Integration issues of a run-time configurable memory management unit to a RISC processor on FPGA

Farid Shamani, Vida Fakour Sevom, Tapani Ahonen, Jari Nurmi

Department of Electronics and Communications Engineering, Tampere University of Technology, Tampere, P.O. Box 553, FIN-33101, Finland

A R T I C L E   I N F O

Article history:
Received 13 April 2016
Revised 16 September 2016
Accepted 2 December 2016
Available online xxx

Keywords:
FPGA implementation
Memory management unit
Virtual-to-physical address translation
Run-time configurable MMU
RISC processor

A B S T R A C T

This paper presents the integration issues of a proposed run-time configurable Memory Management Unit (MMU) to the COFFEE processor developed by our group at Tampere University of Technology. The MMU consists of three Translation Lookaside Buffers (TLBs) in two levels of hierarchy. The MMU and its respective integration to the processor is prototyped on a Field Programmable Gate Array (FPGA) device. Furthermore, analytical results of scaling the second-level Unified TLB (UTLB) to three configurations (with 16, 32, and 64 entries) with respect to the effect on overall hit rate as well as the energy consumption are shown. The critical path analysis of the logical design running on the target FPGA is presented together with a description of optimization techniques to improve static timing performance which leads to gain 22.75% speed-up. We could reach to our target operating frequency of 200 MHz for the 64-entry UTLB and, thus, it is our preferred option. The 32-entry UTLB configuration provides a decent trade-off for resource-constrained or speed-critical hardware designs while the 16-entry configuration poses unsatisfactory performance. Next, integration challenges and how to resolve each of them (such as employing a wrapper around the MMU, modifying the hardware description of the COFFEE core, etc.) are investigated in detail. This paper not only provides invaluable information with regard to the implementation and integration phases of the MMU to a RISC processor, it opens a new horizon to our processor to provide virtual memory for its running operating system without degrading the operating frequency. This work also tends toward being a general reference for future integration to the COFFEE core as well as other similar processor architectures.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

In sophisticated embedded systems, main memory access has the potential to cause some extra delays. Even the fastest processors should be stalled while accessing to the main memory. Hence, speeding up the main memory accesses are one of the most important concerns of the processor designers [1]. One of the main applications of the Virtual Memory (VM) is to share the main memory among multiple programs. The VM allows each program to reference a particular part of the main memory while it prevents the same program accessing to the rest of the memory dedicated to other programs. Technically, as the caches provide faster access to the active portions of a program, the main memory can potentially act as a cache for the VM. A wide research in [2] shows that exploiting an L2 cache in parallel with an L1 cache which offers a hit rate of 90%, brings the hit rate up to 95%. Although the hit rate is only improved by a factor of 5% (in two-level cache system), the miss rate is reduced to half in comparison to the single-level cache system (decreased from 10% to 5%). This is the basic idea that we took into consideration to expand the VM on several hierarchical levels. Using this scenario, instructions have the potential to be executed faster.

1.1. Motivation

Many years ago, programmers were responsible for manually fitting programs which were larger than the main memory storage. The virtual memory eliminated the substantial burden on programmers by moving inactive portions of the program out of the physical memory. As soon as the other programs require more main memory, the inactive portions of the running program(s) are stored in a secondary storage using the virtual memory mechanism. Therefore, the virtual memory enables multitasking by instantly relocation over the main memory [3].

Conceptually, in the virtual mode, the processor produces a virtual address which is convertible to the physical one using a combination of hardware and software approaches. Fig. 1 shows the
address translation procedure. This mechanism is usually performed by the Memory Management Unit (MMU). The primary function of the MMU, which can also be denoted as the most important one, is to translate the virtual addresses into the physical ones [4]. The MMU should guarantee the protection of each program, as well. On the other hand, the MMU is sufficiently rapid to enable all transactions to the main memory [5]. Theoretically, any VM access takes at least as twice long as the main memory access. It requires one memory access to obtain the physical address and a second attempt to get the data. Therefore, by keeping these accesses in a special cache, subsequent memory access will be executed much faster than the first one. Accordingly, the most frequent page translations are kept in a special cache which is referred to as a Translation Look-aside Buffer (TLB). In the virtual mode, the MMU examines its contents to find a matched Virtual Page Number (VPN). When the desired entry is found, the appropriate physical page number is extracted from the corresponding entry and, then, is combined with the offset to form the physical address. The processor employs the obtained physical address to reference a specific location in the memory. On the other hand, rapid technology advancement has enabled digital system designers to design and implement embedded processors capable of performing particular functions [6]. One such embedded processor is the Core For FREE (COFFEE) developed by our group at Tampere University of Technology. All the required materials to run the COFFEE processor (including the latest source codes, the corresponding compiler, etc.) as well as the instruction how to use the core can be found on the university website in [7]. In this work, we used the most updated version of the core released on 17-November, 2008. As the same states, the developed processor is an open-source core which is suitable for embedded computing. However, the processor lacks the integration of an MMU to support a virtual-to-physical address translation mechanism [8]. Indeed, configuring the COFFEE RISC processor with a fully compatible MMU is one the main motivations for this work. Potentially, the developed MMU can be integrated into other standard RISC-based platforms, as well.

1.2. Related works

There are other studies in which an MMU has been implemented on the FPGA devices for different applications. In [9], the authors proposed a method based on a Network Interface Controller (NIC) which includes a bulk memory. A 128 MB DRAM memory stores 16 million of Address Translation Table (ATT) entries (instead of TLBs) which include information with regard to virtual-to-physical address translations. Each entry of the ATT is composed of 64 bits. After the first run, the NIC driver maps all the virtual memories to the physical ones. Subsequently, the ATT is maintained only by the driver. However, since DRAMs (along with the CPU) are the major candidates for total system energy consumption [10], the address translation in such systems is an expensive mechanism in terms of energy consumption.

Hu-Cheung Ng et al. implemented an MMU on a Xilinx Virtex 5 FPGA board in [11]. The MMU has a TLB which includes virtual-to-physical address translations. The TLB has 16 entries and employs the Least Recently Used (LRU) replacement policy. They also claimed that another small cache which contains 4 entries improved the speed of the memory access. The implemented MMU only supports a fixed page size of 4 KB. A successful translation takes 2 cycles whereas any update to the TLB takes 16 cycles. A TLB miss penalty is very varied between 600 to 227,000 cycles depending on the status of the Operating System (OS).

In [12], a reconfigurable architecture which employs a single 512-entry TLB for address translation mechanism alongside a Direct Memory Access (DMA) unit is proposed. In the case of a miss occurrence, the TLB is locked and the FPGA will be interrupted. Consequently, the FPGA will update the TLB with the corresponding missed entry. Thereafter, the TLB is unlocked and the page translation mechanism will be continued. The authors acknowledged that any TLB miss would take thousands of FPGA cycles whereas an address translation with a TLB hit would only take 4 cycles. They also claimed that the hardware implementation (on Xilinx Virtex 5 FPGA device) significantly sped up the performance (approximately five times) over the pure software implementation. However, they have not mentioned the size of the virtual page space supported by the TLB.

The authors in [13] introduced a specialized TLB design which includes a Buffer Search Ram Cache (BSRC) along with a Virtual-to-Physical (V2P) unit where the V2P is mainly responsible for the address translation mechanism. A complete V2P block composed of eight 32-bit entries was integrated with Content Addressable Memories (CAMs) as a specialized memory to act as a hardware search engine. It takes different patterns as search keys and returns the corresponding addresses where the patterns are stored. This architecture, due to the use of tightly coupled RAM structures, returns a translated address in 1 or 2 cycles whereas deleting or updating a new entry is a complex operation which requires several steps. Both the analysis results and the hardware costs are given based on the implementation on Altera Stratix IV FPGA family specification. Of course exploiting a CAM module instead of a simple RAM will result in a very complex implementation (by a factor of 32) which is a very limiting factor in designs with scarce memory resources [13].

This paper is based on our previous work in [14]. In our earlier work, we developed our MMU on two levels of TLBs, since the process of address translation was quite similar to the caching procedure. On the first level, a 4-entry micro-TLB for instruction memory references operated alongside an 8-entry micro-TLB for data memory references. Both of these micro-TLBs were hardware managed. On the second level, there was a 64-entry unified software-managed TLB to cache both instruction and data page address translations. All employed TLBs were design-time configurable and scalable. The MMU was able to configure itself (mainly it reconfigured the TLBs) to operate on eight different page sizes from 1 KB to 16 MB during the run-time. The MMU was implemented on a 28 nm Altera Stratix-V FPGA board [15] based on the specification given in [16] in absolute terms. In this extended version, we try
دریافت فوری متن کامل مقاله

امکان دانلود نسخه تمام متن مقالات انگلیسی
امکان دانلود نسخه ترجمه شده مقالات
پذیرش سفارش ترجمه تخصصی
امکان جستجو در آرشیو جامعی از صدها موضوع و هزاران مقاله
امکان دانلود رایگان ۲ صفحه اول هر مقاله
امکان پرداخت اینترنتی با کلیه کارت های عضو شتاب
دانلود فوری مقاله پس از پرداخت آنلاین
پشتیبانی کامل خرید با بهره مندی از سیستم هوشمند رهگیری سفارشات