Journal of Systems Architecture 56 (2010) 442–451

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Multi-port abstraction layer for FPGA intensive memory exploitation applications

M. Vanegas [a,b,*], M. Tomasi [a], J. Díaz [a], E. Ros [a]

[a] Department of Computer Architecture and Technology, University of Granada, 18071 Granada, Spain
[b] Microelectronic Group, Pontificia Bolivariana University, Medellín, Colombia

ARTICLE INFO

Article history:
Received 1 September 2009
Received in revised form 17 April 2010
Accepted 9 May 2010
Available online 19 May 2010

Keywords:
Memory-control unit
FPGA
Video processing
Hardware design
Real-time processing

ABSTRACT

We describe an efficient, high-level abstraction, multi-port memory-control unit (MCU) capable of providing data at maximum throughput. This MCU has been developed to take full advantage of FPGA parallelism. Multiple parallel processing entities are possible in modern FPGA devices, but this parallelism is lost when they try to access external memories. To address the problem of multiple entities accessing shared data we propose an architecture with multiple abstract access ports (AAPs) to access one external memory. Bearing in mind that hardware designs in FPGA technology are generally slower than memory chips, it is feasible to build a memory access scheduler by using a suitable arbitration scheme based on a fast memory controller with AAPs running at slower frequencies. In this way, multiple processing units connected through the AAPs can make memory transactions at their slower frequencies and the memory access scheduler can serve all these transactions at the same time by taking full advantage of the memory bandwidth.

1. Motivation

In recent years FPGA technology has evolved from being a validation framework to a computing platform. Given that the performance gap between FPGAs and ASICs has been significantly reduced [1], over the last decade ASICs have been replaced by FPGAs in some electronic industries; in the networking field, for instance, routers have FPGAs incorporated into their circuitry to minimize the time to market and related costs. FPGAs are currently being used in the field of system-on-chip design (SoC) [2,3] because they can now offer sufficient resources, even in some cases on-chip hardcore processors. Modern FPGA devices allow massive parallel on-chip computing through deep-pipelined data-paths with large numbers of super-scalar processing units [4,5]. Furthermore, many processing tasks are executed, in a fixed pattern, for a lot of data and their implementation is thus conducive to making the most of the capacity for parallelism offered by FPGAs. Unfortunately, the use of FPGAs requires more advanced hardware design skills to achieve complex systems than those needed to make the same system using GPU-based platforms.

The computing platform's performance is quite sensitive to the behavior and limitations of the memory system. Processors have traditionally used memory hierarchy schemes, in which small memories with faster access times are located close to the processors whereas larger capacity memories with slower access times are located far away from the processors [6]. As a matter of fact, data are moved from larger memories to the smaller ones based on spatial and temporal data locality principles [7,8]. This allows the processors faster memory access. Although these principles work well for most algorithms, if an irregular data access is required, the system's performance will probably be significantly degraded. In fact, code optimization techniques are highly dependent on data structure [9]. As an application example, in real-time video processing systems access to all information and temporary results arrive at a bottleneck; furthermore, the use of a compression module does not entail an increase in system performance since access to compressed data is usually data-dependent and an irregular memory access must be used.

Multiple parallel processing entities are possible in current FPGA devices [10], but this parallelism is forfeited when they try to access external memories. From now on in this paper, the term "external memories" will refer to all the memory chips connected to the FPGA. The inherent sequential behavior of the external memories may limit the system's performance. Therefore, this potential bottleneck must be efficiently dealt with in high-performance systems. Access to external memories must be implemented in specific time windows when implementing massive parallel data-paths (with fine-pipelined processing structures) to avoid data collisions [11]. This task is critical and it would be useful to abstract the memory access to facilitate the design of multiple parallel entities with intensive external memory access requirements. We describe here a generic memory-control architecture designed specifically for reconfigurable hardware (FPGA devices) to be used in embedded

* Corresponding author at: Department of Computer Architecture and Technology, University of Granada, 18071 Granada, Spain. Tel.: +34 607195837.
E-mail addresses: mvanegas@atc.ugr.es (M. Vanegas), mtomasi@atc.ugr.es (M. Tomasi), jdiaz@atc.ugr.es (J. Díaz), eduardo@atc.ugr.es (E. Ros).

1383-7621/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysarc.2010.05.007

systems. Nonetheless, because of its RTL description, the memory-control architecture can easily be adapted to an ASIC.

Nowadays, the use of high-level synthesis tools is becoming usual in most academic and industrial environments. In particular, designs for FPGA devices are being made by using high-level synthesis tools to speed up the design process. Current FPGA circuits can be connected using visual box schemes such as System Generator [12], or the circuit connections may be described using C-like descriptions [13–15], C++ based [16] etc. Over the last decade these languages and tools have been developed considerably and their use is becoming widespread. They are presented as a way in which inexperienced designers can design hardware for reconfigurable devices and also as a way of speeding up the design process. Nevertheless, although these languages accelerate the design process to an acceptable extent, access to the peripherals sometimes leaves much to be desired. In fact, memory controllers must be described using low-level synthesis tools in order to achieve better performance. The purpose of this paper is to describe a memory access control provided with a certain abstraction level capable of using the physical memory working at full capacity. The main benefits of the memory controller are the abstraction of the memory access through ports to read from and to write to, the maximization of the bandwidth at each port according to the number of open ports and its capability of being used in most embedded systems because of its low power consumption.

The paper is organized as follows. Section 2 contains an overview of some related other memory controllers as well as the contexts in which they have been used. The hardware architecture of the MCU together with the results of its implementation are described in detail in Section 3. In Section 4 we assess the performance of the MCU by using it in a real system and finally, in Section 5, we offer a brief summary of the conclusions of our work.

2. Related work

Several authors have studied methods for optimum memory access in media-processing systems. Liu et al. [17] developed a technique to optimize memory access to scratchpad memories (SPMs) within loop nests in order to take advantage of data re-use. Ranganathan et al. [18] implemented dynamically reconfigurable caches in FPGAs to support processor-based platforms to dynamically adapt the cache parameters to different phases of an application and achieve a better system performance. Sohi and Franklin [19] increased data parallelism by using a multi-port cache. Another kind of memory-control interface in FPGAs was introduced by Heithecker et al. [20], who proposed a memory access based on memory scheduling to reduce inherent latency; in this way parallel units can have access to memory concurrently and achieve high bandwidth use. In a recent work, Su-Shin et al. [21] proposed a hybrid architecture that uses a custom parallel cache and a scratchpad memory to allow effective data re-use in spite of data-dependent memory accesses. Nevertheless, all these methods are application-dependent and a more general solution would be extremely useful.

The memory access problem is even more complex if we are working with multi-core processor architectures. In this case, small independent cache memories are located close to each processor whilst larger caches are shared between the different processor cores (see [22]). Moreover, if the application demands a very large volume of data transference, a very large memory bandwidth is also required, such as the one used by graphic processing units (GPUs). For instance, GPUs working on rendering high-resolution images require a large number of memory chips clocked at a very high frame rate [23,24]. Current Tesla NVIDIA GPUs use 512-bit memory DDR3 interfaces clocked up to 1.6 GHz. The memory bandwidth is impressive and contributes significantly to the computing performance of these architectures.

The previous outline shows the importance of a proper memory architecture and hierarchy. Unfortunately, with embedded systems memory issues often limit the system's performance. In fact, high-bandwidth memories are not usually available (or are unsuitable for most applications) and the number of physical memories (devices) should be restricted to a minimum to reduce cost and system power dissipation [25]. Currently, marketers of FPGAs have developed memory controllers for high-bandwidth memories (DDR, DDR2 and DDR3) for use with their products. Xilinx Inc. [26] offers the multi-port memory controller (MPMC). This MPMC provides eight ports, each able to transfer data at lower clock frequency than that used by the controller for interfacing with the DDR. The strategy used by the Xilinx developers is quite useful due to the fact that most of the FPGA designs run at low frequencies compared to the effective DDR frequency.

So far, all works dealing with this problem have concentrated on massive data transfer by using burst transfer in order to increase the memory frame rate, and also on spatial and temporary data-locality schemes to improve the system's performance. Within this context, the MCU described in this paper provides a better solution in applications in which irregular data access renders burst transfer and temporary data-locality schemes useless.

3. MCU hardware implementation

To address the problem of multiple entities accessing shared data we propose an architecture with multiple AAPs to access one external memory. Bearing in mind that hardware designs in FPGA technology are generally slower than memory chips, it is quite feasible to build a memory access scheduler by using a suitable arbitration scheme based on a fast memory controller with AAPs running at slower frequencies. Therefore, the whole design has two clock domains, a slow frequency domain (CD_s) and a fast frequency domain (CD_f), the memory controller always being faster than either AAP. Thus access to data through the ports may run concurrently in the AAP clock domain when the ratio between clock frequencies (memory controller frequency divided by AAP frequency) is at least the number of AAPs.

To achieve a better level of abstraction the MCU is provided with two kinds of AAP (reader and writer) and so the MCU consists of a memory controller and several AAPs (readers and/or writers) depending on the system's requirements. Each AAP works independently; the MCU synchronizes all the requests made by the AAPs and guarantees bandwidth distribution among all open ports. Fig. 1 shows a block diagram illustrating a two-port MCU configuration. It should be noted that the reader AAP uses two interfaces, in contrast to the writer AAP, which uses just one.

The MCU was synthesized and implemented in a Xilinx [26] *XC4VFX60-10* FPGA of the *Virtex4* family in a Seven Solutions *Xirca-V4* platform [27]. The *Xirca-V4* has 4 flow-through zero bus turnaround (ZBT) SSRAM memory chips 72-Mbit ($2M \times 36$), which are totally independent of each other in data, address and control signals. The memory clock signals are shared in the platform. For each ZBT SSRAM memory chip there is a specific MCU designed in VHDL language.

3.1. Abstracted access port (AAP) design

The AAP block is a critical stage in the design due to the problem of crossing clock domains. The data passes across two clock domains, the first being the clock used by the entities connected to the AAPs and the second that used by the MCU. The best way of dealing with different clock domains within one FPGA is to use FIFOs. By using the