
ELSEVIER

Journal of Systems Architecture 47 (2002) 807–819

JOURNAL OF SYSTEMS ARCHITECTURE

www.elsevier.com/locate/sysarc

A PCI bus simulation framework and some simulation results on PCI standard 2.1 latency limitations

Ehud Finkelstein, Shlomo Weiss *

Department of Electrical Engineering—Systems, Tel Aviv University, Tel Aviv 69978, Israel

Abstract

We describe a simulation environment that allows us to simulate the standard peripheral component interconnect (PCI) bus protocol, as well as modified PCI protocols. While there are standard benchmarks (such as the SPEC [IEEE Comput. 33 (7) (2000) 28] benchmarks) available for processor simulation, database system simulation, and now even for simulating embedded systems (from EDN Embedded Microprocessor Benchmarking Consortium, EEMBC, http://www.eembc.org), there are no standard benchmarks for simulating computer buses in general and specifically, for simulating the PCI bus. To address this problem we describe a methodology for gathering information about the PCI traffic from a real system, and to use this information in order to generate PCI cycles that drive the simulator for both standard and modified PCI protocols. Finally, we use the simulation environment to run experiments with various parameters of the standard PCI protocols, and an extension that involves transferring a hint about the expected latency on the data bus at the time the target ends the current burst transaction.

Keywords: Computer architecture; Computer buses; I/O systems; Peripherals; PCI

1. Introduction

In addition to the CPU, PC systems include other essential components that are often neglected in computer architecture research and textbooks; two notable exceptions are [1,2]. One of these essential components is a bus or multiple buses, which connect the various subsystems of a computer system [3]. Since its first release by the peripheral component interconnect (PCI) Special Interest Group, an organization formed in 1992 to develop the PCI local bus Specification, the PCI bus [4–10] has become an industry standard, implemented in almost all PC systems as well as in some workstations [11] and servers [12]. More recently, PCI buses have been also reconfigured for industrial applications and embedded systems [13]. The PCI bus performance has a critical impact on the overall system performance in many applications [14–19].

The advantages of simulation are well known [20], and there are many simulation environments available to the computer architecture research community for various computer systems and subsystems. Yet, despite the predominance of PCI, to our best knowledge no simulation environment was available in the public domain for simulating the

* Corresponding author. Tel.: +972-3-6407400; fax: +972-3-6407095.

E-mail address: weiss@eng.tau.ac.il (S. Weiss).

1383-7621/02/$ - see front matter
PII: S1383-7621(01)00033-9

PCI bus system before the work presented in this paper. The first part of this work describes the implementation of a simulation framework that allows us to simulate the standard PCI bus protocol, as well as modified PCI bus protocols.

Next we describe a methodology for gathering information about the PCI traffic from a real system, and to use this information in order to generate PCI cycles that drive the simulator for both standard and modified PCI protocols. Finally, using the simulation environment and the data gathered about the PCI traffic as an input to it, we provide some results on PCI bus latency limitations and consider a modified PCI protocol that addresses the latency problem.

1.1. The latency problem

A transfer on the PCI bus takes place between a *master* and a *target* (see [9], for example, for a comprehensive description of the PCI protocol and bus signals). The master is capable of driving the control signals needed to perform the transfer. A target is unable to control the bus but must be able to determine that it has been selected and to participate in the transfer by either receiving or supplying data.

PCI revision 2.0 compliant target devices had to hold the bus until they could deliver read data, no matter how long it took. No means were available to make use of the bus during this latency time, nor was the target latency time limited. In theory, the target could retry the master, but since the master was not required to retry the transaction, the only way to guarantee data delivery from a long latency target device was to hold the bus as long as necessary.

In order to solve the problem, the PCI revision 2.1 standard was modified in a few subtle ways:

1. Target latency was limited to 16 cycles on the first word, and eight cycles on any subsequent word in a burst. Any target requiring a longer latency, was required to retry the master, by terminating the current cycle without data.
2. A PCI 2.1 Master is now *required* to re-issue a request if a target is disconnected without delivering data. The master must retry the request until the target is ready.
3. After disconnecting, the target is required to refuse (also by disconnecting without data) any other request other than the previous word. This ensures that the read order is preserved.

This solution allows other masters to compete for the bus between the original request and the target reply, but it has two problems:

1. The Master has to poll the target until it receives the data, wasting bus bandwidth which can be used by other bus masters.
2. The master will always poll the target, and since the polling may happen every N cycles, it will never get the data from the target as soon as the data is ready, but as soon as the master retries after the data has been ready. This implies an average waste of $N/2$ cycles on every delayed transaction.

We address these problems in Section 3 by considering modifications of the PCI bus protocol. Although the recently released PCI-X v1.0 addendum [21] to the PCI bus specification supports true split transactions, many systems still use earlier versions of the PCI, with the latency problems described above.

1.2. Systems with multiple PCI buses

The latency limitations described above are especially critical in multiple bus systems. Although in this work we do not explicitly simulate multiple bus systems, such a system is the primary application of the solution proposed in Section 3, and therefore we briefly describe it here.

While currently most PCI systems are single bus systems, a PCI to PCI Bridge Architecture Specification [22] was released by the PCI SIG as early as 1994, in anticipation of the development of multiple bus systems. As PCI systems grow to multiple bus masters, PCI to PCI bridge chips are used to link multiple PCI bus segments.

Fig. 1 illustrates a simple application for a PCI to PCI bridge, with two PCI buses. The primary interface of the PCI to PCI bridge is connected to PCI bus 1, which communicates with the CPU and with the memory system through the CPU–PCI