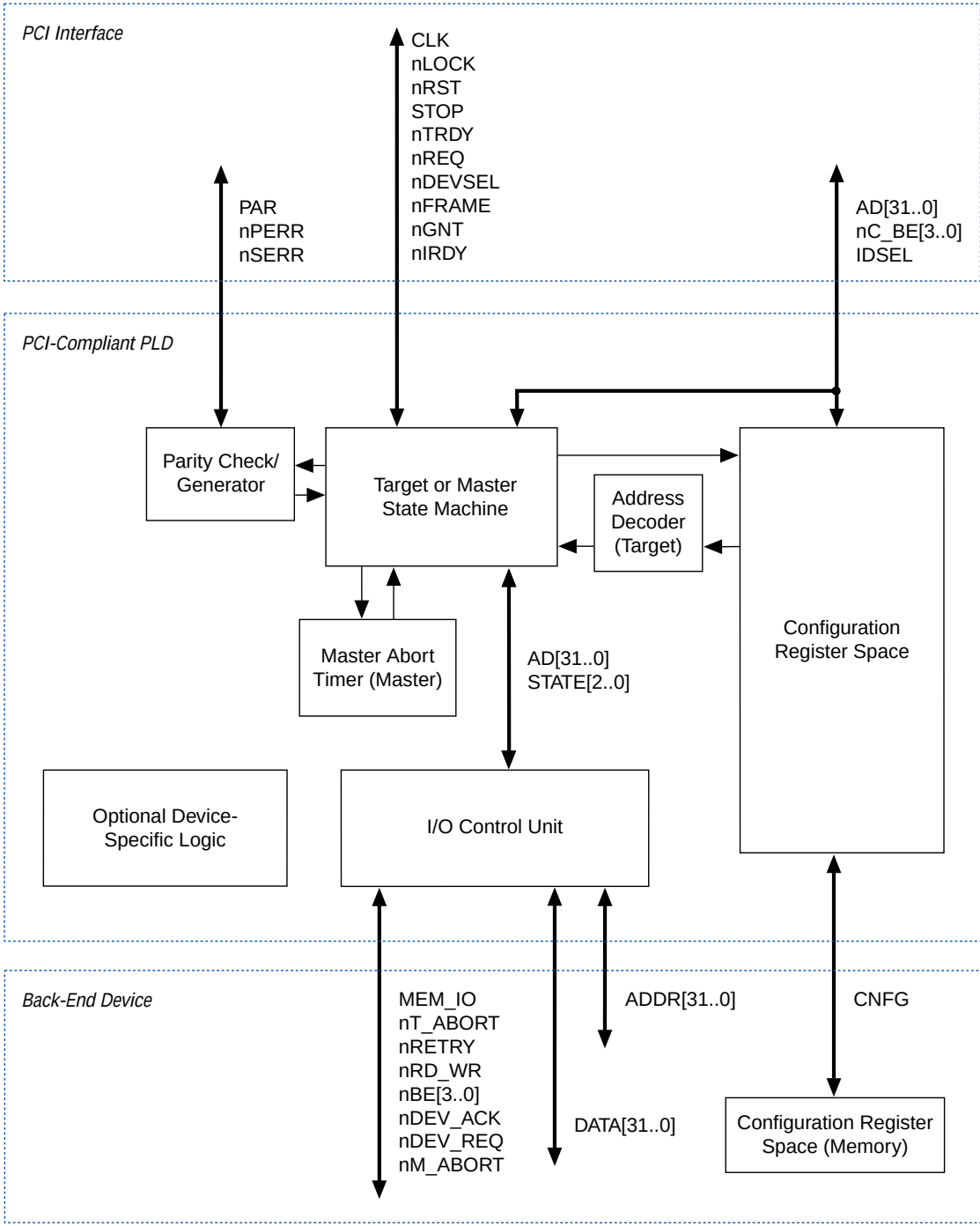


Figure 4 shows how the PCI controller is interconnected with the components of the back-end device.

Figure 4. Master/Target Architecture



Controller State Machines

The master and target state machines are shown in Figure 5 and 6. These figures are similar to those found in the ***PCI Local Bus Specification***; however, they have been modified to support the generic interface described in this application note.

Go to the PCI Development Kit for the AHDL Text Design Files (**.tdf**) and the PALASM2-compatible files that implement these state machines.

Figure 5. Master State Machine

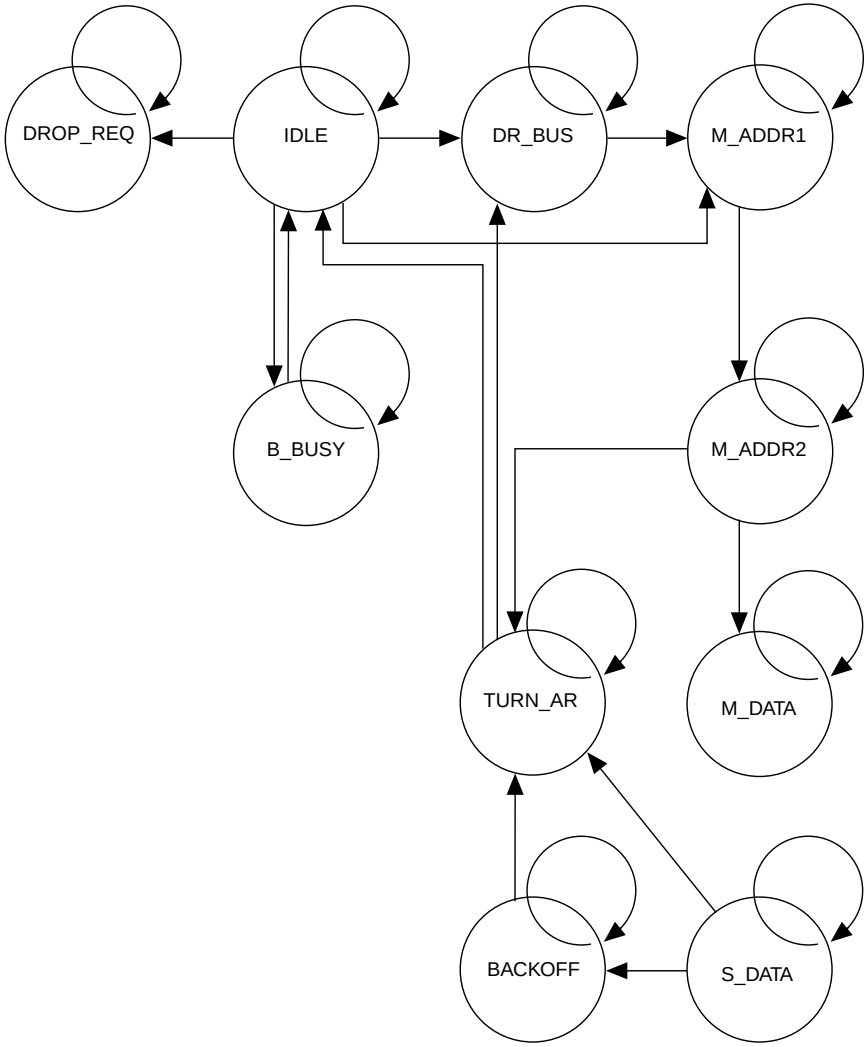
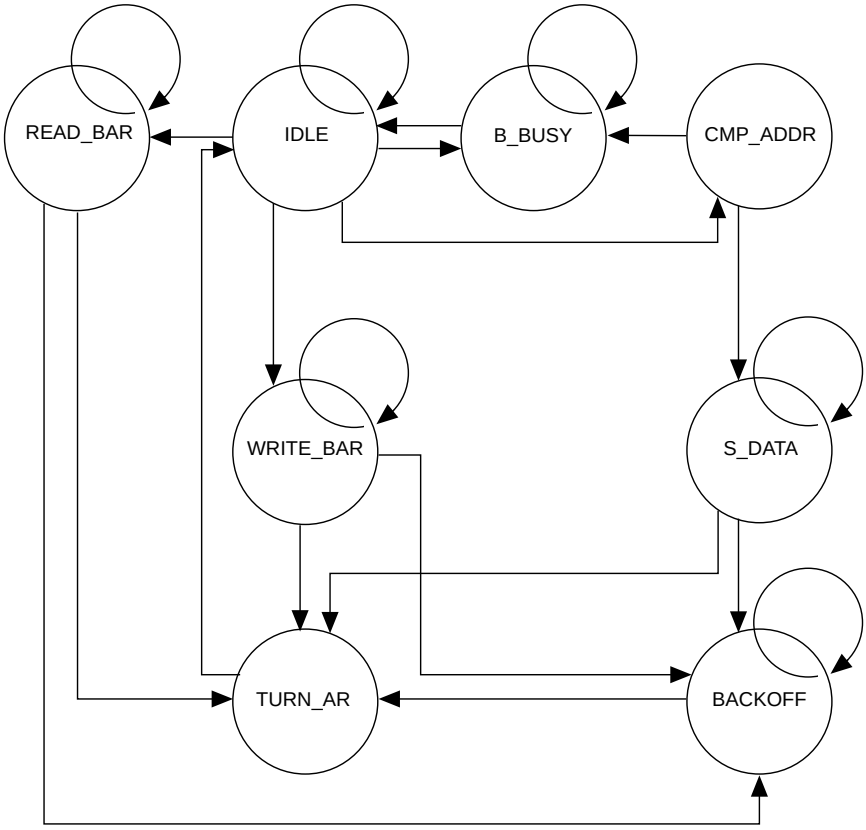


Figure 6. Target State Machine



PCI Bus States

The PCI bus states for masters and targets are described in Table 7.

Table 7. PCI Bus States (Part 1 of 2)

State	Master	Target
IDLE	The master latches signals from the bus to determine whether another master is attempting a configuration access, and checks for a data transfer request from the back-end device. This state is the default.	The target latches signals from the bus to determine whether a master is attempting either a configuration access or an I/O data transfer. This state is the default.
M_ADDR1	The master deasserts nFRAME at the start of the next CLK pulse. This state is used to prepare for the address phase of the transaction.	–
M_ADDR2	The master drives the address phase of the transaction onto the PCI bus. Then, nFRAME is asserted, AD is driven with the address, and nC_BE is driven with the PCI command code.	–
CMP_ADDR	–	When a bus transaction begins, the address on the bus is compared to the address space of the back-end device. If the address matches, the interface goes to the S_DATA state to start a transfer. Otherwise, it goes to the B_BUSY state.
M_DATA	The master drives the data phase of the transfer: the master drives the address parity after the address cycle and data parity after each succeeding data cycle (for write operations only) onto the PCI bus.	–
S_DATA	The master performs configuration access data transfers. When the transaction is complete, the bus goes either to the TURN_AR state if the master did not attempt a burst-mode transfer or to the BACKOFF state if the master did attempt a burst-mode transfer.	The target handles all data transfers. When data transfers are complete, the interface goes to either the TURN_AR state for error-free, non-burst-mode transfers or the BACKOFF state if the target detects an error or burst-mode transfer attempt.
BACKOFF	After the master disconnects a burst-mode configuration access, it waits in this state until nFRAME is deasserted and then goes to the TURN_AR state.	The target disconnects a burst-mode transfer attempt from the master, or signals a target abort or retry. The target waits in this state until the master deasserts the nFRAME signal, then goes to the TURN_AR state.
TURN_AR	The master performs the following functions: releases the PCI bus and actively deasserts PCI signals when the data transfer is complete; checks or generates parity for the final data phase; and reports errors to the back-end device if the target signals a retry or a target abort.	The target releases the bus and actively deasserts bus signals. From this state, the target automatically returns to the IDLE state.

Table 7. PCI Bus States (Part 2 of 2)

State	Master	Target
DR_BUS	Rather than allowing the devices to tri-state signals during periods of inactivity and waste energy, the arbiter parks the bus on an arbitrary master, causing the master to drive meaningless data onto the bus. This state is similar to the IDLE state except that, in the IDLE state, the master does not drive any data onto the bus.	–
B_BUSY	The master cannot detect a configuration access in the middle of a transaction. The IDSEL signal is decoded from the AD signals; therefore, it can be asserted during the data phases of a transaction even if it is not a configuration access. The target only goes to the DROP_REQ state if it detects a configuration access on the first CLK pulse in which nFRAME is asserted (the address phase) but it is not the target of a configuration access. The master returns to the IDLE state when nFRAME is deasserted. In the meantime, the master continues to request the bus while nDEV_REQ is asserted by the back-end device.	The target monitors bus transactions that do not involve the back-end device. As long as another transaction is taking place (nFRAME is asserted), the interface remains in this state. When the other transaction is complete (nFRAME is deasserted), the target returns to the IDLE state to wait for the address phase of the next transaction.
DROP_REQ	The back-end device stops driving all request signals and prepares for a configuration access.	–
READ_BAR	–	The target reads from the base address register (BAR), which is one of the configuration registers.
WRITE_BAR	–	The target writes to the BAR, which is one of the configuration registers.

Parity Generation & Checking

All masters and targets must generate and check for even parity and report parity errors on all data and addresses received on the PCI bus. The target generates parity for the base address register (BAR) when the host bridge reads it. To check parity, devices perform the following functions:

1. The device generates parity internally by latching the values on AD[31..0] and nC_BE[3..0] and XORing all 36 bits.
2. On the next CLK pulse, the device XORs its internal calculation with the value driven on PAR by the device that generated parity.
3. If these values match, no parity error exists. Otherwise, on the next CLK pulse, the device signals a parity error by asserting either nPERR for a data error or nSERR for an address error.

The master design in the PCI Development Kit shows how to support either on-chip or off-chip parity generation and provide parity checking for addresses and data in the back-end device.

According to the ***PCI Local Bus Specification***, parity generation and checking is not required for devices that do not store data or address information, for example, devices designed for use on a motherboard or in real-time transmissions, such as video applications.

PCI Bus Commands

Table 8 lists all PCI bus commands and their corresponding binary codes.

Table 8. PCI Bus Commands

Binary Code	Type of Command
0000	Interrupt acknowledge
0001	Special cycle
0010	I/O read
0011	I/O write
0100	Reserved
0101	Reserved
0110	Memory read
0111	Memory write
1000	Reserved
1001	Reserved
1010	Configuration read
1011	Configuration write
1100	Memory read multiple
1101	Dual address cycle
1110	Memory read line
1111	Memory write and invalidate

Go to the ***PCI Local Bus Specification*** for a complete description of these commands.

Burst-Mode Data Transfers

The PCI specification supports burst-mode data transfer, which consists of an address phase followed by one or more data phases. The burst-mode transfer is the same regardless of whether the data is transferred to a back-end device or to configuration registers. Burst-mode data transfer takes place at a very high rate because multiple data phases are allowed per address phase, instead of one data phase per address phase without burst-mode data transfer. The macrofunctions in the PCI Development Kit are designed to accommodate burst-mode implementations.

PCI Transactions

The following examples illustrate the operation of PCI transactions.

Target Read Transaction

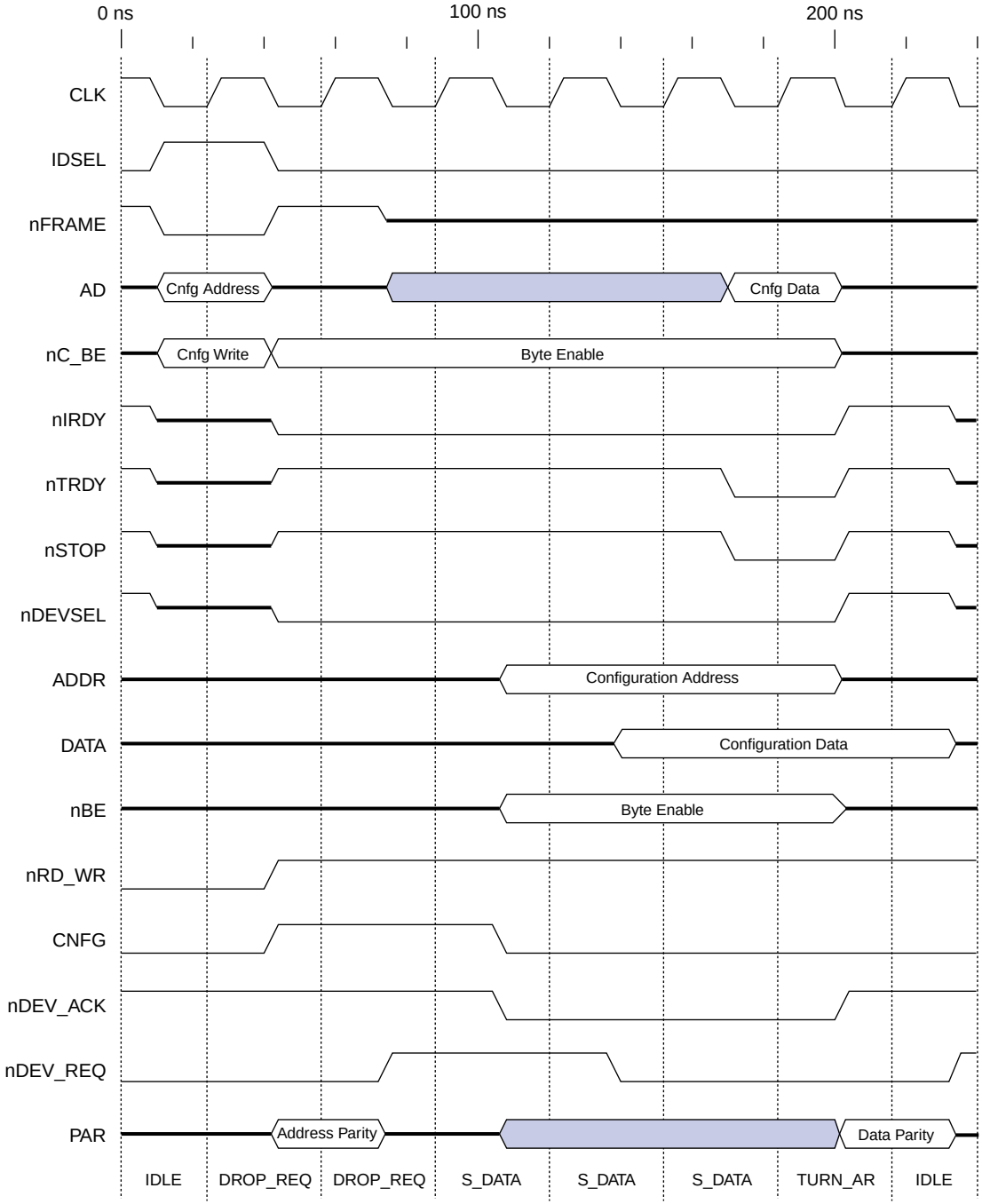
The target performs a read transaction without burst-mode transfers when it is in the IDLE state, detects that nFRAME is asserted, and receives an I/O Read command on nC_BE. In response to these conditions, the target performs the following functions:

1. The bus goes to the CMP_ADDR state and the target asserts nRD_WR, indicating a read transaction.
2. The target compares the address to the contents of the BAR. If the addresses are the same, HIT is asserted. When the target detects that HIT is asserted, then the target acknowledges that it is selected and goes to the S_DATA state. Otherwise, the target goes to the B_BUSY state and waits for nFRAME to be deasserted before returning to the IDLE state.
3. When the target detects that nDEV_ACK is asserted, it acknowledges that there is an access pending and that ADDR and nBE are valid.
4. When the back-end device asserts nDEV_REQ, the target asserts nTRDY and nSTOP, which tells the master that the data on AD is valid, the target is disconnecting the transfer, and the target will not complete the burst-mode transfer request. The target remains in the S_DATA state until the master asserts nIRDY and nTRDY, indicating that the master received the data.
5. The master desasserts nDEV_ACK, informing the target that the transfer is complete. The bus goes to the BACKOFF state.
6. The master deasserts nFRAME and the bus goes to the TURN_AR state. In this state, the target actively deasserts nTRDY, nSTOP, and nDEVSEL.
7. Finally, the target returns to the IDLE state.

Figure 7 illustrates a target read transaction.

Figure 7. Target Read Transaction

Bold lines represent tri-stated signals. Shaded areas represent "don't care" signals.



Target Configuration Write Transaction

The target performs a configuration read transaction when it is in the IDLE state, detects that nFRAME and IDSEL are asserted, and receives a configuration write command on nC_BE. In response to these conditions, the target performs the following functions:

1. The target goes to the S_DATA state, asserts nRD_WR to indicate a write transaction, and asserts CNFG to indicate a configuration access.

2. The target asserts nDEV_ACK to tell the back-end device that an access operation is pending and that ADDR, DATA, and nBE are valid.

3. With nDEV_ACK asserted, the target waits until the back-end device asserts nDEV_REQ, indicating that the back-end device has completed the write transaction without error.

4. When the target detects that nDEV_REQ is asserted, it asserts nTRDY and nSTOP, indicating that the write is complete, that it is disconnecting the transfer, and that the burst-mode transfer request will not be completed.

5. The master deasserts nFRAME, indicating that the transaction was not a burst-mode transaction. Because nIRDY is asserted, the target goes to the TURN_AR state, indicating that the target has completed the transaction.

6. The target desasserts nDEV_ACK to tell the back-end device that the transfer has completed and asserts nTRDY, nSTOP, and nDEVSEL before tri-stating them.

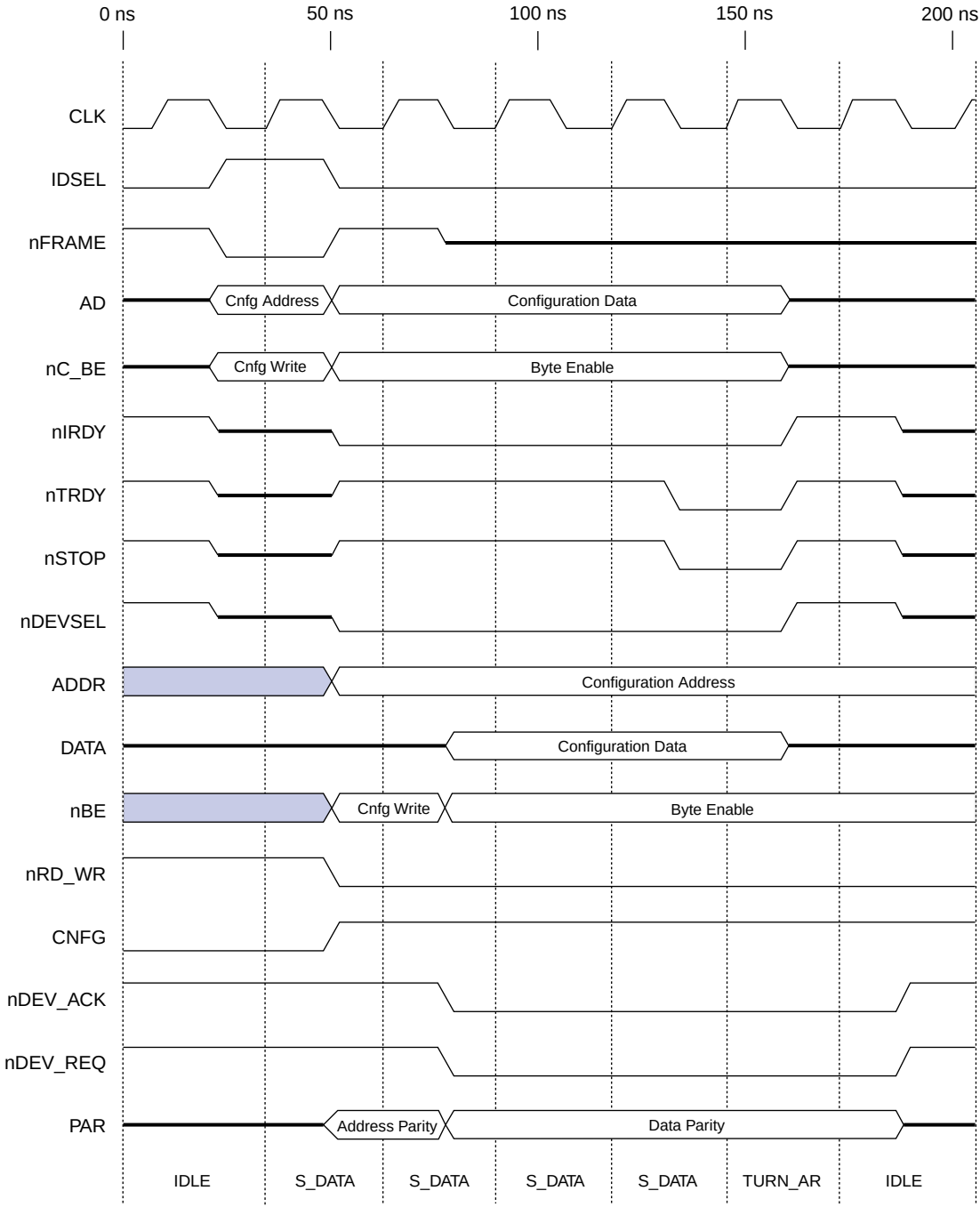
7. The target returns to the IDLE state.

8. The target checks PAR for a parity error on the transfer. If there is a parity error, the target asserts PAR after the next CLK pulse.

Figure 8 illustrates a target configuration write transaction.

Figure 8. Target Configuration Write Transaction

Bold lines represent tri-stated signals. Shaded areas represent "don't care" signals.



Master Write Transaction

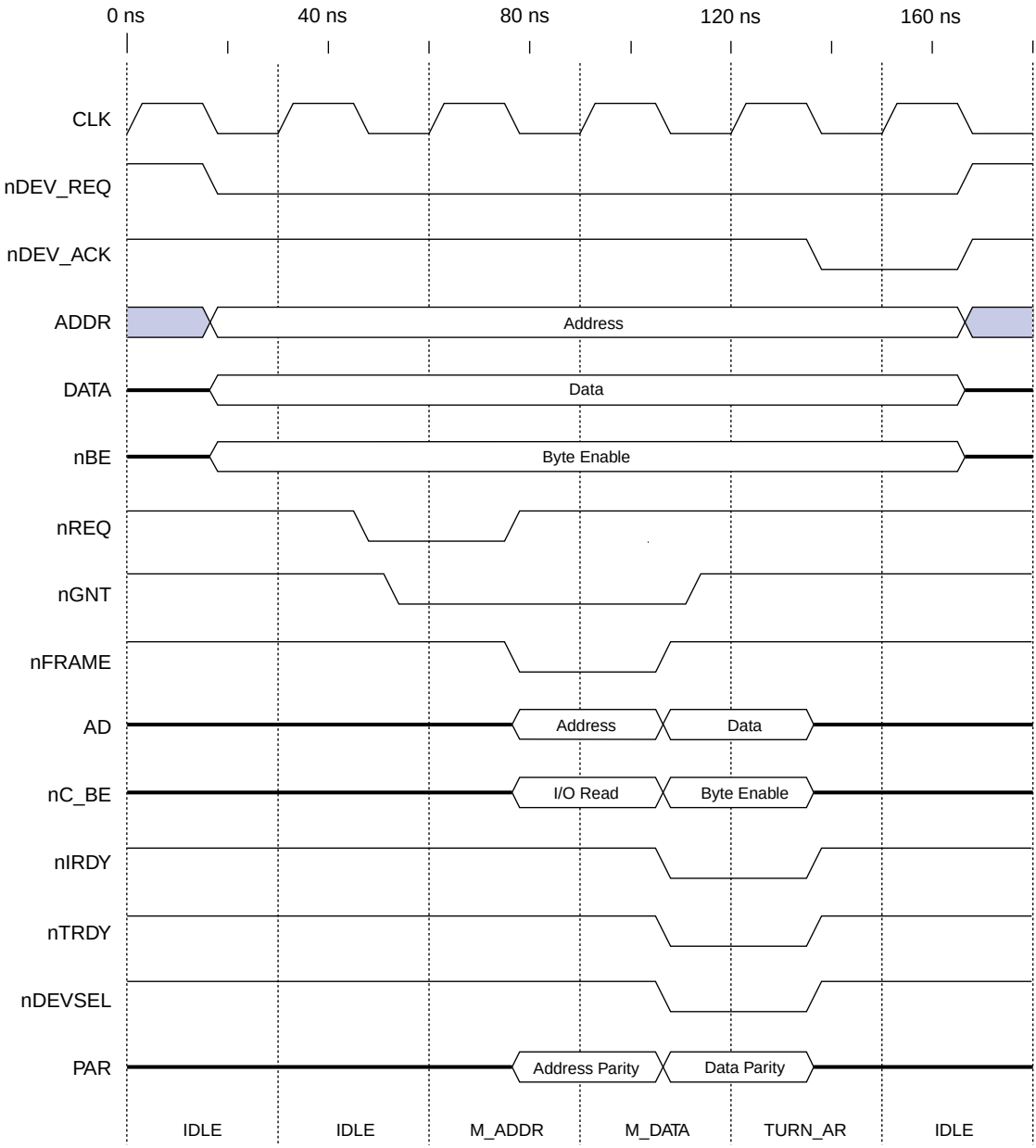
The master performs a write transaction when it is in the IDLE state and detects that nDEV_REQ and nGNT are asserted, and that nIRDY is deasserted. In response to these conditions, the master performs the following functions:

1. The master goes to the M_ADDR1 state and deasserts nFRAME.
2. The master goes to the M_ADDR2 state and drives the address phase onto the bus.
3. The master asserts nFRAME, drives the address from ADDR onto AD, and drives the appropriate command onto nC_BE.
4. The master goes to the M_DATA state.
5. The master drives data from DATA onto AD and byte enables from nBE onto nC_BE.
6. The master deasserts nFRAME to indicate a non-burst-mode transfer and nIRDY to indicate that the write data on AD is valid.
7. The master generates address parity.
8. Because nDEVSEL and nTRDY are both asserted by the target, indicating that it latched the write data, the master goes to the TURN_AR state.
9. The master asserts nDEV_ACK to tell the back-end device that the transfer is complete.
10. The master deasserts nIRDY to tell the next bus master that it can begin a bus cycle on the next CLK pulse.
11. The master generates data parity and returns to the IDLE state.
12. When the back-end device detects that nDEV_ACK is asserted, it deasserts nDEV_REQ and the rest of its request signals.

Figure 9 illustrates a master write transaction.

Figure 9. Master Write Transaction

Bold lines represent tri-stated signals. Shaded areas represent "don't care" signals.



Master Configuration Write Transaction

When the bus is in the IDLE state, the master goes to the DROP_REQ state and begins a configuration write transaction if it detects one of the following conditions:

- nFRAME is asserted
- IDSEL is asserted
- nC_BE[3..0] = 1011
- AD[1..0] = 00

In response to these conditions, the master performs the following functions:

1. In the DROP_REQ state, the master asserts CNFG, which tells the target to prepare for a configuration write transaction. The target stops driving all of its request lines to the interface. The master remains in the DROP_REQ state until it detects that nDEV_REQ is deasserted.

2. The target asserts nDEVSEL to tell the master that the target is claiming the transaction.

3. When the target stops driving its request lines, the interface goes to the S_DATA state. In this state, the configuration address is driven onto ADDR, the configuration data is driven from AD onto DATA, and the byte enables are driven from nC_BE onto nBE.

4. The master asserts nDEV_ACK to tell the target that ADDR, DATA, and nBE are valid, and to start the transaction.

5. When the back-end device completes the transaction, it asserts nDEV_REQ. When nDEV_REQ is asserted, the target asserts nTRDY to tell the master that the transfer is complete and nSTOP to disconnect any possible burst-mode transfer attempt.

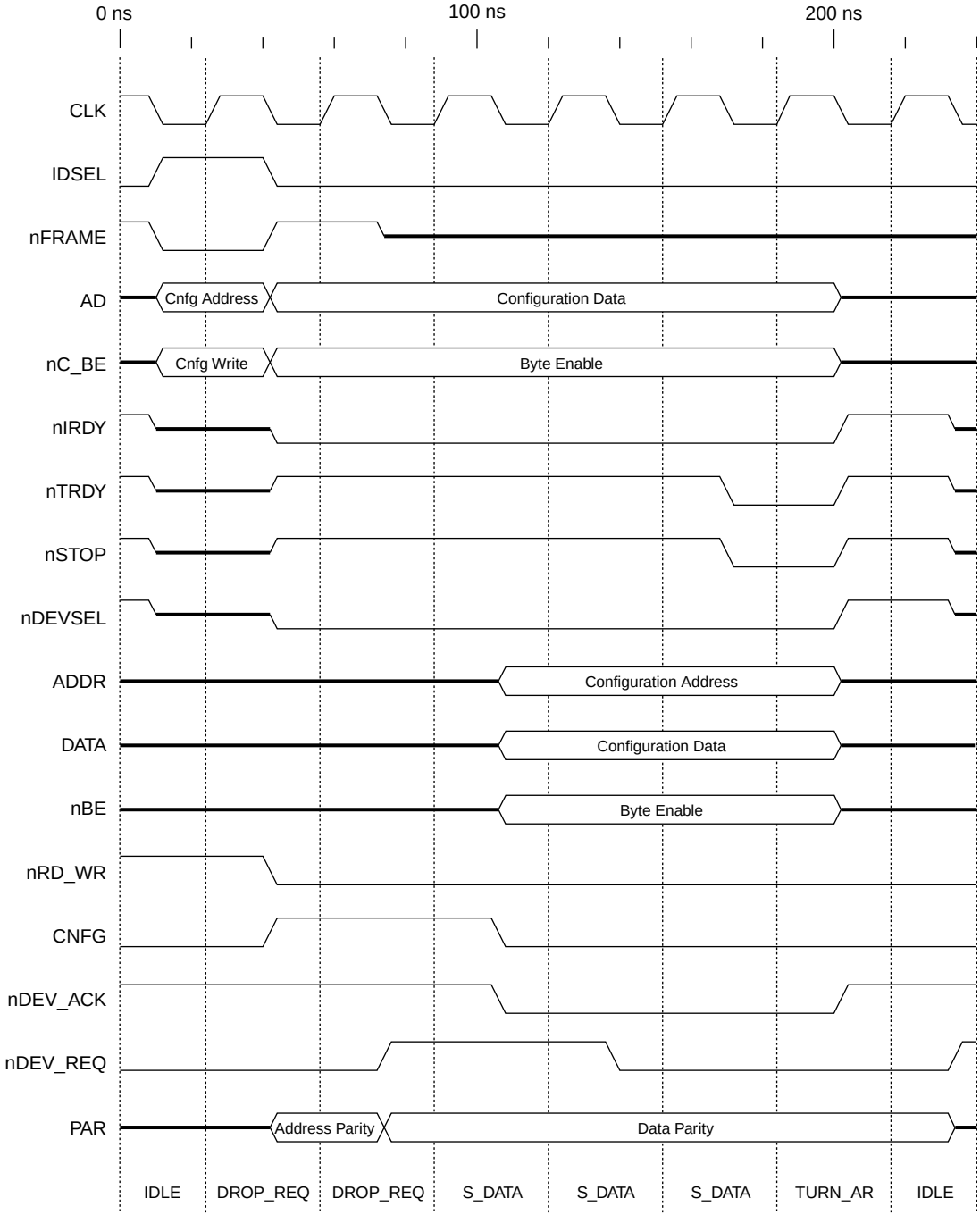
6. Because there was no burst-mode transfer attempt (nFRAME is deasserted), the target goes to the TURN_AR state and actively deasserts nTRDY, nSTOP, and nDEVSEL.

7. Finally, the bus returns to the IDLE state.

Figure 10 illustrates a master configuration write transaction.

Figure 10. Master Configuration Write Transaction

Bold lines represent tri-stated signals. Shaded areas represent "don't care" signals.



Master Configuration Read Transaction

When the bus is in the IDLE state, the master goes to the DROP_REQ state and begins a configuration read transaction if it detects one of the following conditions:

- nFRAME is asserted
- IDSEL is asserted
- nC_BE[3..0] = 1010
- AD[1..0] = 00

In response to these conditions, the master performs the following functions:

1. In the DROP_REQ state, the master asserts CNFG, which tells the target to prepare for a configuration read transaction. The target stops driving its request lines to the interface. The master remains in this state until nDEV_REQ is deasserted.

2. The target asserts nDEVSEL to tell the master that it is claiming the transaction.

3. When the target stops driving its request lines, the interface goes to the S_DATA state. Then, the target drives the configuration address onto AD and the byte enables from nC_BE onto nBE.

4. The master asserts nDEV_ACK to tell the target that the address and byte enables are valid, and that it must return the contents of the specified registers.

5. When the back-end device has completed the transaction, it asserts nDEV_REQ. Then, the target drives the contents of DATA[31..0] onto AD[31..0], asserts nTRDY to tell the master that the transfer is complete, and asserts nSTOP to disconnect any possible burst-mode transfer attempt.

6. Because there was no burst-mode transfer attempt, the master goes to the TURN_AR state.

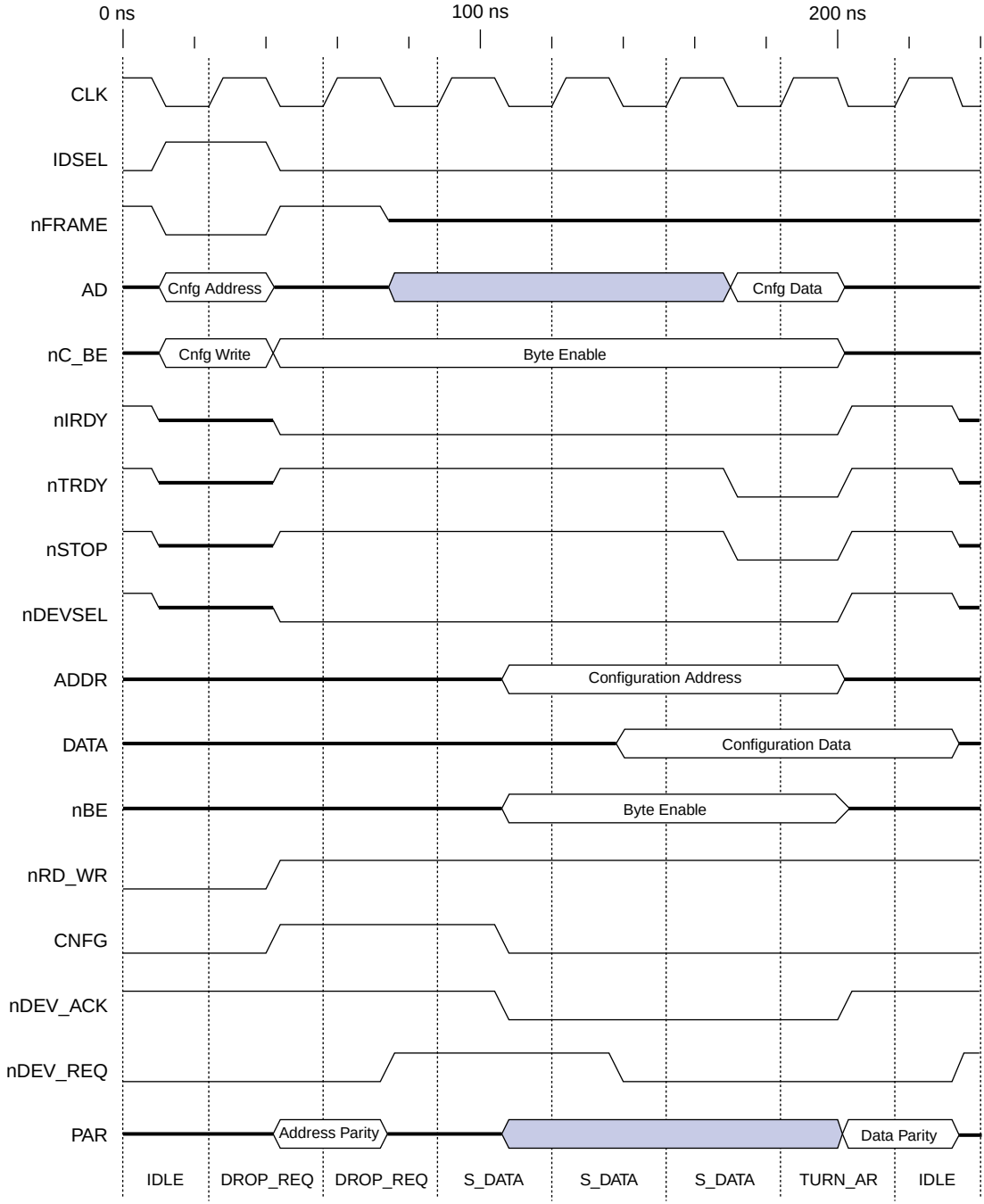
7. The target actively deasserts nTRDY, nSTOP, and nDEVSEL.

8. Finally, the bus returns to the IDLE state.

Figure 11 illustrates a master configuration read transaction.

Figure 11. Master Configuration Read Transaction

Bold lines represent tri-stated signals. Shaded areas represent "don't care" signals.



Conclusion

PCI is an intelligent, high-performance interface that enables system developers to integrate a variety of functions. Altera's FLEX 8000, MAX 7000, and FLASHlogic devices provide a programmable logic solution for the increasing number of PCI bus applications. These devices comply with PCI voltage, current, and timing specifications. In addition, their programmability provides flexible solutions that enable implementation of increasingly specialized peripheral functionality as system requirements change.

Altera's PCI Development Kit can be used to implement a custom PCI interface in Altera devices. The kit includes a diskette containing macrofunctions in AHDL and PALASM2-compatible files that are optimized to meet PCI timing specifications in Altera devices. Template design files containing generic master and target functionality are also provided; these files can be customized to suit any application.

Altera provides a comprehensive PCI hardware and software solution that enables designers to implement customized, PCI-compliant master and target interfaces with Altera devices.

2610 Orchard Parkway
San Jose, CA 95134-2020
(408) 894-7000
Applications Hotline:
(800) 800-EPLD
Customer Marketing:
(408) 894-7104
Literature Services:
(408) 894-7144