Efficient Communication Mechanisms for DSP-based Multimedia Accelerators *

Robert Baumgartl, Hermann Härtig

Department of Computer Science
Dresden University of Technology, Germany
{robert.baumgartl,hermann.haertig}@inf.tu-dresden.de

Abstract

DSPs are very powerful microprocessors ideally suited for the execution of multimedia algorithms. The execution of CPU-intensive multimedia applications burdens conventional PC systems to a maximum. Therefore, the aim of our project is the integration of DSP subsystems as flexible multimedia accelerators into next generation personal computers.

However, existing DSP integration solutions do not take a highly efficient interprocessor communication into consideration which is essential for accelerating coprocessors. Hence, in this paper we discuss, compare and evaluate the most popular hardware communication mechanisms realized in today's DSP plug-in boards. Additionally, we discuss issues for an optimum implementation of the communication software.

1 Introduction

The unique architectural characteristics of DSPs predestine these microprocessors for multimedia applications [3], [6] as follows.

1. They allow a high degree of parallelity and incorporate special features into the instruction set and into arithmetic and addressing units resulting in an extremely powerful processor core. Most multimedia algorithms require exactly this vast amount of computing power.

2. DSPs are designed to handle large amounts of data, which are common in multimedia applications.

3. DSPs are comparatively easy to operate under real-time conditions. It is much easier to obtain exact execution times for a given piece of DSP code than for General Purpose Processor (GPP) code. Most multimedia tasks (e.g. video or audio codecs) run under realtime conditions.

Unfortunately, in the past, personal computers equipped with DSP plug-in boards were almost solely used as development systems. The amount of data exchanged between the system's CPU and the DSP was small and realtime operation was not necessary. The host was degraded to a control terminal. Hence, the hardware interfaces of these boards were designed without regard to the data transmission performance but paying attention to cost effective solutions only.

The availability of state-of-the-art DSPs running at clock frequencies of 200 MHz as well as the PCI bus system capable of transmitting data up to 132 MBytes/s have changed this situation considerably. With these latest developments, DSP plug-in boards are able to work as accelerators for the first time.

The prototype discussed in [2] demonstrated the capabilities of a basic DSP multimedia accelerator. The main disadvantage of the presented solution was the communication overhead introduced by various system components.

The following premises proved being essential for competitive results:

- The interface hardware mechanisms must allow high data transmission rates.

*This work was supported by the DFG with the Graduiertenkolleg "Tools for the effective Use of Parallel and Distributed Computing Systems" and the SFB 358.

- The implementation of the inter–processor–communication services has to be as efficient as possible.
- The pure processing power of the DSP must at least meet the power of the General Purpose Processor.
- The host operating system has to provide ways for efficient communication between applications running at the host and processes executed on the DSP.

This paper reports on experiments carried out to identify ideal hardware communication mechanisms, hence it concentrates on the first two points. We discuss, compare and evaluate the most popular hardware interfaces between DSP plug-in boards and PC hosts. Special attention is paid to aspects of PCI busmaster DMA transfers like comparison of various transfer block lengths, or the identification of potential performance bottlenecks.

2 The Testbed

For our experiments we employ a standard PC equipped with a single Pentium Pro CPU running at 200 MHz. The system incorporates 64 MBytes RAM, the host operating system is Linux. An experimental device driver compiled with gcc with optimization turned on is responsible for communication with the DSP.

The DSP subsystem consists essentially of a modular PCI plug-in card offering three TIM40-conform and two Industry-Pack (IP) sites. Only a single TIM module carrying a TMS 320C44 [5] clocked at 50 MHz together with 256 kWords static RAM is installed. The board features an AMCC S5933 PCI controller [1] which is capable of performing PCI busmaster DMA. However, DMA transfers can only be initiated by the DSP side.

Dedicated to communication between the CPU of the host and the installed DSPs are four 32-bits-wide mailboxes with the according hardware semaphores for access control. The 8-words-deep FIFO of the PCI controller has been expanded externally with a 512-words-deep external FIFO which is transparent to the programmer. The mailboxes and the FIFO are mapped into the port address space of the host and into the address space of the DSP. All DSP software development is done using the TI language tools suite.

For exact timing measurements the mechanisms described in the following sections were analyzed by counting clock cycles with the Pentium's Model Specific Registers (MSRs). All obtained timing values were averaged at least 5 times to eliminate random errors.

3 Hardware Mechanisms

In the following section we individually discuss and evaluate the communication mechanisms encountered on DSP-equipped PC platforms today: mailboxes, dual-ported memory and PCI busmaster DMA.

We do not consider serial interfaces which are common for most of the introductory level DSP boards (e.g. evaluation kits) because PC standard serial communication is limited to 115 KBits/s which is far too slow for processing of multimedia data. However, this situation might change soon with the establishment of new serial busses, like Firewire.

3.1 Mailboxes

Mailboxes are one of the simplest communication mechanisms apart from serial interfaces. They usually consist of single 8 to 32 bits wide words mapped into an address space of the host and the DSP. To guarantee mutual exclusive access for both processors hardware semaphores are usually used which must be aquired before and released after accessing the mailbox.

Data transfer using mailboxes is usually slow and is CPU-intensive because for every word transfer the following mechanism must be used (transfer from host CPU to DSP).

1. The CPU tests and aquires its respective hardware semaphore.
2. The CPU writes a word to the mailbox.
3. The CPU releases the semaphore.
4. The DSP tests and aquires the semaphore.
5. The DSP reads the word.
6. The DSP releases the semaphore.

These operations take place in strict sequential order. State changes of the mailbox and semaphores can be observed by polling or are signalled by interrupts.

Because of its simplicity and the low host bus load the determination of mailbox communication times is easy. This can be an advantage under real-time constraints.

We measured one-way communication performance using mailboxes with different message sizes. Table 1 shows the results for the direction from the host CPU to the DSP. Overhead introduced by measurement and signalling is always eliminated. The message was transmitted word-by-word with the mechanism described earlier and transfer completion was confirmed by the DSP.

Msg Length $[Bytes]$	Transfer Time	Transfer Rate $[Kbytes{\cdot}s^{-1}]$
128	83.2 μs	1502.4
512	323.7μs	1544.6
4096	2.51 ms	1593.6
32768	20.14 ms	1588.9

Table 1: Communication performance using mailboxes

Clearly, we obtained an almost constant transfer rate regardless of the message length. Overhead introduced by per-transfer basis is obviously small. A very small advantage for longer transfers (approx. 6%) can be observed which is due to host caching effects. The achievable maximum transfer rate is dictated by the expensive sequential signalling mechanism in conjunction with the necessary word-wise transfer.

The bare signalling scheme needed a time of only 160 ns which indicates the low setup overhead for the mechanism.

In comparison to other mailbox-based implementations [2] an average transfer rate of 1.5 MBytes/s is quite good. But clearly, it is far from acceptable for multimedia stream processing.

3.2 Dual-ported Memory

Dual-ported memory is a faster hardware mechanism than single mailboxes. In principle, it consists of a continuous memory section which is mapped into both processors' address spaces. On the host side the memory is mapped into the main memory space (instead of the port address space used for mailboxes) which results in faster accesses. The memory section is typically 1..16 KBytes large. Usually, the same access mechanism, namely hardware semaphores as described for mailboxes, is used. The big advantage is that the whole mechanism of testing and obtaining the semaphore, accessing the memory and releasing the semaphore is needed only once for the whole memory section instead of every single word.

Disadvantageously, dual-ported memory is very expensive; so board manufacturers equip their products only with a few KBytes. Additionally, it needs more CPU intervention for data transfer than PCI Busmaster DMA presented in the next section.

Up to now, we were not able to obtain a DSP board equipped with dual ported memory and fitting into our testbed. Therefore we cannot present corresponding data transfer measurements here.

3.3 Busmaster DMA

With the introduction of the PCI bus standard [4] a new possibility of communication between plugged-in hardware and main memory emerged, the PCI busmaster DMA. It is intended for high-speed data transfer and is in theory capable of 132 MBytes/s maximum transfer rate. As an example we look at a transfer from host to DSP memory employing our hardware configuration.

1. The host allocates a buffer suitable for DMA transfer and fills it with data to be transferred.

2. The host signals the DSP that the buffer is ready for transfer (e.g. by mailbox communication or by interrupt).

3. The DSP programs the PCI controller and defines necessary parameters (start address and length of the host buffer, transfer direction).

4. The DSP initiates the DMA by writing a certain status register of the PCI controller.

5. The PCI controller transports data from the host buffer into its own FIFO transparently to the CPU.

6. When the FIFO is filled the DSP receives a signal (e.g., an interrupt) to empty the FIFO.

7. The DSP empties the FIFO by copying the data into an appropriate receive buffer. Simultaneously, the PCI controller again fills the FIFO with new data provided there is space left in it. This proceeds until the end of the host buffer is reached.

The dedicated transfer hardware, the higher degree of parallelity as well as the invisibility to the CPU guarantee a superior performance of this mechanism in comparison to mailboxes. On the contrary, the necessary overhead per transaction is comparatively high favouring mailboxes for short message sizes.

To obtain real-world experience we implemented DMA transfer on our system employing a HLL communication library provided by the board manufacturer. We measured transfer rates for various message sizes beginning with the smallest message size possible equalling the size of the board's FIFO (2 KBytes). During transfer other bus-intensive operations of the host were disabled. Table 2 shows the results.

Msg Length $[Bytes]$	Transfer Time $[ms]$	Transfer Rate $[Kbytes{\cdot}s^{-1}]$
2048	0.690	2898.6
4096	1.31	3053.4
8192	2.54	3149.6
16384	5.00	3200.0
32768	9.94	3219.3
65536	19.80	3232.3
131072	39.52	3238.9

Table 2: Communication performance using unoptimized Busmaster DMA

The transfer performance was disappointing. Although the transfer was twice as fast as mailbox-based communication it was far away from the promised 132 MBytes/s. Unexpectedly, the increase of the message size did result only in an performance improvement of approximately 10%. This suggested an implementation bottleneck limiting the potentially much faster hardware communication mechanism. An increase of the optimization level of the DSP compiler accomplished nothing as did an optimization of the DSP device driver running on the host. So we decided to investigate and analyze the DSP implementation closely.

We found out that the DSP spent 25% of the whole communication time in the interrupt service routine responsible for the data transport from the board's FIFO to the DSP memory. We analyzed the routine's compiler-generated code and estimated a considerable speedup for hand-optimized assembly code.

The optimization concentrated on the following techniques:

- optimize register usage
- move unnecessary index calculations out of the inner loop
- employ hardware looping
- eliminate pipeline stalls by careful instruction scheduling
- take advantage of the Harvard Architecture (a read and a write operation simultaneously).

As a consequence of the optimization we were able to cut down the length of the inner reading loop from 15 to 1 instruction cycles. After the optimization process we repeated the measurements and obtained the results shown in table 3.

Msg Length $[Bytes]$	Transfer Time $[ms]$	Transfer Rate $[Kbytes{\cdot}s^{-1}]$
2048	0.11	18182
4096	0.16	25000
8192	0.28	28571
16384	0.51	31373
32768	0.98	32653
65536	1.91	33508

Table 3: Communication performance using optimized Busmaster DMA

The achieved performance demonstrates the capability of the PCI bus. We measured a peak performance of 32.7 MBytes/s for a sufficiently large memory block. Furthermore, we observed a comparatively high per-transaction overhead, indicated by substantially slower transfer rates for short messages. But, the major speedup extending from 6:1 up to 10:1 for the communication code implemented in hand-optimized assembly language compared to the HLL implementation of the mechanism is most important. This demonstrates the necessity of assembly language for time-critical DSP code.

The empty signalling mechanism without any data transfer needed a time of 15.5 μs mainly caused by the necessary programming of the PCI controller.

A serious disadvantage of busmaster DMA is that for hardware reasons the minimum message size is limited to the FIFO size. This is especially important for messages of only a few bytes length as they are frequently encountered in microkernels. Furthermore, message sizes which aren't multiples of the FIFO size need special attention during PCI controller programming which is in general undesirable.

Another drawback of this mechanism is its bad timing predictability. Simultaneous bus operations (e.g., data transfer between graphics memory and main memory) can interfere with the communication between DSP and host CPU. The transfer rate between the two processors depends on the PCI bus load. Even worse, it is to our knowledge impossible to give timing guarantees for DMA operations involving the PCI bus. That means that in general only soft real-time conditions can be fulfilled by PCI accelerators. Still, multimedia applications running on standard PCs do not require hard real-time deadlines.

4 Conclusions

Our experiments clearly indicate the following facts:

First, mailboxes are too slow for the transfer of large multimedia streams, regardless of the underlying bus system. Their predictable timing and the small latency for short messages predestine them for operating system communication.

Second, PCI Busmaster DMA can be fast, but this requires a very efficient implementation. The transition from high level language to assembly language proved to be essential in relevant parts of the communication code. With this a transfer rate of over 30 MBytes/s is possible. It remains to be investigated whether a further speedup can be achieved or not.

Third, PCI Busmaster DMA introduces a significant overhead and is therefore only relevant for sufficiently large data streams. For short messages with highly predictable timing it must be complemented by another communication mechanism.

5 Outlook

The next step of our investigations is a comparison of several bus systems regarding their timing predictability under various load conditions. As we pointed out in this paper, determining the exact duration of DMA transfers can be impossible. Therefore, we want to find out whether statistical guarantees for the duration can be given and whether other hardware approaches (e.g. a separate SCSI chain) are more appropriate.

The following stage of our project is the transition from monolithic kernels like Linux running on the host to microkernel architectures thus reducing the communication overhead introduced by the host's operating system.

References

[1] Applied Micro Circuits Corp., *S5933 PCI Controller Data Book.* San Diego, 1996

[2] Robert Baumgartl, Hermann Härtig, *On the Integration of DSP Hardware into a Microkernel-based Operating System.* Proceedings of ICSPAT'96, Boston, 1996

[3] Phil Lapsley, *DSP Chips enable PC Multimedia.* In: Microprocessor Report 8(1994), 1994

[4] Tom Shanley, Don Anderson, *PCI System Architecture.* Addison-Wesley, New York, 1995

[5] Texas Instruments Inc. *TMS320C4x User's Guide.* Austin, 1996

[6] V.Živojnović et al., *DSPs, GPPs, and Multimedia Applications - An Evaluation Using DSPstone.* Proceedings of ICSPAT'95, Boston, 1995