An educational framework for learning organization and performance of cache memory

[1]João P. Cunha, [2]António J. Araújo
[1] Faculdade de Engenharia da Universidade do Porto, Portugal
ee04226@fe.up.pt[1]
[2] INESC Porto, Faculdade de Engenharia da Universidade do Porto, Portugal
aja@fe.up.pt[2]

Abstract

Cache memories can improve the performance of processors but choosing the better cache for a system is not trivial. This paper presents a tool that helps understand the operation of a cache whose parameters can be defined with a Java interface. Moreover, the tool can generate synthesizable Verilog code of a parameterized cache to incorporate in a specific design. The whole system, including the cache is implemented successfully on a FPGA prototyping board allowing the users to test different caches and supervise their performances.

1. Introduction

The need to create the cache memory emerges when memories were no longer able to keep up the processor speed. The growing disparity between processor and memory speeds put the cache memory as a critical factor for improving the performance of a program. Cache memory is mainly used to obtain a higher memory access speed, while providing the system a large memory capacity. However choosing the right parameters of this memory is not always simple.
The selection of a cache is usually based on general characteristics such as size, speed and efficiency. Naturally the goal is pursuit better choices, so that is why it is important the understanding of these characteristics. Parameters like cache size, internal organization, replacement strategy and write policy can all affect the performance of the system.

There are many ways to improve the cache performance. In this paper it is presented a framework to help in this regard. The framework will output an efficient cache implementation satisfying a given set of input parameters. The tool is versatile and allows to generate a set of different cache organizations by choosing cache parameters to understand how to get better performances. The well-known MIPS processor [1] was considered to build the system cache. The framework was successfully implemented on a FPGA based on a reconfigurable environment. In this paper, it will be provided implementation results concerning area and speed for different organizations and size of caches on a FPGA platform.

The main motivation to build this reconfigurable cache system was to create a teaching tool. Two specific groups of students were considered the main target for this tool. First, computer science and electrical engineering students that, as part of an introductory course in computer architecture, need help to understand basic concepts about cache memory. The idea was to create a helpful tool to facilitate the comprehension about cache parameters that improve performance, clarifying students to understand how performance can be tuned. In fact, they can study the behavior of a cache more easily, testing different cache organizations with different cache size through this tool. The second group is the electrical engineering graduated students. The framework provides a generator of synthesizable Verilog code of the cache memory circuit and their controller. So, this resource can be useful if these students need a cache to increase the performance of an application that uses memory frequently.

The rest of this paper is organized as follow: Section 2 includes a background about MIPS processor, cache aspects and reports some similar tools. Section 3 describes the general organization of the developed framework and gives some details about the implementation.

Section 4 presents when and how the framework can be advantageously used to improve the teaching and learning process. Section 5 describes the main conclusions and future improvements.

2. Background

The MIPS processor is a 32-bit reduced instruction set computer (RISC). It has a 5-stage pipelined datapath composed by an ALU (Arithmetic Logic Unit) and a register file, where the operands and operation results are kept, and the corresponding control unit. The MIPS was a pioneer in the development of RISC CPUs. Nowadays, it is present in many consumer electronics equipment and embedded systems. MIPS processor was selected to this project because it is a simple processor with a simple instruction set, turning it a good choice for didactic purposes. Moreover, it is largely used to teach computer architecture courses in universities around the world.

A cache is a small size memory located close to the processor storing data that is more frequently used by the processor. It allows to reduce the mean access time to main memory, improving in this way the overall performance. As has been said parameters like cache size, type of organization, replacement policy and write policy can affect the performance of a system. Replacement policy is the strategy used to select which data is replaced from the cache. The write policy is the mechanism used to ensure the integrity of data processed in the system, despite the transfers between the main memory and cache memory. The organization of a cache can be one of three: direct-mapped cache, fully associative cache and *n*-way associative cache.

The idea of using multimedia resources to help teaching cache memory is not new. Some tools with similar goals have been proposed in the literature. A representative set follows. Dinero IV [2] is a trace-driven uniprocessor cache simulator that allows many combinations of cache design parameters. Despite being a powerful tool, is difficult to configure and does not provide a graphic user interface. Dinero IV is a command-line driven simulator written in C. CAMERA [3] provides users with interactive tutorials and simulations to help students understand concepts about cache mapping and also virtual memory using paging. It is written in Java Swing. SIRCA [4] is a reconfigurable cache simulator designed to show that tuning cache parameters properly can improve performance, as a result of a reduction in the miss rate. SIRCA is written in Java Swing, simulates a specific reconfigurable cache design and provides hit and miss rates.

The framework here proposed has an interface written in Java, similar to the CAMERA and SIRCA, but all the rest is implemented in hardware as a reconfigurable system. This framework distinguished from other tools by the fact that the implementation was created on a FPGA based platform, i.e. there is a real implementation instead of a simulation process. Moreover, it allows the user to extract the synthesizable Verilog code of the cache memory optimized to use within different projects.

3. Framework

3.1 Description

The developed framework is composed by hardware and software components. The hardware part implements the MIPS processor, the main memory, the cache memory, the memory controller and some other blocks that are responsible by the configuration and communication between a PC and the FPGA platform. The software interface running on a PC allows the configuration and the operation of the whole system connecting with the hardware by a RS-232 serial port.

The main role of the interface is to control and supervise the system. It is responsible for sending data (like parameters of a cache memory) and supervises the values received. It can show results like number of misses, number of hits and time report expressed in terms of clock cycles. Inside the FPGA they are the MIPS, the memory controller, cache and communication

blocks. These blocks have to control the information between software and hardware. The MIPS processor is responsible by the execution of programs. To do that, the user has to write a program and translate to machine code. For this purpose it is used the open source CASPR (Configurable Assembler Program) [5] which was modified to suit the MIPS assembly language specifications [6]. The code is loaded into an instruction memory and then MIPS is ready to run. Whenever an instruction is a load or a store the memory controller is activated. It guarantees the operation of the system when a memory access occurs, i.e., when MIPS execute a load or a store instruction. The cache is implemented in specific memory blocks of the FPGA. So, cache is near the processor, both housed in the FPGA. The SDRAM is an external chip of the FPGA and is where the main memory of the system is implemented. Figure 1 shows the architecture of the whole system.

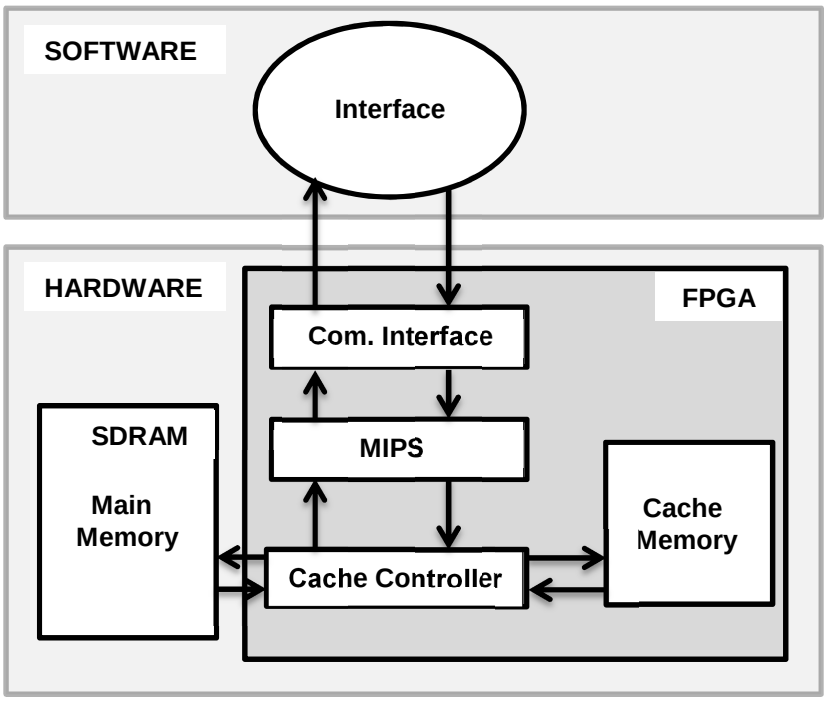

Figure 1: Organization of the system.

3.2 Implementation

The framework is based on a reconfigurable cache that was implemented in a FPGA platform equipped with a FPGA Spartan 3E from Xilinx [7]. Beyond the FPGA, this prototyping board also provides a SDRAM memory. It is a 512 Mbit Micron Technology DDR SDRAM [8] where the main memory of the system is operated.

The allowed number of ways that currently can be chosen is 1, 2, 4 or 8. However, other values can be considered. The capacity ranges from 4 Kbits to 16 Kbits and the reason for these values is the limited capacity of the BRAM (block RAM) memories of the FPGAs. BRAMS are used because they are small and fast, and moreover they exist on the FPGA without taking up logic resources. The limitation of this kind of memories is the restricted size that they can provide. All the combinations between cache capacity and blocks size are possible.

When a block must be replaced in an associative cache, two strategies can be used. The simple one consists in chose a block randomly. The other policy uses LRU (last recently used), where the block replaced is the one that has been unused for the longest time. In practice, LRU is too costly to implement for hierarchies with more than a small degree of associativity (two or four, typically), since tracking the usage information is costly. For larger associativity, either LRU is approximated or random replacement is used. Random replacement is simple to implement in hardware, and for a 2-way set-associative cache, random replacement has a miss rate about

10% higher than LRU replacement [1]. As the caches become larger, the miss rate for both replacement strategies falls, and the absolute difference becomes small. In fact, the random one can sometimes be better than the LRU.

Figure 2 shows a state diagram of this finite state machine used in the cache controller. At the beginning the controller waits for an instruction to read or write (load or store in MIPS) data. If the instruction executed is a load, the controller begins a set of steps. First, it has to do a comparison. If the address tag is equal to the cache tag, a read hit occurs. The value of the cache is used, then waits until the transaction is completed and return to idle state. If a read miss occurs, the data will be read from main memory, waits until the process is finished and returns to begin. This is a slower process than a read hit because the access time to the external SDRAM is greater than accessing internal BRAMs. If the instruction received is a store it is also necessary compare the tags. If the address tag is equal to the cache tag, a write hit occurs and a data is written into the cache and into the main memory. If a write miss occurs the data is written into the main memory.

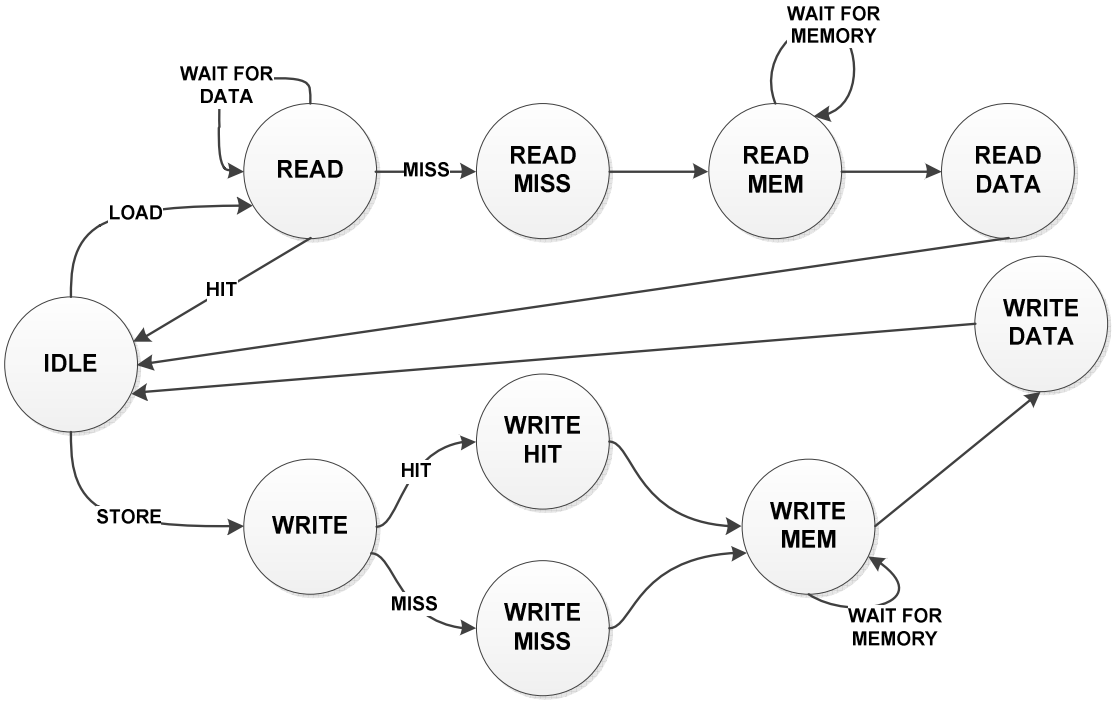

Figure 2: Finite state machine used in the cache controller.

All the hardware components were written in Verilog HDL and the platform interface was implemented in Java using the Eclipse Software Development Kit (SDK) [9].

Table 1 shows some statistics about the implementation of the whole system in the FPGA, comprising the blocks shown on figure 1 (communication interface, MIPS processor, cache controller and cache memory). The results are presented considering four degrees of associativity. For example, for the direct-mapped cache (1-way) the occupation of BRAMs are 20% and in the 8-way associative cache organization all the available BRAMs are used.

Table 1: FPGA usage statistics.

	Used				Utilization			
	1-way	2-way	4-way	8-way	1-way	2-way	4-way	8-way
Slices	2090	2167	2390	2780	22%	23%	26%	30%
Flip-Flops	1942	2009	2116	2210				
LUTs	4039	4200	4479	4812	43%	45%	48%	52%
BRAMs	4	6	10	20	20%	30%	50%	100%

4. Using the framework

The most part of this project is developed in hardware and that's why it is necessary to create a control interface. To facilitate the communication between the user and the platform the interface tries to be simple and user friendly. The interface is prepared to receive some input about the organization of the cache and allows the user oversee and analyze the given results (mainly hit and misses rate).

Following, some orientation lines to allow a good use of the framework are given. First, the user has to pick the inputs about the cache memory. It is possible to select the cache organization (direct-mapping, full associative, 2-way associative, 4-way associative or 8-way associative) and its size. Data used by a program should then be loaded to main memory. After that, the system is ready to runs a program. In the menu "Load" is possible to select predefined programs. The results after the execution of a program are displayed in different windows present in the operation interface. The process can be repeated for other organization allowing an user to understand how different cache affects the performance of the system when running of the system when running a program. Figure 3 shows the main window of the implemented interface.

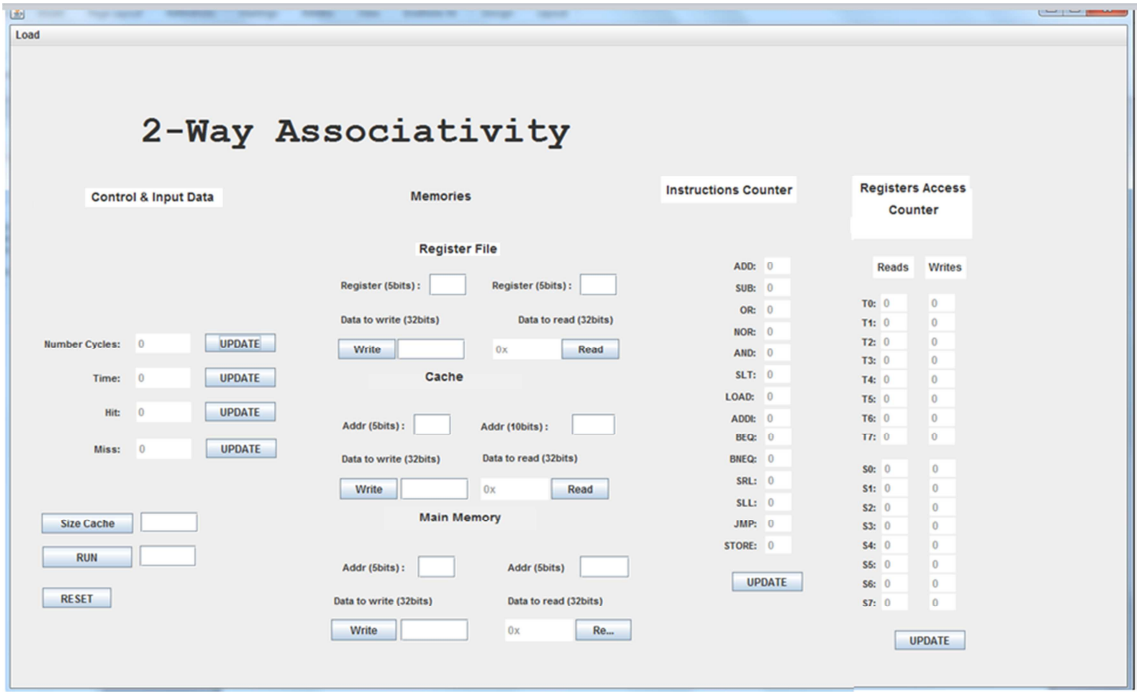

Figure 3: Interface window for a 2-way associative cache.

In order to demonstrate the use of this framework it was created a simple program to test and analyse the outcomes. The program was tested with different cache organizations and allowing collects results in terms of miss rate.

The program performs the sum of all the elements of a sequence of 50 integer values in main memory. Following is the assembly code of such program:

```
        addi  $s0, $zero, 50
        add   $s0, $zero, $zero
next:   lw    $s1, 0 ($s2)
        add   $s3, $s3, $s1
        addi  $s2, $s2, 4
        addi  $s0, $s0, -1
        beq   $s0, $zero, next
```

Before the execution of the program two tasks were needed: store the sequence of values to the main memory and initiate the cache memory with random values.

With this program, several tests were made, using different organizations (direct-mapped, 2-way associative and 4-way associative) and each one with different number of blocks (256, 128, 64).

Table 2: Miss rate occurred in a test program.

	direct-mapped	**2-way associative**	**4-way associative**
64	88%	80%	64%
128	94%	84%	78%
256	97%	94%	90%

Through these results it is possible to establish relations between cache organization and the resulting performance. The direct-mapped cache shows the worst results. In the opposite extreme, the 4-way associative cache presents the better results. Another aspect that is possible to verify is that, for all organizations, the increasing of cache size decreases the miss rate. These results are not surprise taking into account the considered program and the expected behaviour of the different caches. In fact, the presented case aims to concretize how the framework can be used.

The conclusions here exposed can certainly be different for other programs. That's why it is important to have a framework, like this one, to help understand how programs respond with different environment.

5. Conclusion

This work presented a framework allowing the analysis of the behavior of cache memories on a FPGA platform. The framework has a user-friendly interface that controls and supervises the entire system. Since the hardware is implemented on a FPGA based platform, some statistics on area and speed for different organizations were presented.

The paper showed the main features of a cache memory and the associated trade-offs in terms of its performance. The implementation of this framework allows to try and to study these relationships. That's why it can be advantageously used by students from an educational point of the view. Students could use it as a tool for testing the different theoretical aspects about cache memories in the regular courses on computer architecture. In this way, students can clarify possible doubts and so acquire a better understanding about these subjects. Due to the developed interface, this tool can be easily used to teach and learn about cache memories.

As future improvements it will be implemented some additional features. One of that is to integrate in the framework the ability to search and find automatically an optimal cache organization for a given set of programs considered representative of the kind of applications that a system will run. On the current version the users have to try different cache organization to conclude what is the better choice for each test program.

References

1. J. Hennessy and D. Patterson, *Computer Organization and Design - The Hardware/ Software Interface*, 4th. edition, Morgan Kaufmann, 2005.
2. J. Edler and M. D. Hill, “Dinero IV Trace-Driven Uniprocessor Cache Simulator”, (*http://www.cs.wisc.edu/~markhill/DineroIV*).
3. L. Null and K. Rao, “CAMERA: Introducing memory concepts via visualization”, SIGCSE Bulletin, Vol. 37, No. 1, 2005, pp. 96-100.
4. R. Quislant, E. Herruzo, O. Plata, J. Benavides and E. Zapata, “Teaching the Cache Memory System Using a Reconfigurable Approach“, *IEEE Transactions on Education*, Vol. 51, No. 3, August 2008, pp. 336-341.
5. “CASPR- A Configurable Assembler Program”, 2008, (*http://www.ele.uri.edu/~tparys/caspr/*)
6. J. Pereira, *Educational package based on the MIPS architecture for FPGA platforms*, M. S. thesis, University of Porto, Porto, Portugal, 2009.
7. Micron Technology, “MT46V32M16 DDR SDRAM Data Sheet”, 2000, (*http://download.micron.com/pdf/datasheets/dram/ddr/512MBDDRx4x8x16.pdf*).
8. Xilinx Corporation, “Spartan-3E FPGA Starter Kit Board User Guide”, (*http://www.xilinx.com/*).
9. The Eclipse Foundation, “Eclipse”, (*http://www.eclipse.org/*).