

Controlling the Influence of PCI DMA Transfers on Worst Case Execution Times of Real-Time Software *

Jürgen Stohr Alexander von Bülow Georg Färber

Institute for Real-Time Computer Systems

Prof. Dr.-Ing. Georg Färber

Technische Universität München, Germany

{Juergen.Stohr,Alexander.Buelow,Georg.Faerber}@rcs.ei.tum.de

Abstract

The PCI Local Bus is used in all general purpose computer systems. Peripheral devices connected to this bus may perform transactions autonomously. If a processor accesses the main memory or performs an I/O instruction, the execution time of these operations depends on the working load of the PCI bus and of the communication protocols being used by the chip set. In this paper the influence of the PCI Local Bus on real-time software is demonstrated. A method is presented reducing these impacts of the PCI Local Bus on the execution time of real-time software. Thus accesses to PCI peripherals from real-time tasks behave more deterministically.

1. Introduction

When determining the worst-case execution time of real-time software, all the underlying hardware has to be taken into account. Major reasons for varying execution times of software are the caches and the TLBs as the costs of a cache miss are immense. If there is a miss the execution time of loading a cache line from the main memory into the processor depends on the transactions being performed by the PCI peripherals. In addition, if a real-time task wants to perform I/O and therefore has to access a peripheral device directly, the chip set also should not be busy if deterministic computation times are favored.

In most cases the behavior of the chip set is transparent to software. It interconnects the processors, the main memory and the peripheral devices. An essential part of the chip set of general purpose computers is the PCI Local Bus which is used to connect the peripheral devices to the host. As these peripheral devices can be programmed to perform accesses to

the main memory autonomously by processing DMA transfers, there is a steady influence on the execution time of software.

In this paper the impacts of the PCI Local Bus on the worst-case execution time of real-time software are examined. A method is presented which can be used to postpone the DMA transfers of PCI devices in order to get more deterministic execution times of real-time software. This method can be used if a general purpose and a real-time operating system are running in parallel on the same machine and real-time and non real-time devices are connected to PCI.

This paper is organized as follows. Section 2 gives a survey of the PCI Local Bus being used in real-time systems. In section 3 some important facts of PCI are explained. Our method making the PCI Local Bus behaving more deterministically is described in section 4. The advantages of this method are demonstrated in section 5. The paper is summarized in section 6.

2. Related Work

In real-time systems, the PCI Local Bus is used to interconnect the various components. Examples are the RAPID [2] [6] and the SARA [3] project. However, only a few publications deal with PCI: In [4] PCI connects a reconfigurable computing device to its hosts processor. In this paper, some performance measurements concerning PCI are done. Baumgartl and Härtig discuss in [1] PCI busmastering DMA. In contrast to the conclusion of this paper, they noticed that to their knowledge it is impossible to give timing guarantees for DMA operations involving the PCI bus. In [7] the impact of PCI-Bus load on real-time applications is evaluated. A slowdown factor is defined to describe these impacts formally.

When designing and validating a computer system for hard real-time usage the worst case execution time (WCET) of software has to be known. There are many components of the system that have an influence on the WCET, for example the architecture of the CPU and the design of the caches. The chip set and its buses and communication protocols affect the exe-

*The work presented in this paper is supported by the *Deutsche Forschungsgemeinschaft* as part of a research programme on "Real-time with Commercial Off-the-Shelf Multiprocessor Systems" under Grant Fa 109/15-1.

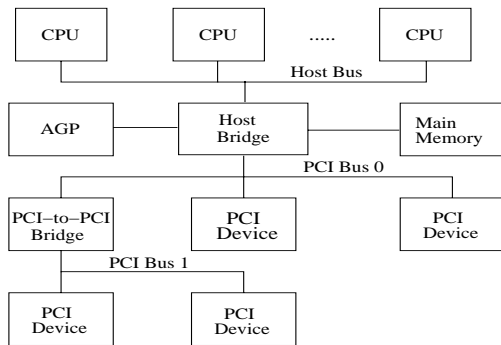


Figure 1: System Architecture

duction time, too. There exist two different approaches when evaluating the WCET: the modelling and the measurement based approach. The pros and cons of each approach are discussed in [5]. As the specifications of processor and chip set are often not available for public use, we have chosen the measurement based approach to start our work with.

3. PCI Local Bus

In state of the art computer systems, the Host Bridge interconnects the CPUs, the Advanced Graphics Adapter (AGP), the main memory and the PCI peripherals. Figure 1 illustrates a simplified system architecture: The Host Bridge connects the CPUs to the Host Bridge and the PCI Local Buses are used to connect the peripheral devices to the Host Bridge.

3.1. Functionality

The PCI Local Bus is a multi-master bus. Every device is allowed to become initiator. The initiator, or bus master, initiates a transfer. The target, or slave, is the device currently addressed by the initiator for the purpose of performing a data transfer. PCI devices can access the bus autonomously without the aid of a CPU. In order to avoid collisions, each initiator has to request the bus from the PCI bus arbiter before performing any transfers.

Usually the arbiter is integrated into the PCI chip set; specifically, it is typically integrated into the Host Bridge. As the PCI specification does not define the scheme to be used by the PCI bus arbiter, the order the devices are accessing the bus depends on the chip set being used. The PCI specification only states that the arbiter is required to implement a fairness algorithm to avoid deadlocks.

As the Host Bridge acts to the PCI bus as a PCI device, it has to request the PCI bus from the arbiter like any other device before becoming initiator. If a peripheral device wants to access the main memory the Host Bridge is the target.

A PCI-to-PCI bridge provides a bridge from one PCI bus to another. It works as a traffic coordinator between the two

buses. It monitors the transactions that are initiated on the two PCI buses and decides whether or not to pass the transaction through the opposite PCI bus.

3.2. Burst Transfers

When performing a transfer, a peripheral device first has to request for bus ownership as mentioned above. The transfer itself is consisting of a single address phase followed by two or more data phases. The start address and the transaction type are issued during the address phase. The target device latches the start address into its address counter which is incremented from one data phase to the next one. This kind of data transfer is called a *DMA burst transfer*.

If a bus master acquires ownership of the PCI bus, it initiates a transaction. The two most important types of transactions are listed below:

memory transactions A device accesses the memory of another PCI device. If the target addresses reside in the main memory, the Host Bridge serves as target.

I/O transactions These transactions are used to access the command and status registers of the PCI devices. Normally, I/O transactions are generated by the Host Bridge.

4. Impact of the PCI-Bus

If a PCI peripheral device performs a burst transfer accessing the main memory, the execution of real-time software can be delayed in two ways:

- If the CPU executing a real-time task has to access the main memory, these accesses are affected by parallel transfers of peripheral devices. This happens if a peripheral device initiates a DMA burst transfer accessing the main memory just before the access to the main memory of a CPU is initiated.
- A real-time task wants to perform I/O, e.g. writing data to a hard disk, and therefore has to access a peripheral device. If there is a high workload on the PCI bus, the I/O transactions may be delayed. This latency depends on the arbitration scheme being used by the bus arbiter.

As the policy of the PCI arbiter depends on the chip set, it cannot be said in which order the peripheral devices are allowed to access the bus. If some devices have got pending requests, the arbiter chooses one of them. If the access to the bus is granted to a certain device, it is allowed to perform its transfers for a certain span of time, which is specified by the *Latency Timer*.

Master Enable Bit The execution time of real-time software when accessing a PCI peripheral varies depending on the workload on the PCI bus. But there is a way to configure the

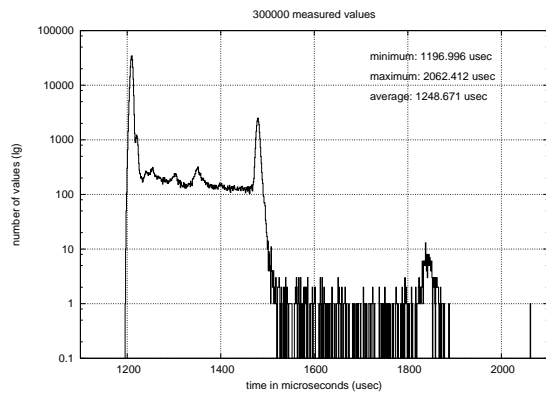


Figure 2: Accesses of a CPU to main memory with concurrent PCI activity

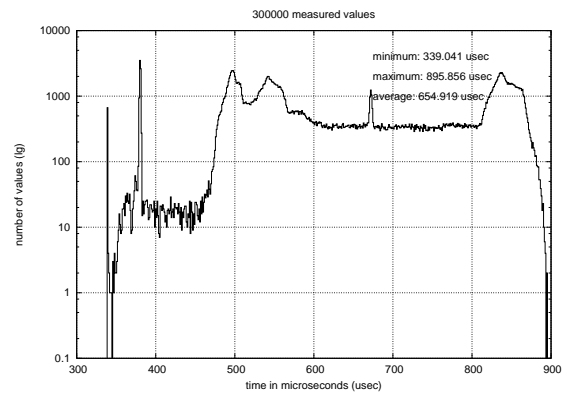


Figure 4: Accesses of a CPU to a PCI device with concurrent PCI activity

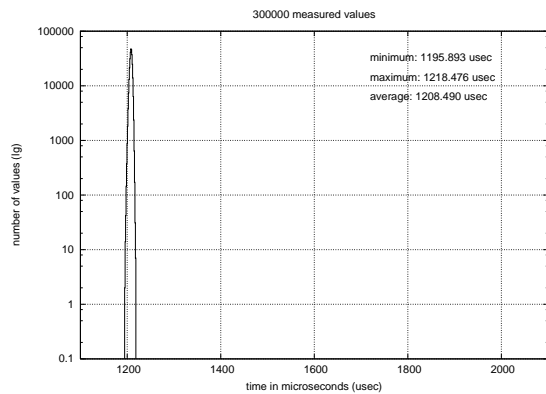


Figure 3: Accesses of a CPU to main memory without concurrent PCI activity

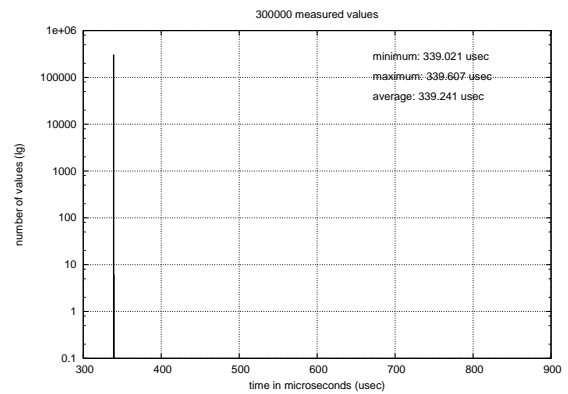


Figure 5: Accesses of a CPU to a PCI device without concurrent PCI activity

PCI Local Bus to behave more deterministically: Each peripheral device can be prevented from becoming initiator by clearing the *Master Enable Bit* which is defined in the *PCI Command Register*. If this bit is cleared on every peripheral device which may become initiator during certain critical sections, the PCI Local Bus can be used by the Host Bridge exclusively. Thus a real-time task performing transactions across the PCI bus cannot be delayed.

The time needed to clear every Master Enable Bit depends on the number of peripheral devices connected to the bus, the activity of each device and of the arbitration algorithm. Before performing the I/O transactions needed to clear the Master Enable Bit, the Host Bridge has to arbitrate for the bus. This latency depends on the peripherals able to perform transactions. If the number of devices being able to become initiator decreases, the time needed to clear the Master Enable Bit of a remaining peripheral decreases, too.

PCI-to-PCI bridges If PCI-to-PCI bridges are used, all peripherals not needed in real-time context should be put behind this bridge if possible. Thus only the Master Enable Bit of the bridge has to be cleared. Then the bridge ignores all memory and I/O transactions detected on the secondary side. In order to minimize the time needed to clear the Master Enable Bit of each relevant PCI device, a few guidelines can be stated:

- The devices causing the heaviest workload should be disabled first.
- All non-RT relevant devices should be grouped behind a bridge, thus only the bridge has to be disabled.
- The devices needed in RT context should be plugged into the PCI bus nearest to the Host Bridge.
- Only the devices able to initiate DMA burst transfers should be disabled. The Master Enable Bits of periph-

eral devices which are not used by the operating system or which only act as slaves need not to be cleared.

5. Results

We have performed some measurements in order to study the effects when the Master Enable Bits are cleared:

Accesses to Main Memory These measurements have been taken on a Dual PII machine using the Intel 440FX chip set. On one CPU a real-time task was started which accessed the main memory performing 2048 read and write accesses. Attention has been paid not to cache the memory accesses. The second CPU was kept in an idle loop not affecting the measurement. The results of this measurement are illustrated in figure 2 and 3. If there is PCI activity in parallel, the execution time to perform the 2048 accesses varies by 72%; if there is not any PCI activity the execution times are varying by 2%.

Accesses to PCI Devices In the following measurements the time needed to access a PCI peripheral from a real-time task are examined. These measurements have been performed on a Dual Athlon machine using the AMD 760MPX chip set. The real-time task performed 1024 read accesses to a PCI peripheral using memory-mapped I/O, whereby always the same memory-mapped address was used in order to avoid burst transfers. The other processor was kept in an idle loop during each measurement. The results of this measurements are shown in figure 4 and 5. If the 1024 accesses are done with PCI activity in parallel, the execution time varies by 164%; if the Master Enable Bits are cleared, the execution time is nearly constant.

Time needed to enter the deterministic state We have also measured the time needed to clear the Master Enable Bit of the relevant PCI device — a PCI-to-PCI bridge — of the Dual Athlon. In parallel to each measurement the host was busy, performing network and disk I/O heavily. The time needed to clear the Master Enable Bit varies between 1.41 microseconds and 5.94 microseconds.

6. Conclusion

As the PCI peripherals are allowed to initiate transactions autonomously, there is an influence on the execution time of real-time software due to DMA burst transfers. These transfers affect the time needed to access the main memory from a CPU leading to varying execution times of software. On the other side, if a real-time task wants to perform I/O, the access time depends on the current workload on the PCI Local Bus.

In this paper a method is presented which makes accesses from a CPU across the PCI Local Bus behaving more deterministically. When switching to real-time context the bus may

be configured by clearing the Master Enable Bit of relevant devices. Thus only the Host Bridge is allowed to become initiator and all peripherals are only acting as slaves.

This method is useful for real-time systems accomplishing a lot of I/O. When estimating the WCET of specific code performing I/O, the time needed to enter the deterministic state has to be considered. The measurement based approach is useful when determining the WCET of a given system. However, hard- and software has to be configured in a way which leads to deterministic execution times.

References

- [1] Robert Baumgartl and Hermann Härtig. Efficient Communication Mechanisms for DSP-based Multimedia Accelerators. In *Proceedings of the International Conference on Signal Processing Applications and Technology (ICSPAT'97)*, San Diego, September 1997.
- [2] Franz Fischer, Thomas Kolloch, Annette Muth, and Georg Färber. A configurable target architecture for rapid prototyping high performance control systems. In Hamid R. Arabnia et al., editors, *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97)*, volume 3, pages 1382–1390, Las Vegas, Nevada, USA, June 30 – July 3 1997.
- [3] L. Lindh, T. Klewin, and J. Furunas. Scalable Architectures for Real-Time Applications - SARA. In *Swedish National Real-Time Conference SNART'99*, Linköping, Sweden, August 1999.
- [4] Laurent Moll and Mark Shand. Systems Performance Measurement on PCI Pamette. In Kenneth L. Pocek and Jeffrey Arnold, editors, *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 125–133, Los Alamitos, CA, 1997. IEEE Computer Society Press.
- [5] Stefan M. Petters. *Worst Case Execution Time Estimation for Advanced Processor Architectures*. PhD thesis, Institute for Real-Time Computer Systems, Technische Universität München, September 2002.
- [6] Stefan M. Petters, Annette Muth, Thomas Kolloch, Thomas Hopfner, Franz Fischer, and Georg Färber. The REAR framework for emulation and analysis of embedded hard real-time systems. In *Proc. of the 10th IEEE Int. Workshop on Rapid Systems Prototyping (RSP'99)*, pages 100–107, Clearwater, Florida, June 16–18 1999. IEEE Computer Society Press.
- [7] S. Schönberg. Impact of PCI-Bus Load on Applications in a PC Architecture. In *Proceedings of 24th IEEE International Real-Time Systems Symposium*, Cancun, Mexico, 2003.
- [8] Tom Shanley and Don Anderson. *PCI System Architecture*. Addison-Wesley Publishing Company, 3 edition, 1995.