A PCI-X Verification Environment Using C and Verilog

Kai-Hui Chang, Yu-Chi Su, Wei-Ting Tu, Yi-Jong Yeh, and Sy-Yen Kuo

Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan
sykuo@cc.ee.ntu.edu.tw

Abstract

Today many complex ASIC (Application Specific Integrated Circuit) designs use standard I/O interfaces in order to make sure that they can cooperate correctly with other components from different vendors. Moreover, many SoC (System-on-a-Chip) designs directly incorporate standard interfaces (such as PCI and AMBA AHB bus), and therefore system verification becomes important to ensure that all design blocks work together correctly. Among these standard interfaces in the world, PCI/PCI-X is probably the most popular and widely used one. In this paper, we propose a PCI-X verification environment that integrates both C and Verilog in the testbench development. Two benchmarks, including a PCI-X to PCI-X bridge and a PCI-X core, are presented in this paper. From the experimental results, it is concluded that the proposed PCI-X verification environment can efficiently and effectively perform the verification for PCI-X designs.

1. Introduction

Verification is important for bus controllers to ensure its conformance to the protocol. Several methodologies have been proposed to achieve this goal, with different approaches including simulation-based [1] and formal [2][3] methods. However, with the release of the latest PCI-X 2.0 protocol [4], there are new challenges to the current verification environments. Therefore a reusable PCI-X verification environment is proposed in this paper. The environment is based on the PCI testing environment proposed in [5], and several enhancements are done for the PCI-X 2.0 bus specification. New features, including checklist profile, transaction database, and end-to-end checker, are also proposed in this verification environment. The end-to-end checker provides the infrastructure for transaction-level verification that is new to the PCI bus verification environments.

2. Verification Architecture

The verification environment enables C and Verilog co-simulation to make the testbench development more flexible and efficient. The architecture of the verification environment is shown in Figure 1.

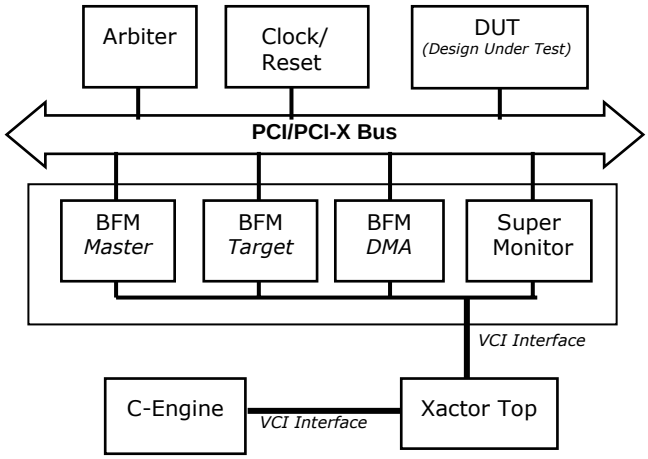

Figure 1. Verification Architecture.

2.1 Bus Functional Models (BFMs)

A bus functional Model acts as a pseudo device that interacts with other devices on the bus. In the verification environment, all the BFMs are based on the Verilog behavioral model and are controlled and driven by the C-based diagnostic driver. These BFMs include both PCI initiator (PCI Master) and PCI target (PCI Slave), and support the PCI standard including PCI 2.2, PCI-X 1.0, and PCI-X 2.0.

2.2 Diagnostic C-Engine

The Diagnostic C-Engine is a C-based program that is loosely coupled with the PCI BFMs to permit rapid simulation. It supports a diagnostic programming interface (DPI) that enables transaction-level test authoring, and provides a simple and powerful programming API to generate PCI transactions and check the results.

2.3 Super Monitor

The Super Monitor connects to one or more PCI buses at the top level, but is not considered a PCI device. Instead, it monitors all the activities of transactions that occur on the buses. The Super Monitor has four major functions including tracing analyzer, protocol checker, performance analyzer, and coverage analyzer.

Tracing analyzer prints information about transactions that occur on the bus, including address, command type, data transferred, byte enable, transaction time, and target response (retry, split-terminated, etc).

Protocol checker checks if any PCI-X protocol is violated. If a rule is violated, a message indicating the violated rule will be given. Performance analyzer measures bus utilization and request-to-grant latency, etc. Coverage analyzer reports transaction coverage like bus commands and byte enable combinations.

2.4 PCI Arbiter

The PCI arbiter is used to arbitrate the request signals from the PCI devices. It has two modes, free running and Diagnostic C-Engine controlled. In the free running mode, the arbiter is round-robin with bus parking. In the Diagnostic C-Engine mode, the Diagnostic C-Engine handles all the arbitration algorithms.

2.5 Compliance Test Suits

The verification environment provides compliance test suits for PCI 2.2, PCI-X 1.0, and PCI-X 2.0 [6]. A compliance test suit is a group of test scenarios written for the Diagnostic C-Engine by DPI. The compliance test suits are written according to those compliance checklists provided by the organization that constitutes the bus specification.

3. Enhancements

3.1 PCI-X 2.0

PCI-X 2.0 includes the following new features: Source synchronous mode, ECC protection, and device ID message.

In source synchronous mode, C/BE signals are used as the data strobe to transfer data. There can be two or four data strobes between two PCI clocks and transfers double or quadruple data per PCI clock compared with PCI-X 1.0.

ECC protection uses 7-bit code for 32-bit mode and 8-bit code for 64-bit mode. With ECC, single bit error can be corrected and double bit error can be detected. ECC provides better system reliability compared with conventional PCI bus.

Device ID message provides another addressing mode in PCI bus and enables simplified peer-to-peer transactions for applications such as streaming-media.

These new features are implemented in the PCI-X verification environment proposed in this paper.

3.2 Checklist Profile

There are checklists of protocol rules for the verification of a PCI-X design. The original PCI testing environment automatically checks these rules. However, it is not known whether adequate testbenches have been generated to test these rules or not. Therefore checklist profile is implemented so the test coverage of the rules can be reported.

There are two profiles for checklists. The first one is the rule violation profile, which reports the number of times that a rule is violated. The second one is the rule trigger profile, which reports the number of times that a rule has been checked. Since the testbench development tool [7] used in this verification environment provides a closed-loop profile to testbench feedback, it is possible to change the testbench generator to achieve a better rule check coverage.

For example, in rule XTP18 in the PCI-X checklist, it says:

If the target signals Disconnect at Next ADB four or more data phases from an ADB, the target expects the transaction to disconnect at that ADB.

For this rule, the rule trigger count for XTP18 profile is increased when the target signals Disconnect at Next ADB four or more data phases from an ADB. Later if the transaction does not disconnect at that ADB, the count for XTP18 rule violation profile is increased.

3.3 Transaction Database

Transaction database records transactions on the bus and can be used for higher-level check like data integrity, time-out, and transaction ordering. There are two types of transactions saved in the transaction database: Request transaction and data transfer transaction. Request transaction is a transaction that requests to transfer data on the bus but no data have been transferred. It occurs when a transaction is retried or split-terminated by the target. If data are transferred, then it is a data transfer transaction. It may be a split-completion or a write transaction with data transfer.

An event notifier is used to signal the outside world the occurrence of a transaction on the bus. It is a 1-bit register initialized to zero. Whenever a transaction occurs, the transaction will be saved in the transaction database, and then the notifier variable will toggle. The verification environment that uses the transaction database can thus be notified.

4. End-to-End Checker

End-to-end checker is used to check the correctness of PCI bus behavior at transaction level, including data integrity, transaction ordering, time-out, etc. It is built upon the transaction database. End-to-end checker is application specific and should be customized by the user. In the PCI-X verification environment proposed in this paper, basic infrastructures are provided for the users to write their own checkers.

4.1 Read Transaction Checker

The scenario of read transaction in the end-to-end checker is illustrated in Figure 2.

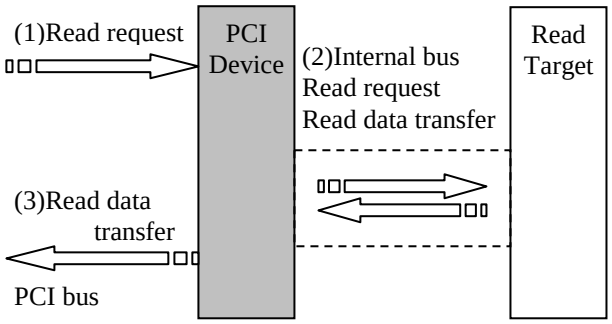

Figure 2. Read Transaction.

The tag-augmented Sequence is chosen for the implementation of read transaction checker because it provides a temporal logic checker and the functionality to carry information with the temporal flow. There are three steps to verify whether a read transaction is correct or not:

1. When a read request occurs, a Sequence is created. The request record is saved and carried with the Sequence. The address is also saved and is used to qualify the read transaction that should occur later on the bus. A time-out condition is also set by the Sequence.

2. When the read data transfer occurs, the Sequence qualified by the read address will report a success. The transaction database on the target will be searched for read transactions that cover the data returned. If no such read transactions are found, an error will be reported. If transactions are found, data integrity is checked.

3. If no read data transfer occurs on the bus, the time-out mechanism will report an error.

4.2 Write Transaction Checker

The scenario of a write transaction is illustrated in Figure 3.

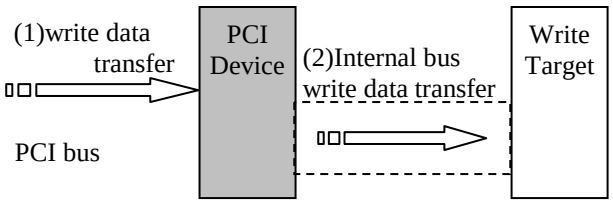

Figure 3. Write Transaction.

A write queue is used for the verification of write transactions. When a write occurs, the record of the write transaction is inserted in the queue. When the write on the target occurs, the originator's queue is searched based on the address of the transaction. Once the record is found, data integrity is checked.

It is possible that several write transactions will be combined to one write transaction on the target. In this case, more than one write transactions in the queue may be consumed by the write transaction on the target.

4.3 Example Verification

A PCI-X to PCI-X bridge is chosen as the example design to illustrate how end-to-end checker can be used to verify the behavior of a PCI-X circuit on transaction level.

4.3.1 Read and Write Checking

The read and write transactions that cross the PCI-X bridge is illustrated in Figure 4 and can be checked by the read and write transaction checkers directly.

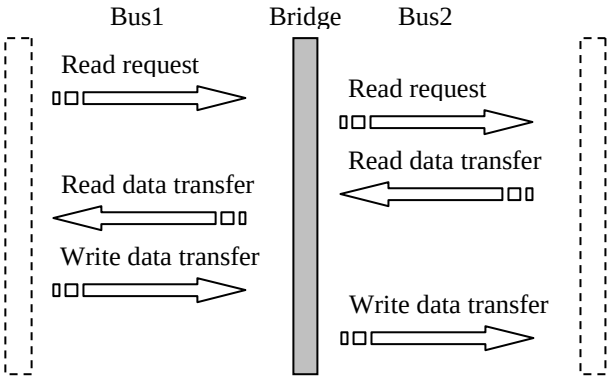

Figure 4. Transactions Across a PCI-X Bridge.

Data integrity and time-out are also checked by the transaction checkers.

4.3.2 Transaction-Ordering Checking

There are several transaction-ordering rules for a PCI-X bridge. In summary, writes should occur in sequence, and reads have no ordering constraints, except that no read should occur before previous writes, unless relaxed ordering flag is set.

Transaction ordering can be checked with the transaction database of the PCI-X buses and a queue. When a transaction that crosses the bridge occurs, the transaction is added to the queue. When a transaction occurs on the target bus, the queue is checked: For a write transaction, it should be the first transaction in the queue. If it is not, then there is an ordering violation. For example, if two writes to address 0x10 and 0x20 cross the bridge targeting on the same bus, then the transactions that occur on the target bus must be writes to address 0x10 and 0x20 in sequence. If write to 0x20 occurs first, there is a write-order violation.

Read order is not as restrictive as write. When a read occurs, all transactions before the read in the queue are checked. If relaxed ordering is not set, there should not be any write transactions before the read, otherwise a read-order violation will be reported. However, read transactions before the current read is allowed.

Whenever a transaction ordering violation is detected, information about the current transaction and the transaction that causes the violation is reported.

5. Experimental Results

5.1 End-to-End Checker

A PCI-X to PCI-X bridge that connects two PCI-X buses is used to perform the transaction-level verification by the end-to-end checker. The end-to-end checker reported that one write ordering violation and five read ordering violations occurred during the simulation.

Simulation environment: AMD MP 1.8G, ModelSim 5.6 on Red Hat 7.3, 512M RAM
Simulation time: 78 seconds
Testbench description: VCD Dump of bus activities for a PCI-X Bridge connecting two buses in PCI-X 266 mode. PCI clock is 133 MHz, 28507 cycles in total, 213803500ps. There are 1203 transactions on bus 1, and 1282 transactions on bus 2.
Example Profile Output:
Format:
("Profile item": bin value) = count : percentage

Example Rule Trigger Profile(bus1):
("XGP1": 32'sh00000001) = 60 : 0.14
("XGP2": 32'sh00000002) = 334 : 0.80
("XGP3": 32'sh00000003) = 1203 : 2.88
("XGP4": 32'sh00000004) = 410 : 0.98
("XGP5": 32'sh00000005) = 1017 : 2.44
("XGP6": 32'sh00000006) = 792 : 1.90

Example EE Checker Profile:
("BUS 2->1, IO R": 7'h48) = 0 : 0.00
("BUS 2->1, IO W": 7'h49) = 0 : 0.00
("BUS 2->1, MEM R": 7'h4a) = 31 : 2.81
("BUS 2->1, MEM W": 7'h4b) = 24 : 2.17

Example Transaction Ordering Profile:
("total tranx across bridge": 32'sh00000000) = 157 : 96.32
("write ordering violation": 32'sh00000001) = 1 : 0.61
("read ordering violation": 32'sh00000002) = 5 : 3.07

5.2 Compaq's PCI-X Core

To test whether the proposed verification environment can be practically used to verify the real-world PCI/PCI-X designs, the Compaq's PCI-X core was incorporated as a design under test (DUT). In order to make the Compaq PCI-X core work as a slave device in the verification environment, some additional modules, such as configuration register, decode block, and target block, are added.

Simulation environment: Intel Pentium III 733MHz, NC-Verilog 3.40 on Red Hat 7.1, 384M RAM
Environment configuration: One PCI-X host as initiator and one Compaq PCI-X core as slave. Bus operates at clock speed 133MHz.
Simulation time: 176 seconds
Applied tests:
- Configuration Read and Write
- Memory Write with different data lengths
- Memory Write at different memory address
- Memory Write Block with various slave decode speed
- Special Cycle test
- Parity error checking

Simulation results: There are 1496 transactions issued by the PCI-X initiator. Totally 20039 clock cycles are simulated in 176 seconds, and the average simulation speed is 114 cps. Some rule violations, including XEP5, XIP22, and XIP24, are reported by the protocol monitor during the simulation.

6. Conclusions

In this paper, a simulation-based verification environment for PCI-X 2.0 bus is proposed. This verification environment contains complete models and tools including master and slave BFMs, protocol monitor, arbiter, coverage analyzer, C-based test generator, and compliance test suits. The integration of C and Verilog makes the test authoring more flexible and efficient by using common high-level languages. The assertion-based protocol monitor also effectively helps the development of BFMs and the protocol verifier. From the experimental results shown in section 5, it can be seen that the running time is reasonable, and the tests with the PCI-X to PCI-X bridge and the Compaq's PCI-X core show that the proposed verification environment is practical for verifying real-world PCI-X designs.

7. References

[1] Thomas L. Anderson, "*Design and Verification IP for PCI and PCI-X*", Applied Computing Conference, Santa Clara, CA, May 2000

[2] Vigyan Singhal and Joe Higgins, "*Compliance Verification for SoC and IP Interfaces*", International Design Conference (DesignCon 2002), January 2002.

[3] Beer I., Ben-David S., Eisner C., Engel Y., Gewirtzman R., and Landver A., "*Establishing PCI compliance using formal verification: a case study*", in Proceedings of the 1995 IEEE Fourteenth Annual International Phoenix Conference on Computers and Communications, Mar 1995, pp. 373 –377

[4] PCI-SIG, PCI-X Protocol Addendum to the PCI Local Bus Specification Revision 2.0, July 29, 2002

[5] Dave Duxstad and Chris Browy, "*PCI Device Compliance Testing Using a Mixed C-Verilog Environment*", International HDL Conference (HDLCon), March 2001

[6] PCI-SIG, PCI-X 1.0a Compliance Checklist.

[7] VCK User's Manual, Version 1.1, Avery Design Systems, 2002