XILINX®

PING64 Application Example

April 2, 2003 (Version 3.0) Application Note

Summary

This application note describes the **PING64** example design. This example design is intended for use as a design flow test. For detailed instructions on how to run this design through synthesis and implementation tools, please consult the appropriate implementation guide.

Xilinx Families

Spartan-II(E), Spartan-III, Virtex(E), Virtex-II(Pro)

PCI LogiCORE Version

PCI64

Introduction

This application example, available in VHDL and Verilog, provides an easy to understand example which demonstrates some of the techniques required to successfully use the PCI LogiCORE in a design.

PING64 consists of a top level VHDL or Verilog testbench named **PING_TB**. This testbench contains HDL behavioral stimulus files and **PCIM_TOP**. The stimulus files contain behavioral models of an arbiter, two targets (one 32-bit target and another 64-bit target), and a central resource. These behavioral modules interface to **PCIM_TOP** through the PCI bus. The HDL wrapper, **PCIM_TOP**, combines three sub-modules. The first sub-module is **PCIM_LC**, a wrapper for the LogiCORE. The second is the **PING64** application. The third is the **CFG** module, which configures the LogiCORE interface. The **PING64** sub-module is an HDL design that interfaces to the user-application signals from the **PCIM_LC** module. Figure 1 shows a block diagram of the complete system.

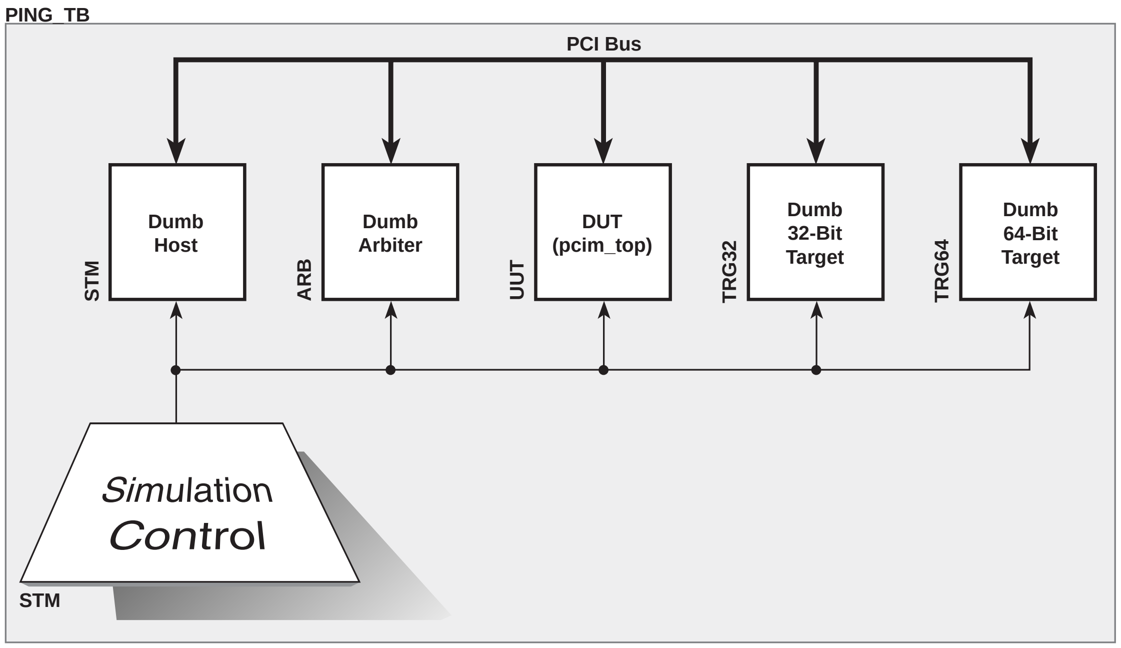

Figure 1: System Block Diagram

LogiCORE Configuration

To configure the LogiCORE PCI interface, a special configuration file, `cfg_ping`, is used. This file must not be modified or **PING64** will fail to respond properly to the simulation controller (stimulus generator). Note that the `cfg_ping` configuration file should not be used when generating a new design. Instead, use the `cfg` configuration file provided in the `src/xpci/` directory.

Ping User Application

The **PING64** module takes its name from the TCP/IP utility named ping which allows network users to verify that a particular machine is on a network and “alive”.

As such, **PING64** is designed to provide “just enough” functionality to verify a design flow. The **PING64** design provides little other utility. The **PING64** design accepts data as a target PCI device, then uses the data to perform initiator transactions over the PCI bus.

Figure 2 is a reduced block diagram of **PING64**. For specific details about the implementation, consult the source code. A summary follows.

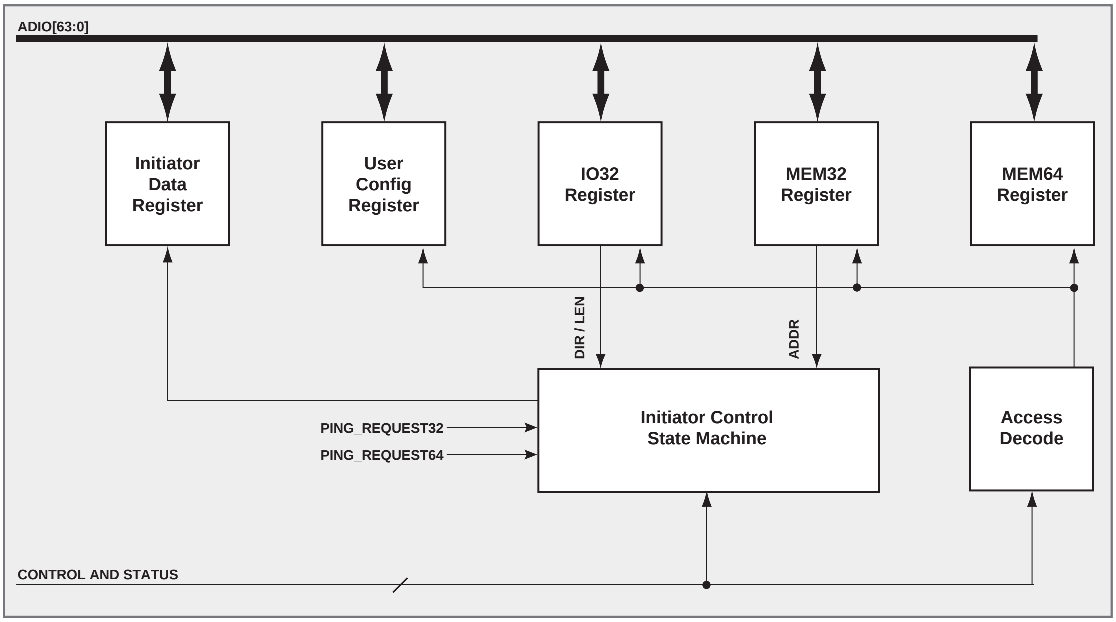

Figure 2: PING64 Block Diagram

Access Decode

The access decode logic is used to generate configuration space, memory space, and I/O space select signals. This is achieved by monitoring the **`CFG_HIT`** and **`BASE_HIT`** signals which indicate that the current PCI transaction is directed at the LogiCORE configuration space or to an address space mapped by one of the base address registers in the PCI interface. In this design, BAR0 is configured as an I/O space, BAR1 is configured as a 32-bit memory space, and BAR2 is configured as a 64-bit memory space.

This block generates read and write select signals for each of the four address spaces.

User Configuration Register

This 32-bit register is mapped into the user configuration space of the PCI interface. This register is controlled by the `CFG_WR_CS` and `CFG_RD_CS` signals generated by the access decode block. This is a general purpose read/write register.

I/O32 Register

This 32-bit register is mapped into I/O space by BAR0. This register is controlled by the `BAR0_WR_CS` and `BAR0_RD_CS` signals generated by the access decode block. This is a general purpose read/write register.

MEM32 Register

This 32-bit register is mapped into memory space by BAR1. This particular memory space is designated as 32-bit only. This register is controlled by the `BAR1_WR_CS` and `BAR1_RD_CS` signals generated by the access decode block. This is a general purpose read/write register.

MEM64 Register

This 64-bit register is mapped into memory space by BAR2. This particular memory space is designated as 64-bit capable. This register is controlled by the `BAR2_WR_CS` and `BAR2_RD_CS` signals generated by the access decode block. This is a general purpose read/write register.

Initiator Control State Machine

The initiator control state machine is a reduced version of the initiator control state machine presented in the *Design Guide*. This state machine has five states. One state is an "IDLE" state. When either `PING_REQUEST32` or `PING_REQUEST64` is asserted, the state machine transitions into one of four data transfer states ("READ32", "WRITE32", "READ64", or "WRITE64"). After a transfer completes, the state machine returns to the "IDLE" state.

The supporting glue logic is taken directly from the *Design Guide*.

Initiator Data Register

This 64-bit register is the source and destination of data during initiator transactions initiated by **PING64**. This register is not memory mapped and is therefore not accessible through target transactions to **PING64**. During initiator writes, the data in this register is transferred to a target. During initiator reads, the incoming data is stored in this register. This register is controlled by the initiator control state machine.

Operation

A simulation testbench is provided in both VHDL and Verilog. This HDL testbench generates PCI transactions as a stimulus for **PING64**. A small PCI system is created by connecting the top level HDL design **PCIM_TOP** to a set of behavioral PCI agents.

Initially, the testbench configures **PING64** by writing to its configuration registers. Both the base address registers are loaded, and the master enable bit is set in the command register.

Target Operation

All target registers are accessible by issuing the appropriate bus transaction from the stimulus module. Although there is only a single register mapped by each base address register, **PING64** will accept burst transactions. For burst reads, the same data is transferred each data phase. During burst writes, only the last data transferred will remain in the register.

Initiator Operation

In order to enable **PING64** to perform initiator transactions, the testbench must set specific fields in the I/O32 and MEM32 registers. The 32-bit data in the MEM32 register is used as the PCI Bus address during initiator transactions. The I/O32 register has three separate fields. Bit 31 of this register indicates the direction of the transfer; logic one indicates write, while logic zero indicates read.

The PCI Bus command to be used during the transaction is specified in bits 7 through 5.

The transaction burst length (in data phases) is indicated by bits 3 through 0.

Assuming that the testbench has already set the master enable bit in the configuration header command register, simply write valid settings into the I/O32 and MEM32 registers and assert either `PING_REQUEST32` (for a 32-bit transaction) or `PING_REQUEST64` (for a 64-bit transaction).

Limitations and Restrictions

Xilinx, Inc. does not make any representation or warranty regarding these designs or any item based on these designs. Xilinx disclaims all express and implied warranties, including but not limited to the implied fitness of these designs for a particular purpose and freedom from infringement. Without limiting the generality of the foregoing, Xilinx does not make any warranty of any kind that any item developed based on these designs, or any portion of them, will not infringe any copyright, patent, trade secret or other intellectual property right of any person or entity in any country. It is the responsibility of the user to seek licenses for such intellectual property rights where applicable. Xilinx shall not be liable for any damages arising out of or in connection with the use of the designs including liability for lost profit, business interruption, or any other damages whatsoever.