An Interactive Approach to Timing Accurate PCI-X Simulation

Kevin Andryc, Russell Tessier and Patrick Kelly
Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA, 01003

To date, most system-level bus simulation platforms have focused on the functional correctness of individual bus components rather than the full system-level evaluation of multiple components operating simultaneously. In this paper, a new scalable system-level bus simulation environment is described which allows for the evaluation of the PCI-X bus and a series of components. This simulation environment takes advantage of the tight integration of timing-accurate simulation of PCI hardware components with software-level functional modeling to create a fast, accurate system. A series of software techniques is used to allow for time step synchronization across multiple bus components and bus recovery following a transaction. A graphical user interface allows designers to add new components to the system easily, enhancing modularity. The accuracy of the new simulation environment has been validated for a collection of candidate PCI-X systems using an in-circuit PCI-X emulator. The new simulation environment is shown to be accurate to within a percent error of 0.95%, 3.79%, and 2.78% for utilization, efficiency, and bandwidth, respectively.

1 Introduction

The use of system-level buses, such as the Peripheral Component Interconnect (PCI) [4][12] and the Peripheral Component Interconnect Extended (PCI-X) [13] in system designs adds unpredictability to the design process. Accurate and efficient system-level simulation requires careful modeling of cycle-level bus functionality for a variety of design cases. Since simulation is often performed early in the design process, these verification systems are critical to the assessment of available bandwidth and intercomponent latencies. In addition to accuracy, the simulator must execute quickly to allow for the verification of many system cycles. One approach to achieving this efficiency is the use of contemporary software engineering techniques. In general, contemporary PCI and PCI-X simulators do not provide timing-accurate bus simulations for multiple master and slave devices. Most simulators are instead used for function verification of ASICs and system-on-chip designs.

In this paper, a new scalable, timing-accurate bus simulation environment is described which allows for the evaluation of PCI and PCI-X bus systems. The system can simulate the software-level behavior of multiple master and target devices while providing timing accurate hardware-level bus operation. Each master and target device model employs a set of user-adjustable parameters which allow for accurate and flexible simulations. These parameters are then evaluated and used to determine synchronization across components as well as device specific interaction. The configuration of each simulation run is performed via a scalable enterprise-level, web-based graphical user interface, allowing users to quickly create simulations and add and remove components with ease.

2 Background

The Peripheral Component Interconnect, or PCI, bus is a 32 or 64-bit synchronous multiplexed bus which is designed to interconnect high-performance components. Arbitrated data transfers in PCI are always between an *initiator device* (master), one which initiates the data transfer, and a *target device*, the receiver of the data. PCI-X protocol was designed as a revision to the PCI standard and offers several improvements over the original PCI standard including a faster clock (133 MHz versus 66 MHz), lower latency (*Split Transaction* versus *Delayed Transaction*), and improved fault tolerance, amongst other enhancements. By increasing the clock to 133 MHz, a theoretical bandwidth of 1.06 GB/s can be achieved using a 64-bit bus path in contrast to the 532 MB/s offered by PCI. In addition to offering higher performance, PCI-X is generally backwards compatible to PCI. However, while both PCI and PCI-X devices may be intermixed, the bus speed is determined by the slowest device [13].

2.1 Related Work

An important aspect of this work is the more accurate modeling of PCI/PCI-X bus activities, including master and slave recovery periods following transfers. Schönberg [14] developed mathematical models to describe bandwidth and latency on a bus with arbitrary devices. Specifically, a descriptor was defined as:

$$D = (s, d, r) \tag{1}$$

where s defines a *non-data phase*, d defines a *data phase*, and r defines the *recovery phase*. The recovery phase is not a physical bus phase and only guarantees that devices do not access the bus during this time. Unlike Schönberg, our approach uses calculated values for the recovery phase during simulation to enhance modeling accuracy.

Finkelstien [3] developed a C++ based PCI/PCI-X simulator that uses adjustable performance parameters for both target and master devices. The approach improves the PCI modeling efficiency and accuracy by eliminating wait states initiated by the target. The simulation environment is command line driven with devices and performance parameters embedded within the code, complicating ease-of-use. In addition, the timing accuracy of the simulator is not numerically validated versus a physical platform, although a bus analyzer is used for logical verification. Our simulator uses similar performance parameters from the PCI specifications, as well as two additional parameters: injection rate and recovery period.

3 Implementation

3.1 Architecture

The simulator system architecture is shown in Figure 1. The web application architecture is designed using the Java 2 Enterprise Edition framework which provides a robust and scalable architecture [2]. The web application dynamically creates the graphical user interface using data that was previously stored in the database by the user.

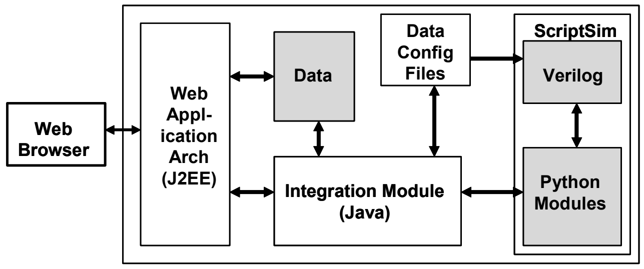

Figure 1: PCI Simulator System Architecture

The *integration module* uses information about the requested simulation to configure device files. These device files contain the information describing the device behavior on the PCI/PCI-X bus including configuration information, performance information and listings of every transaction each device must make on the bus. In addition, the integration module dynamically creates the Verilog code which describes what devices are attached to the bus as well as the bus width and speed.

The web *application architecture* uses the classic model-view-controller (MVC) architecture [1] with internal designs following J2EE patterns, as shown in Figure 2. The MVC architecture allows for the separation of the business logic from the control and presentation logic providing a scalable and maintainable infrastructure.

A client interacts with a single view which dynamically updates content depending on the selected action. A client action triggers an asynchronous HTTP request using the `XMLHttpRequest` object created in JavaScript. The Controller servlet receives the request and, using dynamic binding, retrieves one of the three `Action` objects. The `Action` object extracts the data from the request and invokes the proper method in the `SimulationFacade` which performs the necessary work. This action involves the retrieval of business objects represented as entity beans. The entity beans use container-managed persistence so data access logic is handled by the EJB container. The business objects are passed back to the `Action` object via the `SimulationFacade` to a handler method. The data is then encapsulated within XML tags and sent back to the view using an HTTP response. Within the view, a callback method extracts the XML data and presents it to the client.

The GUI allows a user to configure a simulation by adding devices and configuring them using the parameters described subsequently in Sections 3.2.1 and 3.2.2.

ScriptSim [15] is an open source software tool that integrates Verilog with scripts such as Python and Perl. ScriptSim allows Verilog to dynamically create script processes and communicate with those processes by passing Verilog data types to the scripts. In turn, the scripts have access to any Verilog data used in the design and can perform the equivalent of any Verilog assign including: blocking, non-blocking, continuous assign, or force. In Figure 3, a block level view of the ScriptSim architecture is provided.

To communicate data between the various Verilog modules and the scripts, ScriptSim uses Verilog's *Programming Language Interface* (PLI). The PLI interface allows user supplied C programs to interact with the simulation environment and access Verilog's internal data structures. One major issue with PLI is that when Verilog calls a PLI routine, execution is suspended until the C subroutine is terminated. This action causes problems since the software may generate multiple access cycles, each of which must be handled by the Verilog module. To solve this problem, ScriptSim runs the software and simulator as two different processes, communicating through Unix sockets. The scripts then use the C programs as a proxy to send and receive data.

ScriptSim simulates all functionality of the PCI Local Specification Version 2.2 protocol; however, we have modified ScriptSim to allow for simulation of the PCI-X bus. ScriptSim uses several Python scripts to simulate the PCI bus. The PCI bus is comprised of a bus monitor (`Monitor.py`), arbiter (`Arb.py`), and one or more agents (`Agent.py`). The agents represent the devices on the bus and use command files (`pci_cmds#`) to configure and generate activity.

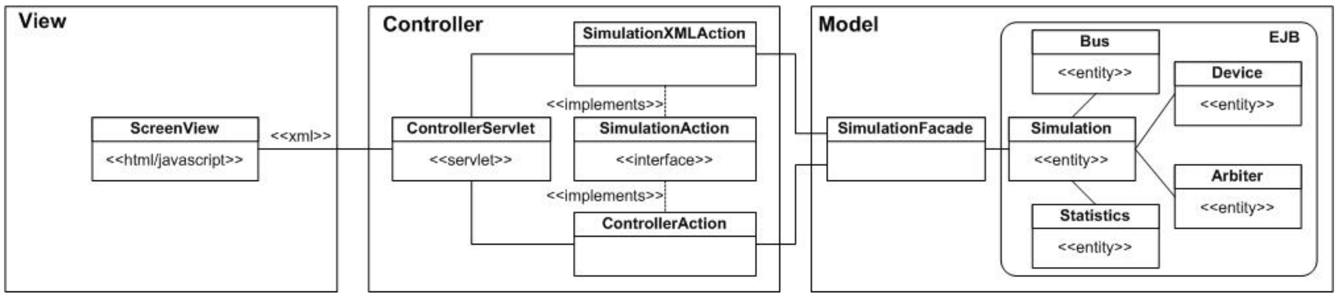

Figure 2: MVC Architecture of the PCI Simulator

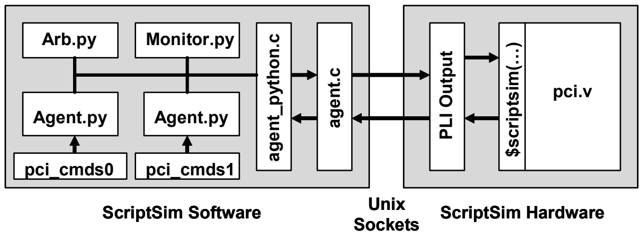

Figure 3: ScriptSim Architecture

The integration module uses the data from the database to dynamically create the Verilog and configuration files which are fed into ScriptSim. Each configuration file describes a device including the transactions it will perform on the bus along with its corresponding performance characteristics, as described by the device descriptor. The integration module takes the information from the database, dynamically creates the devices using the parameters described in Section 3.2 and 3.2, and then triggers ScriptSim to run the simulation. The results are then fed back to the user in the form of text and graphs and stored in the database for future viewing.

3.2 Modeling Approach

In this section, device descriptors for both master and target devices are described. The descriptor consists of several previously-defined user adjustable performance parameters [3][14], data from PCI specifications, and analyzed bus data. A master device descriptor consists of eight performance parameters. The read/write ratio, burst length, and transaction parameters values (taken from [3]) and latency parameters (taken from [14] and [12]) provide consistency with earlier models. Additionally, injection rate and recovery period are used to increase modeling accuracy.

The injection rate is the amount of data placed into the system per unit time. In our simulator, the injection rate is strictly defined as the required bandwidth of a device. More complex methods, such as queuing models based on Poisson processes [9], may provide a more fine-grained approach at the cost of additional computation. As described in Section 5, the use of required bandwidth was found to provide sufficient modeling fidelity.

Parameter	Description
Injection Rate	The amount of bandwidth required by the device.
Read/Write Ratio	Describes the ratio between the number of reads and the number of writes.
Burst Length	The number of data words a master can send contiguously. A value of 1 places the device in normal mode where only one data word is transferred per transaction.
Initial Wait States	The number of wait states from the assertion of FRAME# until the first word is ready to be sent or received.
Subsequent Wait States	The number of wait states a master waits before sending or receiving the next data word, after the initial data word.
Recovery Period	The minimum number of cycles the master must wait before requesting the bus.
Master Latency Timer	The minimum number of clock cycles the master is allowed to retain ownership of the bus. This value is decremented on each clock cycle after initiating a transaction.
Transactions	The number of transactions the master is allowed to initiate.

Table 1: Master Device Parameter Summary

The recovery period for PCI devices was introduced in Section 2.2. In a shared bus environment, each device will incur an arbitration latency. This latency is a function of the arbitration algorithm, the sequence in which masters are granted access, and the amount of time each is allowed access on the bus. Each device must therefore provide sufficient buffer space to match the injection or consumption rate of data that can be moved across the bus. Without loss of generality, we define $c_{req,n}$ as the first clock cycle the device is able to request the bus for transaction n. The minimum amount of time a device must wait before requesting the bus for transaction $n+1$, regardless of whether transaction n has completed or not, is given by:

$$recovery_{n+1} = \left\lceil \frac{bytes_{n+1}}{bandwidth_{device} \times clock\_period_{pci}} \right\rceil \quad (2)$$

where $bytes_{n+1}$ is the number of bytes expected to be sent during transaction $n+1$, which is a function of the burst length. Therefore, transaction $n+1$ can start on clock cycle:

$$c_{req,n+1} = c_{req,n} + recovery_{n+1} \quad (3)$$

The specific performance parameters associated with a master device descriptor are listed in Table 1.

Parameter	Description
Decode Speed	The number of clock cycles required to claim a transaction. Both PCI and PCI-X offer four decode speeds: Fast, Medium, Slow, and Sub. Sub, or subtractive decode, only responds to decodes ignored by other targets.
Burst Length	The maximum number of data words a target can send or receive.
Initial Wait States	The number of wait states from the assertion of **FRAME#** until the first word is ready to be sent or received.
Subsequent Wait States	The number of wait states a target waits before sending or receiving the next data word, after the initial data word.
Initial Retry Threshold	The number of wait states allowed by the target before a retry is generated. This only applies to the first data word in a burst and is limited to 16 per the PCI 2.1 specification.
Subsequent Retry Threshold	This is the same as the initial retry threshold except it applies to burst cycles and is limited to 8 per the PCI 2.1 specification.

Table 2: Target Device Parameter Summary

The target device descriptor shares many performance parameters with a master device with a few exceptions. Burst length and subsequent retry threshold (taken from [3]), and other parameters (taken from [3] and [12]) define the descriptor. Performance parameters associated with a target device descriptor are summarized in Table 2.

4 Experimental Design

To verify the accuracy of the simulator, we gathered PCI and PCI-X cycle snapshots from a variety of configurations generated by Vanguard's VMetro Bus Analyzer [16]. Statistical information was then extracted and compared to simulated values.

Our target PC architecture splits the memory controller and I/O controller into separate chips, the Northbridge and Southbridge. The Northbridge [8], or memory controller hub, handles high speed communication between devices such as the CPU, RAM, and AGP via the PCI Express bus. The Southbridge, or I/O controller hub, handles less performance critical I/O devices via the PCI or PCI-X bus, or LPC bus. In our experiments, the PCI/PCI-X bus contains one or more PCI/PCI-X devices and the VMetro Bus Analyzer, which is used to passively monitor the bus. Data collected by the analyzer are sent to the PC via a USB port and are analyzed using VMetro's BusView software.

Table 3 outlines the various configurations used for testing. Each test configuration was implemented on a physical system and assessed with the bus analyzer. Additionally, simulation was performed for the same system configuration. The following statistics are determined for both cases:

- *Utilization*: Indicates how frequently the bus is being used. The value is calculated by dividing the number of transactions (i.e.: the number of cycles where FRAME# and/or IRDY# are active) by the total number of cycles.
- *Efficiency*: Measures how efficient the system is at transferring data by determining the duration of data transfers versus duration of transactions. The value is calculated by dividing the data total percentage (the number of data cycles divided by the total number of cycles) by the utilization.
- *Bandwidth*: Amount of data sent over the bus per unit time. The value is calculated by dividing the total number of bytes sent by the total time.

Trial	Bus			PCI / PCI-X Devices	
	Type	Speed MHz	Width	Master	Target
1	PCI	33	32-bit	1 PCI	1 PCI
2	PCI	33	32-bit	3 PCI	2 PCI
3	PCI-X	133	64-bit	1 PCI-X	1 PCI-X

Table 3: Simulation Configurations

5 Results

5.1 Experiment 1: Single PCI Master

Our first system configuration consisted of a 32-bit, 33 MHz PCI bus using Intel's 82801DB I/O controller hub [7]. A single bus master device, Foresight Imaging's PCI frame grabber, can inject data onto the bus at a rate of 110 MB/s (calculated using a resolution of 1280 by 1024 with 24 bit depth at 28 frames per second) and exhibits performance characteristics described in Table 4. The frame grabber writes to and reads from system memory, acting as the target device, via the host bridge. The target device incurs no initial wait states on writes and exhibits an average of 15-34 initial wait states on reads. It can sustain long bursts (up to a 4K page boundary) with no subsequent wait states. The performance characteristics of system memory are described in Table 4. The frame grabber performs 34 burst writes at the maximum speed (i.e.: no subsequent wait states) for each transaction until it hits a cache line which falls on a 4KB page boundary, in which case a read will occur. A round robin arbiter is modeled after the PCI scheduler found in Intel's 82801 ICH with a MTT (multi-transaction timer) set to 20.

Device	Frame grabber	Host bridge
PCI Device Type	32-bit Master	32 bit Target
Injection Rate	110 MB/s	N/A
Read/Write Ratio	Perform write until 4K boundary and then 1 read transaction	N/A
Burst Length	Read: 4 Write: 34	4K page boundary
Initial Wait States	Read: 0 Write: 0	Read: Random (15-24) Write: 0
Subsequent Wait States	Read: 0 Write: 0	Read: 0 Write: 0
	Master Latency: 64	Decode Speed: Medium
	Recovery Period: Calculated using Equation (2)	Initial Retry Threshold: 16
	Transaction Count: 200	Subsequent Retry Thresh: 8

Table 4: Experiment 1: Master and Slave Device

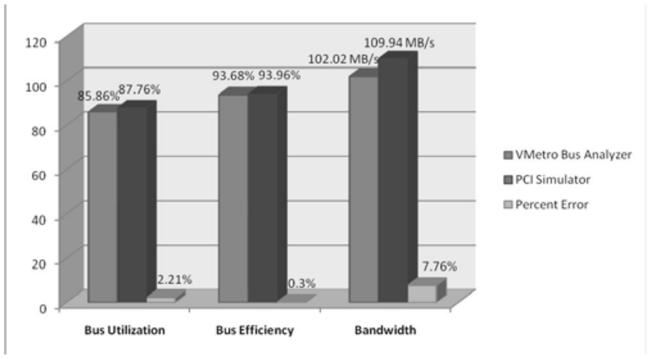

Figure 4: VMetro Analyzer vs. PCI Simulator Results for Experiment 1

Figure 4 shows the results of our simulation versus the statistics gathered by the VMetro Bus Analyzer. Minimal errors are incurred with respect to bus utilization and efficiency, 2.21% and 0.30% respectively. However, there is a 7.76% difference in the bandwidth, with the simulator producing a larger bandwidth value. We believe that this is due to our optimistic calculation of the recovery period. Recall that our definition of recovery period is the minimum amount of time between transaction requests, calculated using Equations 2 and 3. Generally, this value is the amount of time required to refill I/O buffers, although, there may be other device specific factors involved which may delay requests.

5.2 Experiment 2: Three PCI Masters

The system for the second experiment consists of a 32-bit, 33 MHz PCI bus using Intel's 82801DB I/O controller hub and three PCI master devices. In addition to the frame grabber used in Experiment 1, a camera and PCI interface which can inject live data at approximately 110 MB/s onto the bus is included. Their parameters are shown in Table 5. The camera performs burst writes for 66 cycles at the full data rate (i.e. no wait states). While the device's master latency timer is set at 64, the PCI specification allows two extra cycles before a device must complete a transaction. After each burst write, a memory read is performed followed by an I/O read of 4 cycles and 1 cycle, respectively. Periodically, a CPU read of the camera's PCI interface is performed and injects approximately 1 MB/s of data. Reads and writes to memory are performed via the host bridge. However, I/O reads via the host bridge now incur initial wait states of between 5 and 8 cycles (Table 6). A round robin arbiter with the MTT set to 20 cycles is used.

Figure 5 shows the results of our simulation versus the statistics gathered by the VMetro Bus Analyzer. A significant bandwidth error decrease (87%) with respect to the Experiment 1 bandwidth error is noted. We recall from Experiment 1 that our optimistic calculation of the recovery period causes our simulator to produce a higher bandwidth.

Device	Camera	CPU
Device Type	32-bit PCI Master	32-bit PCI Master
Injection Rate	110 MB/s	1 MB/s
Read/Write Ratio	Repeat memory write followed by read and then I/O read	All memory reads
Burst Length	Read: 4 Write: 66	Read: 1 Write: 0
Initial Wait States	Read: Random (13-15) Write: 0	Read: 0 Write: 0
Subsequent Wait States	Read: 0 Write: 0	Read: 0 Write: 0
Master Latency Timer	64	64
Recovery Period	Calculated using Equation (2)	Calculated using Equation (2)
Transaction Count	200	200

Table 5: Experiment 2: Master Devices

Device	Host Bridge (I/O Device)
Device Type	32-bit PCI Target
Decode Speed	Medium
Burst Length	1
Initial Wait States	Read: Random (5-8) Write: 0
Subsequent Wait States	Read: 0 Write: 0
Initial Retry Threshold	16
Subsequent Retry Threshold	8

Table 6: Experiment 2: I/O Target Device

However, in this case the recovery period is hidden by the fact that another device is transferring data on the bus. Consider a simple example with two devices on a bus, D_1 and D_2, where D_1 has been granted access to the bus and D_2 is beginning its recovery period. If the recovery period for D_2 is less than or equal to the bus access time required by D_1 to complete its transfer, then D_2 will be able to immediately start after D_1 completes.

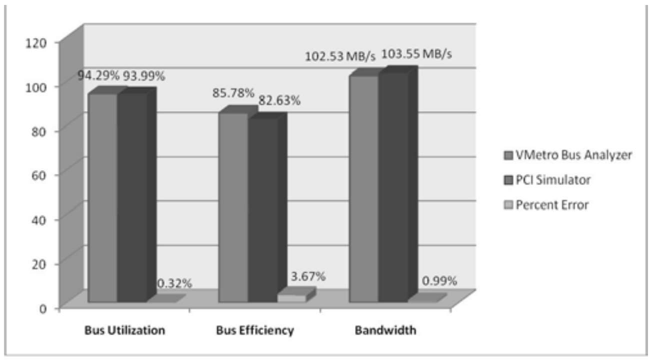

Figure 5: VMetro Analyzer vs. PCI Simulator Results for Experiment 2.

5.3 Experiment 3: Single PCI-X Master

Our final simulated system consists of a 64-bit, 133 MHz PCI-X bus using Intel's 6700PXH 64-bit PCI hub [6]. A single bus master device, a Nallatech 64-bit 133 MHz PCI-X FPGA computing motherboard [11] can inject data onto the bus at a rate of 192 MB/s and exhibits performance characteristics as described in Table 7. An FPGA can perform burst writes to system memory, acting as the target device, via the host bridge. The target device incurs no initial wait states on writes and can sustain long bursts (up to a 4K page boundary) with no subsequent wait states. The characteristics of system memory are the same as the host bridge in Table 4, except that the device is a 64-bit PCI-X target. The FPGA will perform 1,024 burst writes with no subsequent wait states for each transaction until it hits a cache line, which falls on a 4KB page boundary.

As shown in Figure 6, modest errors are incurred with respect to bus utilization, efficiency, and bandwidth, 0.95%, 3.79%, and 2.78%, respectively. As discussed in Section 3, the simulator calculates a minimum recovery period, thus leading to a higher bandwidth. However, the simulator determines a slightly higher efficiency than the actual system. For completeness, we calculate the bus efficiency using the following formulas:

$$efficiency = \frac{percent_{data}}{percent_{utilization}} \tag{4}$$

The data percentage is calculated by:

$$percent_{data} = \frac{clock\_cycles_{data}}{clock\_cycles_{total}} \tag{5}$$

Device	Nallatech FPGA
Device Type	64-bit PCI-X Master
Injection Rate	192 MB/s
Read/Write Ratio	Perform write until 4K boundary and then 1 read transaction
Burst Length	Read: 1024 Write: 0
Initial Wait States	Read: 0 Write: 0
Subsequent Wait States	Read: 0 Write: 0
Master Lat. Timer	1024
Recovery Period	Calculated using Equation (2)
Transaction Count	100

Table 7: Nallatech FPGA Master Device

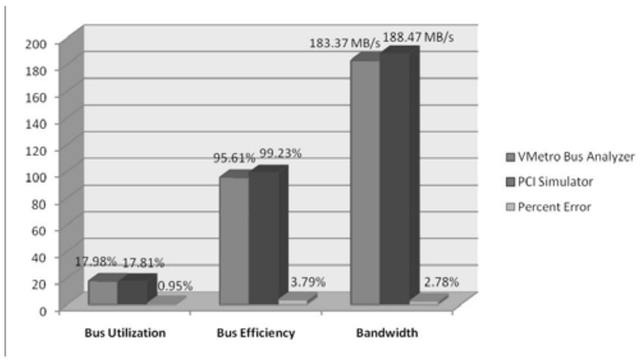

Figure 6: VMetro Analyzer vs. PCI Simulator Results for Experiment 3.

6 Conclusion

In this work a new approach to PCI simulation using accurate bus parameters and an interactive simulation environment has been developed. A web-based graphical user interface is used which provides users with a high level of configurability to model advanced bus systems. The architecture and design of the system employ well known software engineering techniques that ensure scalability. By using well known design patterns, we promote reuse while decreasing overall design time.

In order to achieve a high-level of accuracy in our simulations, we developed techniques that allow devices to exhibit individualized behavior on the bus. This assessment was done by decomposing devices into sets of performance parameters that make up a device descriptor. Two unique simulation parameters, injection rate and recovery period, are introduced. The parameters make it possible to specify how quickly a device can place data on a bus and the minimum amount of time is needed before a subsequent transaction can start, once a transaction has started. Experimental results show that a high level of accuracy (a few percent difference in the worst case) is achieved for bus utilization, efficiency, and bandwidth versus system data captured by a commercial bus analyzer.[1]

References

[1] A. Leff, J.T Rayfield, "Web-application development using the Model/View/Controller design pattern", in *Proceedings of the Fifth IEEE International Enterprise Distributed Object Computing Conference*, pp. 118-127, Sept. 2001.

[2] D. Alur, J. Crupi, and D. Malks, Core J2EE Patterns. Upper Saddle River, NJ: Prentice Hall, 2nd ed., 2003.

[3] E. Finkelstein, "Design and Implementation of PCI Bus Based Systems." Master's Thesis, Tel Aviv University, 1997.

[4] E. Solari and G. Willse, *PCI & PCI-X Hardware and Software Architecture & Design*. Research Tech Inc., 6th ed., 2005.

[5] Intel Corp., Santa Clara, *Intel 440FX PCISET 82441FX PCI and Memory Controller (PMC) and 82442FX Data Bus Accelerator (DBX)*, May 1996. Order # 290549-001.

[6] Intel Corp., Santa Clara, *Intel 6700PXH 64-bit PCI Hub,* July 2004. Order # 302628-002.

[7] Intel Corp., Santa Clara, *Intel 82801DB I/O Controller Hub 4 (ICH4),* May 2002. Order # 290744-001.

[8] J.L. Hennessy, D.A. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufmann, Menlo Park, California, 2003.

[9] L.M. Ni, C.-F.E Wu, "Design tradeoffs for process scheduling in shared memory multiprocessor systems", in *IEEE Transactions on Software Engineering*, vol. 15, issue 3, pp. 327-334, Mar. 1989.

[10] M. Fowler and K. Scott, *UML Distilled Second Edition*. Reading, MA: Addison-Wesley, Jan. 2000.

[11] Nallatech. *http://www.nallatech.com/.*

[12] PCI Special Interest Group, *PCI Local Bus Specification Revision 2.2*, Dec. 1998.

[13] PCI Special Interest Group, *PCI-X Addendum to the PCI Local Bus Specification*, rev. 1.0b, July 2002.

[14] S. Schönberg, "Using PCI-Bus Systems in Real-Time Environments." PhD Thesis, Technische Universität Dresden, 2002.

[15] ScriptSim. *http://www.nelsim.com/scriptsim/intro.html.*

[16] VMETRO. *http://www.vmetro.com/.*

[1] Research sponsored by Kollmorgen Electro-Optical.