Designing a 33MHz, 32-Bit PCI Target Using Lattice Devices

August 2013 Reference Design RD1008

Introduction

The evolution of digital systems over the past two decades has placed new requirements on system designers. They now need to design interfaces that are both high performance and compatible with other vendors' systems. At the same, time they need to meet immense time-to-market demands. The compatibility issue has been resolved by designing systems with bus interfaces that are standards in the industry such as ISA, EISA, VESA and Micro Channel.

As performance became an increasingly important factor, a new interface standard called PCI (Peripheral Component Interconnect) was developed to meet the requirements of today's digital computer systems. PCI is a well-documented standard supported by a special interest group and features the performance of a 33MHz, 32-bit version of the specification reaching 132Mbytes per second at its peak transfer rate. This document describes a reference design solution for a 33MHz, 32-bit PCI target for LatticeECP3™, LatticeXP2™, MachXO™ and ispMACH® devices. It is designed to provide users with a starting point for designing a PCI target into Lattice devices.

The reference design source code is available from Lattice upon the signing of a simple non-disclosure agreement. The 33MHz, 32-bit PCI target reference design comes with a fully-automated HDL test environment and RTL source code. This gives the designer the flexibility to modify the back-end interface to meet the requirements of the interfacing system. Using the design's fully-developed test bench to verify its functionality, both new and experienced designers will quickly be "up and running." Although this design is not guaranteed to be fully PCI 2.2 compliant, efforts have been made to ensure its conformity.

Design Goals and Limitations

The following goals were considered during development of this reference design:

- 33MHz PCI and back-end interface clock speeds
- 32-bit PCI and back-end I/O interfaces
- Support for two base address regions (I/O and memory regions)
- Single cycle and burst mode support for read and write cycles
- Implementation of all required PCI configuration registers
- Support for one interrupt signal from the back-end device to the PCI bus
- Parity generation for all read cycles
- Strive for compliance with all PCI 2.2 requirements
- Implementation in the latest LatticeECP3, LatticeXP2, MachXO and ispMACH devices
- Hierarchical HDL design, for simple end-user modifications
- A fully-automated and self-checking HDL test bench for ease of verification

The PCI Target does not support the following features:

- PERR and SERR (optional in embedded systems)
- Expansion ROM
- Built-in Self Test (BIST)
- Burst cycles into the configuration register space
- Cache line register

© 2011 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal. All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Theory of Operation

Overview

PCI is a highly documented specification. A special interest group (SIG) called the PCI SIG publishes the *PCI Local Bus Specification* (currently at Rev. 2.2). To understand the basic functionality of the PCI Bus, Rev. 2.2 is recommended reading for the PCI bus system designer. This reference design will not detail the functionality or electrical characteristics of the specification. Please review the latest version of the specification for more details. Consult *PCI System Architecture* by Tom Shanley and Don Anderson for a clearer interpretation of the specification.

The PCI bus is a multiplexed bus, where the same bus is shared for both address and data. During the address phase, a PCI command is present on the PCI_CBE_L bus telling the target device what PCI cycle is taking place. In the data phase, the same PCI_CBE_L bus is used to determine which bytes of data are enabled during the transaction. Bus cycle timing between the initiator and target PCI devices is handled by PCI_FRAME_L, PCI_TRDY_L, PCI_IRDY_L, PCI_STOP_L, PCI_DEVSEL_L, PCI_IDSEL, PCI_INTA_L, PCI_CLK and PCI_RST. Cycles supported by this PCI target reference design are I/O Read and Write, Memory Read and Write, and Configuration Read and Write.

The back-end interface of this reference design is user-configurable. The back-end device is provided with 20 bits of address space and a 32-bit data bus. During the data phase, the byte enables are passed from the PCI bus to the back-end device and two separate chip selects are provided, one for each base address region. Bus cycle timing between the back-end device and the PCI target is controlled by R_W_L, DATA_WRITE_L, DATA_READ_L, READY_L, DATA_STOP_L, BKEND_INT_L, BKEND_ABORT_L and PCI_CLK.

Functional Description

This 33MHz, 32-bit PCI target reference design is used to interface a back-end device that does not support the PCI protocol to the PCI bus. The PCI target responds to PCI cycles started by a PCI initiator. The PCI target functions as a data path controller, transferring data to and from the back-end device onto the PCI bus. When a particular PCI initiator has been granted the PCI bus, it performs read or write cycles with PCI targets on the bus. The PCI target that decodes the cycle address as a “base address hit” acknowledges and completes the cycle.

Figure 1. PCI Target I/O Interface

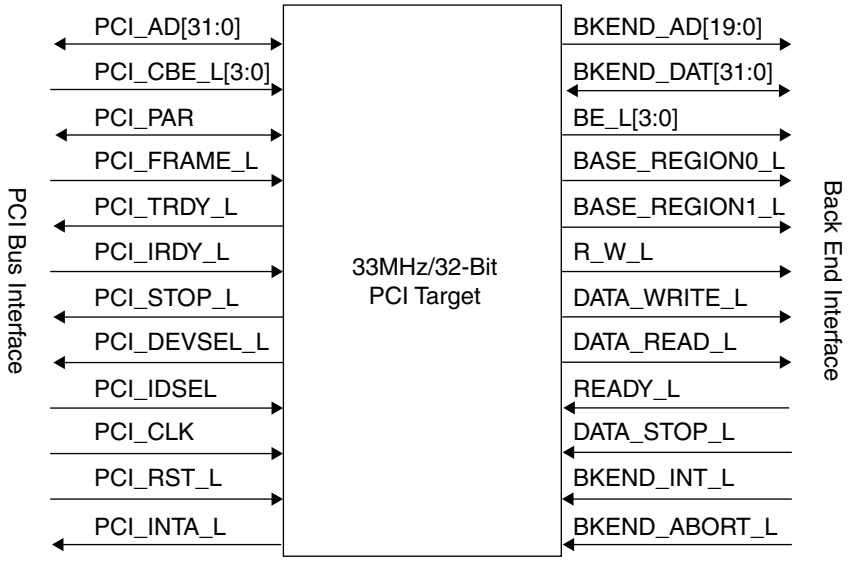


Table 1. PCI Signal Descriptions

Name	Direction	Active State	Description
PCI_AD	Bi-Directional	N/A	The multiplexed PCI address/data bus
PCI_CBE_L	Input	Active Low	The multiplexed PCI command/byte enables
PCI_PAR	Bi-Directional	Even Parity	The even parity bit. The PCI target drives this signal during read cycles. The PCI initiator drives this signal during the address phase of all transactions and the data phase during writes.
PCI_FRAME_L	Input	Active Low	The PCI initiator drives this signal low at the beginning of a cycle and high at the clock edge before the last data phase on a burst operation.
PCI_TRDY_L	Output	Active Low	The PCI target drives this signal low prior to the positive edge of a clock when it can complete a data phase.
PCI_IRDY_L	Input	Active Low	The PCI initiator drives this signal low prior to the positive edge of a clock when it can complete a data phase.
PCI_STOP_L	Output	Active Low	The PCI target drives the stop signal low during a transaction to indicate the termination of a cycle, signaling a retry, disconnect or abort (consult Rev. 2.2 of the specification).
PCI_DEVSEL_L	Input	Active Low	The PCI target drives DEVSEL low to indicate the address of the current transaction is in the address space of one of the base address registers.
PCI_IDSEL	Input	Active High	The PCI initiator will drive the IDSEL signal high at the input of the PCI target that should complete the current configuration cycle on the PCI bus.
PCI_CLK	Input	Positive Edge Sensitive	The clock input to all PCI devices on the PCI bus including PCI targets, PCI initiators and PCI arbiters.
PCI_RST_L	Input	Active Low	The active low reset for all PCI devices on the PCI bus.
PCI_INTA_L	Output	Active Low	The interrupt signal passed through the PCI target from the back-end device.

Table 2. Back-End Signal Descriptions

Name	Direction	Active State	Description
BKEND_AD	Output	N/A	The 20-bit address bus for the back-end device.
BKEND_DAT	Bi-Directional	N/A	The 32-bit data bus for the back-end device.
BE_L	Output	Active Low	The active low byte enables are passed through to the back-end device from the PCI bus. They can be used to determine which byte lane is active.
BASE_REGION0_L	Output	Active Low	This signal goes low to tell the back-end device a transaction is starting, and the cycle is accessing the address space decoded by Base Address Register 1.
BASE_REGION1_L	Output	Active Low	This signal goes low to tell the back-end device a transaction is starting and that the cycle is accessing the address space decoded by Base Address Register 0.
R_W_L	Output	1 = RD 0 = WR	This output tells the back-end device if the current transaction is going to be a read or a write.
DATA_WRITE_L	Output	Active Low	This signal is a write enable for the back-end device. Once the back-end device asserts READY_L a transaction will start, and a write is performed on a positive going transition (PGT) of PCI_CLK while DATA_WRITE_L is asserted.
DATA_READ_L	Output	Active Low	This signal is a read enable for the back-end device. Once the back-end device asserts READY_L a transaction will start, and a read is performed on a PGT of PCI_CLK while DATA_READ_L is asserted.
READY_L	Input	Active Low	The READY_L signal comes from the back-end device and is active following the assertion of BASE_REGION0_L or BASE_REGION1_L. This signals to the PCI target state machine that the back-end device is ready to start the transaction.
DATA_STOP_L	Input	Active Low	This active low input comes from the back-end device and signals the PCI target state machine that the back-end device will need to stop the burst cycle in two data phases. *Note: the back-end device is required to tie this signal low if it does not support bursts.*
BKEND_INT_L	Input	Active Low	The interrupt signal is an active low input and is passed through the PCI target to the PCI bus.
BKEND_ABORT_L	Input	Active Low	The back-end device asserts this active low input to the PCI target when a catastrophic failure is encountered.

Back-End Design Requirements

The following list contains requirements that the back-end device must follow to ensure proper operation of the system:

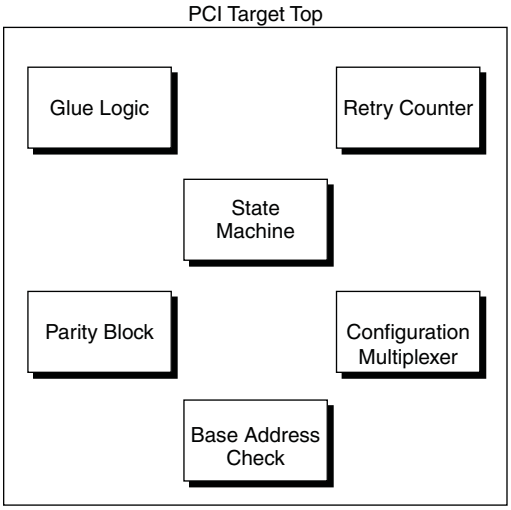
- The back-end device must tie DATA_STOP_L to a logical zero if it cannot support burst mode
- If the back-end device can handle bursts, then DATA_STOP_L should be a logical one at the start of a cycle
- Once the back-end device starts a burst cycle, it is not permitted to insert wait states. It must support full bursts or stop the transaction.
- In order for the back-end device to stop a burst transaction, it must assert DATA_STOP_L two clock cycles ahead of where a FIFO would be empty or full. On a FIFO, application DATA_STOP_L is a function of the almost empty and almost full flags.

Register Transfer Level (RTL) Implementation

The HDL code for this reference design contains the following functional blocks:

- PCI target top
- Miscellaneous glue logic block
- Configuration multiplexer block
- Base address check block
- State machine
- Parity generation block
- Retry counter block

Figure 2. The 33MHz, 32-Bit PCI Target Block Diagram



PCI Target Top

This is the top-level HDL block of this reference design. This block was created to instantiate the tristate buffers, and all the lower-level blocks of the design.

Miscellaneous Glue Logic Block

This module contains the miscellaneous glue logic required for the design. It contains the PCI address registers, CBE registers and the IDSEL register. It latches these during the address phase of any PCI transaction.

Configuration Multiplexer Block

This block implements the PCI target configuration registers and the PCI data output MUX. It controls when data is written into the configuration registers and what data is presented onto the output data bus (PCI_DAT_OUT) during reads.

The read values for the base address registers are set in this block. BA0 and BA1 always return the size of the memory block for the back-end application when read. All other read-only configuration registers are hard-wired in this block and the user can modify them in accordance with the *PCI Local Bus Specification, Rev. 2.2*.

The designer may modify this block of the HDL code to implement features of configuration registers the application-specific design may require. The HDL code is broken up into sections implementing the specific configuration registers, and the designer can add or remove registers depending on the system requirements. Consult the comments embedded in the HDL code for specific implementations of the following registers.

Configuration Register Implementation

Table 3. Reg00h Device ID and Vendor ID

Bit	Direction	Name	Description
31-16	R	Device ID	The device ID is user-defined by a vendor. It is hard wired to 0120h in this design.
15-0	R	Vendor ID	The vendor ID identifies the manufacturer, and is set to AMD = 1022h

Table 4. Reg04 Status and Command

Bit	Dir.	At Reset	Name	Description
31-28	R	0	Status Bits	Unimplemented status bits that return 0's when read.
27	R/W	0	Status Bit - Signalled Target Abort	Set to a 1 when the target aborts a transaction. Reset to 0 by writing a 1 to this register.
26-25	R	10b	Status Bits - DEVSEL Timing	Status bits for DEVSEL timing. Hard wired to 10b = slow.
24-16	R	0	Status Bits	Unimplemented status bits that return 0's when read.
15-2	R	0	Command Bits	Unimplemented command bits that return 0's when read.
1	R/W	0	Command Bit - Memory Space Enable	This command register bit enables accesses to base address regions configured as memory space. This register MUST contain a 1 for the PCI target to respond to memory transactions. Make sure PCI configuration software writes a 1 to this register during configuration.
0	R/W	0	Command Bit - I/O Space Enable	This command register bit enables accesses to base address regions configured as I/O space. This register MUST contain a 1 for the PCI target to respond to I/O transactions. Make sure PCI configuration software writes a 1 to this register during configuration.

Table 5. Reg08h Class Code and Revision ID

Bit	Direction	Name	Description
31-8	R	Class Code	The class code identifies the function of the PCI target and its subclass. This is hard wired to 0580h.
7-0	R	Revision ID	The revision number of this PCI target. Hard wired to 01h.

Table 6. Reg0Ch BIST, Header Type, Latency Timer and Cache Line Size

Bit	Direction	Name	Description
31-0	R	BIST, Header Type, Latency Timer, & Cache Line Size	Miscellaneous registers hard wired to 0000_0000h.

The base address register's BA0 and BA1 can be configured as either memory or I/O regions. The HDL is written such that Reg10h is for I/O space and Reg14h is for memory space. The designer can modify the HDL code for the configuration multiplexer block to change the size, location or region type for the registers.

Table 7. Reg10h Base Address 0 Configured for I/O Space

Bit	Direction	Name	Description
31-2	R/W	Base Address 0 Register	Base Address Register 0 is configured as an I/O region. Hard wired to a 1MB of memory mapped I/O (size can be changed).
1	R	Reserved	Set to 0b
0	R	I/O Space Indicator	Set to 1b indicating I/O space.

Table 8. Reg14h Base Address 1 Configured for Memory Space

Bit	Direction	Name	Description
31-4	R/W	Base Address 1 Register	Base Address Register 1 is configured for a memory region. Hard wired to allocate a 1MB block (size can be changed).
3	R	Prefetchable	Set to 0b (Prefetch OFF)
2-1	R	Type	Allocates where in memory this region must reside. Hard wired to 00b meaning anywhere in the 32-bit address space.
0	R	Memory Space Indicator	Set to 0b indicating memory space.

Table 9. Reg2Ch Subsystem ID and Subsystem Vendor ID

Bit	Direction	Name	Description
31-16	R	Subsystem ID	Hard wired to 0120h
15-0	R	Subsystem Vendor ID	Hard wired to 1022h

Table 10. Reg3Ch Max_Lat, Min_Gnt, Interrupt Pin and Interrupt Line

Bit	Direction	Name	Description
31-16	R	Max_Lat, and Min_Gnt	Miscellaneous registers hard wired to 0000h
15-8	R	Interrupt Pin	The interrupt pin is hard wired to 01h indicating only one interrupt is available.
7-0	R/W	Interrupt Line	The configuration software will write the interrupt line register to set the system IRQ used for this device.

Base Address Check Block

The base address check block has two functions. It is used to control the write-only registers for Reg10h through Base Address 0 and Reg14h through Base Address 1. The top-level “address decode” for both regions are stored here. The second function of this block is to decode an address hit in Base Address Region 0 or 1 and assert a signal informing the PCI target state machine. If only one base address region is used, the designer can remove the check for the unused region from the HDL.

The State Machine

Description

The PCI target state machine is at the heart of this reference design. It controls the bus cycle timing of all data flow paths to and from the PCI bus and back-end bus. The state machine goes from the idle state to one of three possible paths during any given PCI operation. While in the PCI address phase of a transaction the values on PCI_CBE_L and PCI_IDSEL determine if the transaction is a configuration read or write, memory-I/O read or a memory-I/O write.

The designer can modify the state machine HDL code if the functionality of the back-end device has special system requirements. It may be necessary to implement wait states, or new enable signals for particular applications. These require modifications to the HDL code, but embedded HDL comments will guide the designer through the state machine implementation and signal descriptions.

Configuration Read Write Cycles

The PCI target only responds to certain types of configuration transactions. The HDL is coded so the target only responds to type 00 transactions (PCI_AD[1:0] == 00b) directed at function 0 (PCI_AD[10:8] == 000b). During configuration cycles, the address bits PCI_AD[7:2] determine which configuration double word is being accessed, while the PCI_CBE_L[3:0] bits determine if the cycle is a configuration read or configuration write.

The target will respond to configuration reads to unimplemented registers, and return a value of 0000_0000h. The target will respond to writes to a configuration register that is not implemented, although the data will never be written.

Burst addressing of the configuration registers is uncommon and is not supported.

For further information on PCI configuration transactions, consult the *PCI Local Bus Specification, Rev. 2.2*.

Memory – I/O Read Cycles

The PCI target state machine supports unlimited length burst reads and single cycle reads. During a typical operation, the state machine determines if the address is a hit to one of its base address regions. It accepts the cycle, and informs the back-end device of a transaction by asserting BASE_REGION0_L or BASE_REGION1_L. After the back-end acknowledges the cycle by asserting READY_L, the first double word is read. If the transaction is a burst, the next double word is then read. Once the master starts a burst, if the back-end device supports bursts, it must assert DATA_STOP_L to inform the target state machine when it can no longer provide burst read data. The state machine expects to see DATA_STOP_L asserted, two data phases before the back-end must halt the burst reads. This gives the state machine time to stop the cycle, smoothly. If a catastrophic failure occurs, the back-end device can assert BKEND_ABORT_L at any time once the read operation begins.

Memory – I/O Write Cycles

The PCI target state machine supports unlimited length burst writes and single cycle writes. Just as the read cycles worked, during a typical operation the state machine determines if the address is a hit to one of its base address regions. If the target accepts the cycle, the state machine informs the back-end device of a transaction by asserting BASE_REGION0_L or BASE_REGION1_L. The back-end device acknowledges the cycle by asserting READY_L. The PCI target will assert PCI_TRDY_L, and if the initiator has PCI_IRDY_L asserted, a data write will occur. If the transaction is a burst, the next double word can be written at the next clock edge. Once the initiator starts a burst, if the back-end device supports bursts, the back-end must assert DATA_STOP_L to inform the target state machine when it can no longer accept burst write data. The state machine expects to see DATA_STOP_L asserted, two data phases before the back-end must halt the burst writes. This gives the state machine time to stop the cycle, smoothly. If a catastrophic failure occurs, the back-end device can assert BKEND_ABORT_L at any time once the write operation begins.

Parity Generation Block

Parity is generated on configuration read, and memory-I/O reads cycles. The parity signal PCI_PAR is pipelined to guarantee the 11ns t_{CO} requirement for 33MHz systems, in accordance with the *PCI Local Bus Specification, Rev. 2.2*. The parity generated by the PCI initiator is not checked in this design. Most applications utilizing this reference design will be in embedded systems where PERR and SERR are unmonitored. Simple HDL code modifications can be made to implement these signals, and the hard XOR of the ispMACH 4A3 devices will provide the designer the flexibility to easily meet all timing and area constraints.

Retry Counter Block

If the PCI target acknowledges a read or write cycle, it must provide or accept data within 16 clock cycles of asserting DEV_SEL. If the PCI target acknowledges a cycle, and the back-end device does not assert READY_L within 12 clock cycles, the PCI target will initiate a data retry. A data retry can only be asserted before the first data phase completes. Once the back-end device acknowledges the cycle, it must provide or accept the requested data. The retry counter block implements the counter used to signal the state machine when the back-end device “time out” occurs.

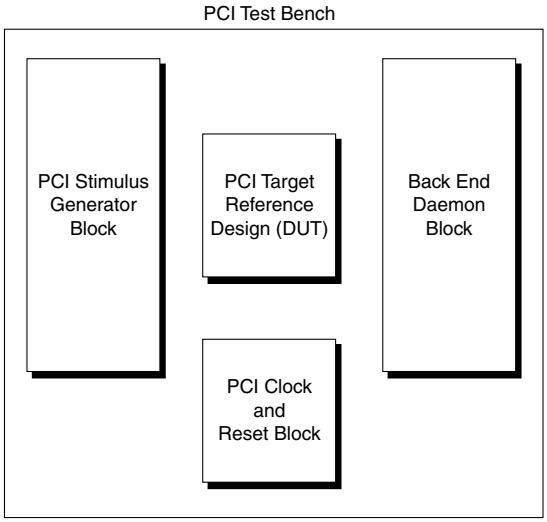
HDL Verification

The Functional Blocks of the HDL Test Suite

The HDL code for the 33MHz, 32-bit PCI target reference design test suite contains the following functional blocks:

- PCI test bench
- PCI stimulus generator
- PCI clock and reset block
- Back-end daemon
- Device-under-test (PCI target)

Figure 3. 33MHz, 32-Bit PCI Target Test Suite



PCI Test Bench

The PCI test bench is a self-contained HDL test suite used to verify the timing and functionality of the 33MHz, 32-bit PCI target reference design. The HDL test bench is a fully-automated, self-checking test environment that tests all normal and corner case scenarios that a PCI target will encounter interfacing to a typical back-end device.

The Device-Under-Test (The PCI Target)

The Device-Under-Test (DUT) is the 33MHz, 32-bit PCI target reference design. For RTL simulations, the DUT is the Verilog or VHDL RTL source code implementation of the design prior to synthesis. The gate-level HDL netlist and its associated Standard Delay File (SDF) are written by Lattice ispLEVER® software after fitting the design. The gate-level netlist and SDF are then used in the gate-level simulations with timing to verify functionality and timing paths.

PCI Clock and Reset Block

The PCI clock and reset block are used to generate the 33MHz PCI and back-end clock and generate a 120ns active low reset pulse for the test suite.

Back-End Daemon

This block is a daemon or behavioral model used to model the functionality of a generic back-end device. A daemon is a special simulation module that once activated will operate without user intervention to emulate the functionality of a particular interface. Any back-end device, whether an SDRAM, SRAM, FIFO or simple I/O device, will need to implement control logic to arbitrate the back-end signals with the PCI target state machine. The control logic can easily be added to the free space available in the ispMACH 4A3 device for the specific application. This

daemon simply arbitrates the back-end signals and reads and writes the data to the appropriate memory bank. Depending on the address and bank written to, this daemon will respond normally, or with a retry, a stop, an abort or an interrupt. There are two banks of memory in the daemon, and they correspond to the two base address regions allocated inside the PCI target. Base Address Region 0 uses BK0 and Base Address Region 1 uses BK1. All reads and writes to BK0 will function normally. BK1 has added functionality to test a retry, a stop, and an abort. To use the special features of BK1, the cycle must start with the specified address. The daemon WILL burst by the address with no effect if the special address is not the starting address.

- If a write is made to address 000A0h and BK1, the daemon will NOT respond with READY_L, causing a data retry.
- If a write is made to address 000B0h and BK1, the daemon will start the cycle then respond with DATA_STOP_L == 0 causing a data stop.
- If a write is made to address 000C0h and BK1, the daemon will start the cycle, but respond with BKEND_ABORT_L == 0 an abort.
- If a write is made to address 000D0h and BK1, the daemon will run the cycle, but respond with BKEND_INT_L == 0 causing an interrupt to be passed through the PCI target.
- if a write is made to address 000E0h and BK1, the daemon will run the cycle, but respond with BKEND_INT_L == 1 causing the interrupt to be disabled.

PCI Stimulus Generator

The PCI stimulus generator block is the heart of the PCI test suite. The stimulus generator creates all the PCI cycles as a PCI initiator would on a PCI bus, and also checks the data returned from the PCI target and the condition of its control signals. All the simulation tasks and functions are defined and executed in this block. It controls all the automation and self check verification features of the whole test suite. Each major task or function is derived of minor tasks or functions used to generate the transactions. The following is a list of tasks contained in this module:

- **pci_reset** – Sets the stimulus block up for a reset.
- **read_config** – Performs a PCI configuration read.
- **write_config** – Performs a PCI configuration write.
- **read_cycle** – Performs a single-cycle, burst, memory or I/O PCI read cycle.
- **write_cycle** – Performs a single-cycle, burst, memory or I/O PCI write cycle.
- **pci_sniff** – Calls the entire configuration read and write cycle set needed to initialize the PCI target.
- **write_test** – Calls memory and I/O read write cycles to test burst and single-cycle transactions.
- **write_special_cycle** – Calls special read and write cycles to test data retry, data stop, data abort, data interrupt on/off.
- **single_cycle_only_test** – Calls a test that ensures that if the back-end device ties data_stop_l low, the PCI target only accepts single-cycle transactions or issues a data stop after the first data phase.
- **read_special_cycle** – Calls a test that performs data reads when the back-end asserts data_stop_l mid-burst.
- **check_cycle** – Tests the termination of any PCI cycle and determines if the cycle was a data transfer, a data transfer with a stop, a data retry, an abort or a violation of the PCI protocol. The result of this check is passed onto the other tasks for self-checking.
- **kill_time** – Runs the simulation and spaces out PCI transactions by five PCI clock cycles.
- **check_data** – Checks data returned by the PCI target during configuration, memory or I/O reads and compares them with expected values.
- **check_parity** – Checks the PCI parity value returned during a configuration, memory or I/O read and compares it with the expected parity value.

The 33MHz, 32-Bit PCI Target Transaction Waveforms

The following waveforms illustrate a typical operation of the 32-bit PCI target reference design implemented in an M4A3-384/192-65FAC. All the waveforms were taken from functional gate-level simulations using the ModelSim® simulator from Model Technology®. ispLEVER software can export a gate-level HDL netlist and an SDF for use in at-speed gate-level simulations. These simulations are run at 33MHz using typical gate-level delays from the Standard Delay File (SDF).

Figure 4. 33MHz, 32-Bit PCI Target – Configuration Write

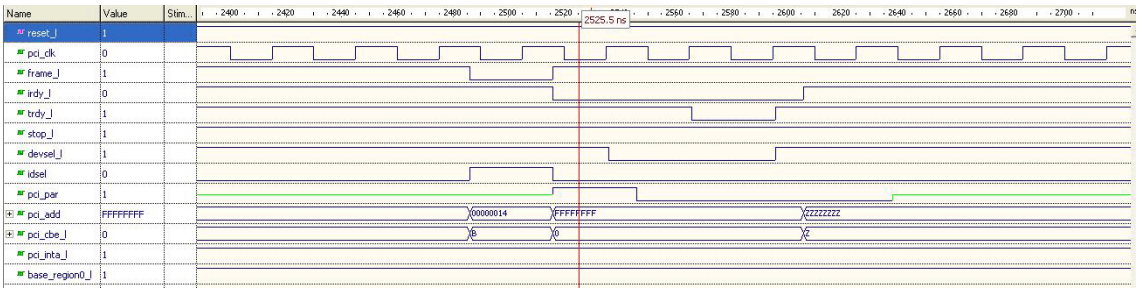


Figure 5. 33MHz, 32-Bit PCI Target – Configuration Read

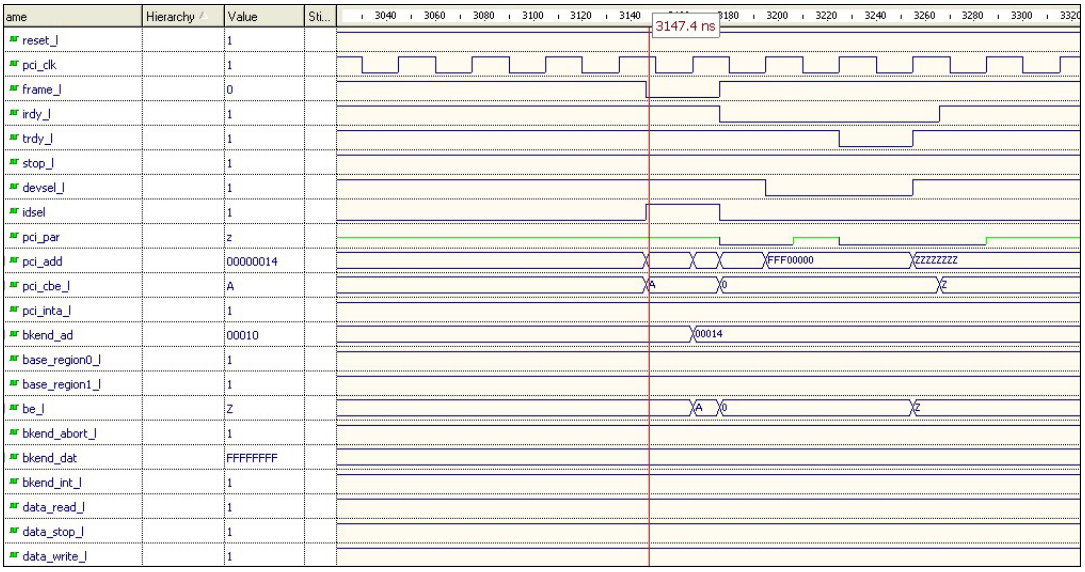


Figure 6. 33MHz, 32-Bit PCI Target – Single Write

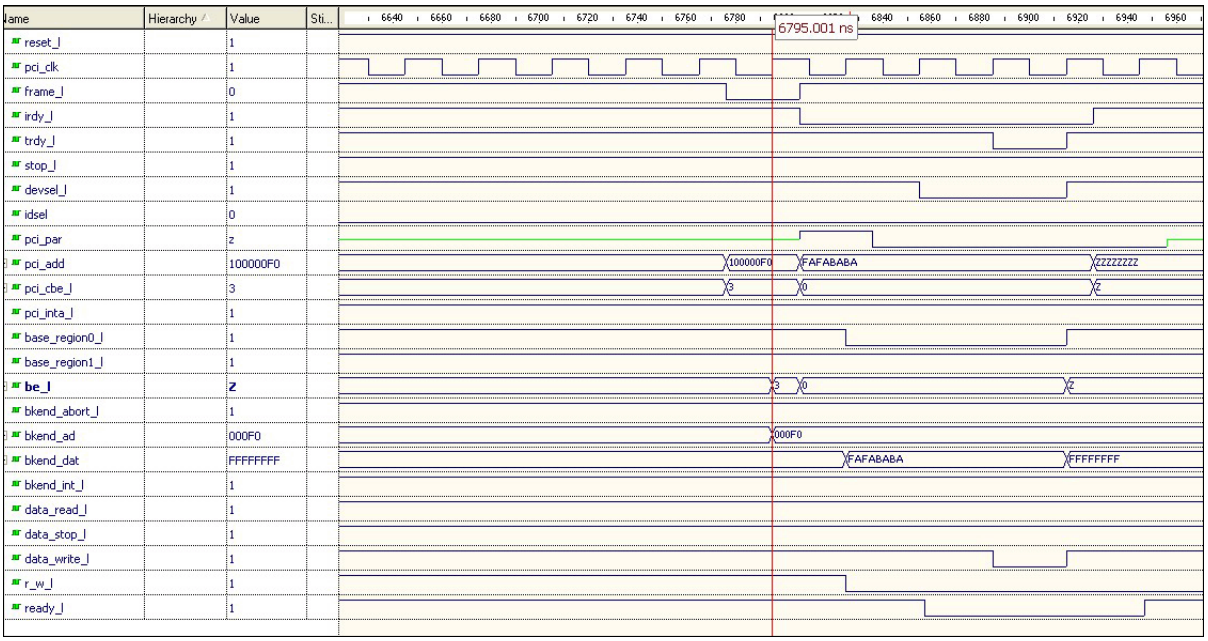


Figure 7. 33MHz, 32-Bit PCI Target – Single Read

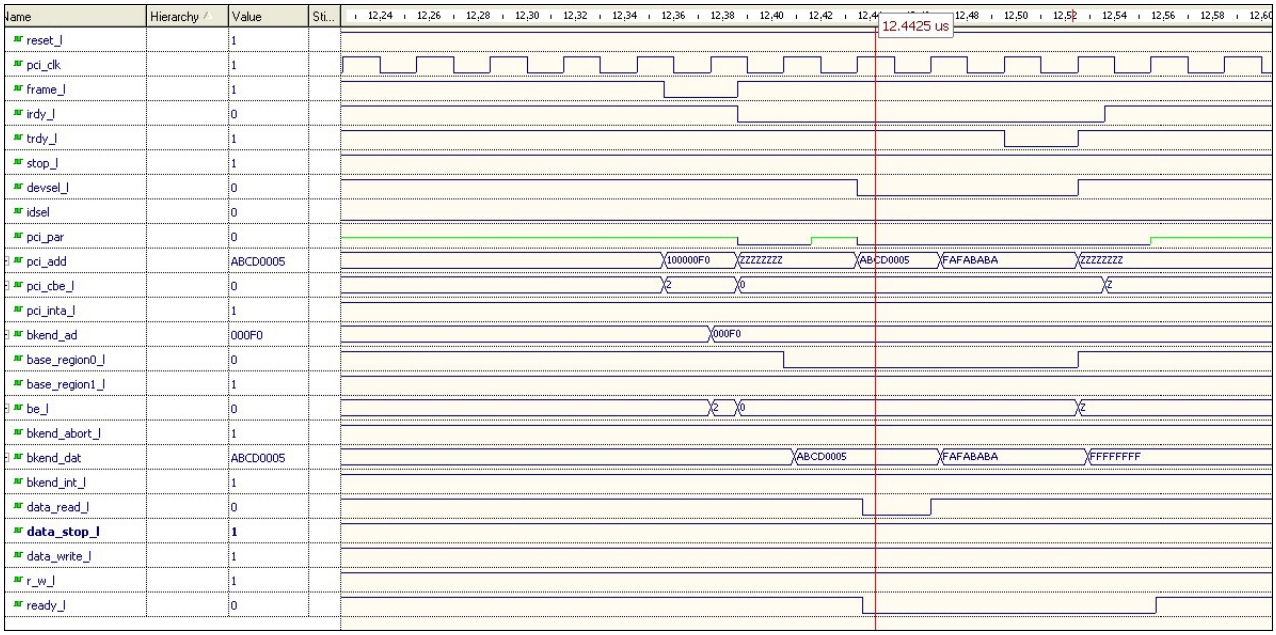


Figure 8. 33MHz, 32-Bit PCI Target – Burst Write

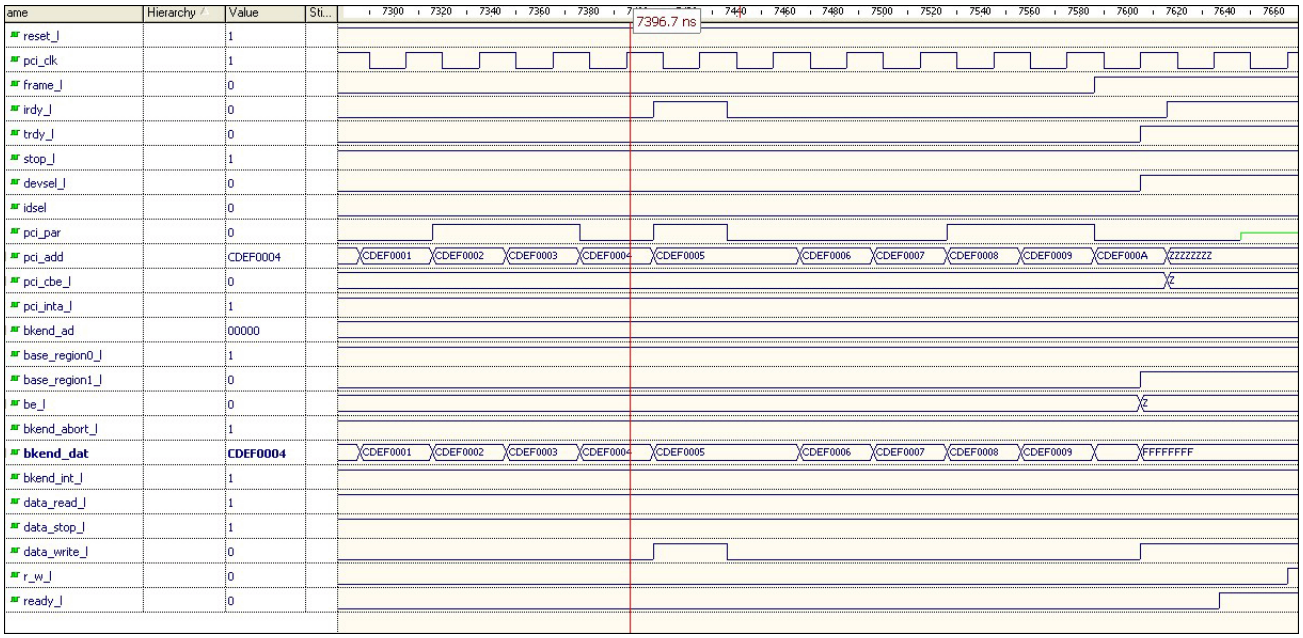
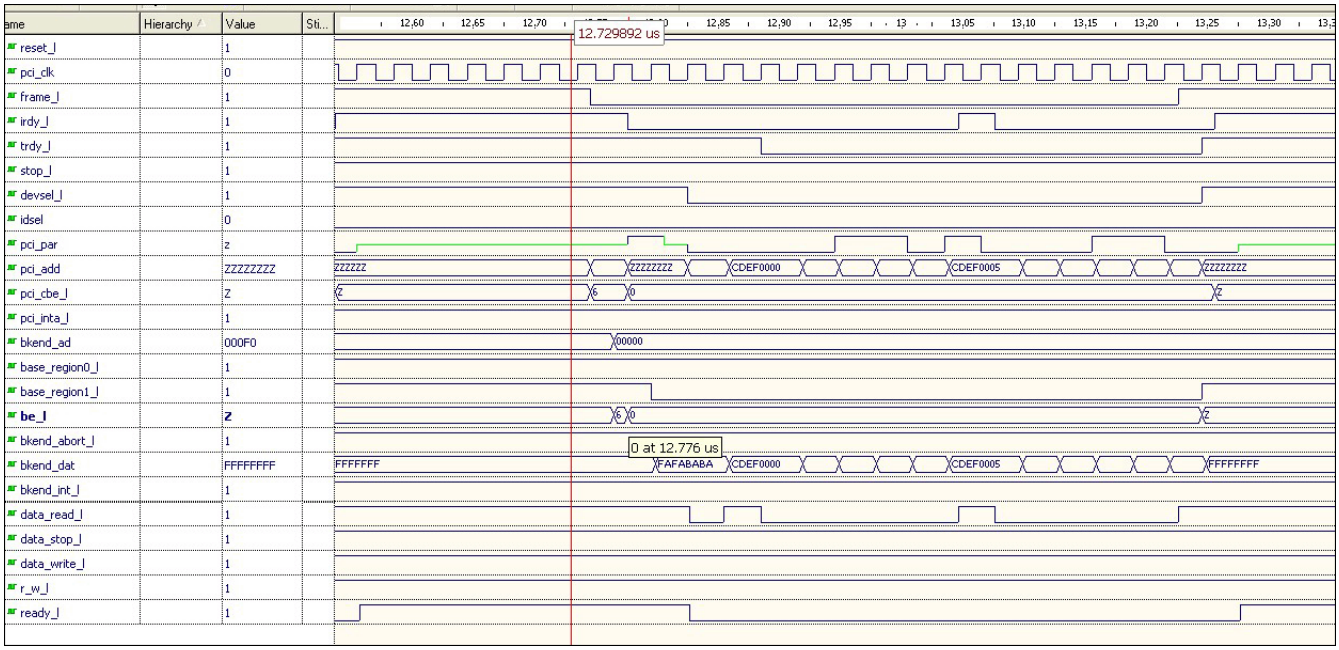


Figure 9. 33MHz, 32-Bit PCI Target – Burst Read



Implementation

This design is implemented in Verilog and VHDL. When using this design in a different device, density, speed, or grade, performance and utilization may vary. Default settings are used during the fitting of the design.

Table 11. Performance and Resource Utilization

Device Family	Language	Speed Grade	Utilization	f_{MAX} (MHz)	I/Os	Architecture Resources
LatticeECP3™ [1]	Verilog	-8	250 LUTs	33	111	N/A
	VHDL	-8	247 LUTs	33	111	N/A
LatticeXP2™ [2]	Verilog	-7	252 LUTs	33	111	N/A
	VHDL	-7	251 LUTs	33	111	N/A
MachXO2™ [3]	Verilog	-6	248 LUTs	33	111	N/A
	VHDL	-6	257 LUTs	33	111	N/A
MachXO™ [4]	Verilog	-5	244 LUTs	33	111	N/A
	VHDL	-5	249 LUTs	33	111	N/A
ispMACH® 4000[5]	Verilog	-3.5ns	270 Macrocells	33	111	N/A
	VHDL	-3.5ns	270 Macrocells	33	111	N/A

1. Performance and utilization characteristics are generated using LFE3-70E-8FN484C, with Lattice Diamond™ 1.2 software.
2. Performance and utilization characteristics are generated using LFXP2-5E-7FT256C, with Lattice Diamond 1.2 software.
3. Performance and utilization characteristics are generated using LCMXO2-2000HC-6FTG256C, with Lattice Diamond 1.2 software.
4. Performance and utilization characteristics are generated using LCMXO-2280C-5FT256C, with Lattice Diamond 1.2 software.
5. Performance and utilization characteristics are generated using LC4512V-35176C, with Lattice ispLEVER® Classic 1.4 software.

Implementing a Design in MachXO and LatticeXP2 Devices

The design requires 46 PCI33 compatible interface signals and a total of 111 I/O pins.

The MachXO1200 and MachXO2280 devices support 3.3V PCI on the top bank of the I/O buffers of the device. Therefore, MachXO devices that can support PCI33 are the LCMXO1200 and LCMXO2280 devices in 256-ball or larger packages.

For LatticeXP2 devices, the 3.3V PCI is supported on both the top and bottom banks of the I/O buffers of the device. This design can be supported in any LatticeXP2 device that has enough usable I/O pins to meet the I/O pin requirement.

Conclusion

This reference design is free to the design engineer implementing digital systems using Lattice devices. Except for the LatticeECP3 and ispMACH 4A3 device families, 3.3V PCI I/O buffers are supported in specific banks of the device. The implementation results shown in Table 11 provide guidelines for selecting the appropriate device speed grade to meet the PCI specification.

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
—	—	Previous Lattice releases.
July 2007	03.1	Updated Implementation Using MachXO Devices section.
September 2009	03.2	Added VHDL support.
January 2010	03.3	Added support for LatticeXP2 and ispMACH 4A3 device families.
April 2011	03.4	Added support for LatticeECP3 device family.
		Added support for Lattice Diamond 1.2 design software.
August 2013	03.5	Updated document with new corporate logo.
		Updated Table 7 title to Reg10h Base Address 0 Configured for I/O Space.