MEMORY ASPECTS OF DUAL CORE PROCESSOR DESIGN

Thiago Nunes Coelho Cardoso, Celina Gomes do Val,
José Augusto Nacif, Antônio Otávio Fernandes, Claudionor Nunes Coelho Jr.
Computer Science Department, Universidade Federal de Minas Gerais
{*thiagon,lina,jnacif,otavio,coelho*}*@dcc.ufmg.br*

ABSTRACT

Nowadays multi-core processors are being widely used to improve performance. In this paper a dual core MIPS32 is designed and implemented in Verilog. MIPS32 is a MIPS Technologies architecture used extensively in embedded solutions. A quick overview of the single core architecture is given. The design of the dual core architecture is then detailed focusing in the main challenges: memory access and cache coherence. In order to solve this problems a bus arbiter for simultaneous memory access and the write-invalidate cache coherence protocol are implemented. Simulation waveforms of processor behavior in important events are discussed. The synthesized processor fits Spartan 3E XC3S1200E using 8.67% of system gates.

1. INTRODUCTION

For decades the computer industry had basically used two techniques in order to improve the processors performance: The first is reducing the transistor size and the second is increasing its switching frequency. Shrinking transistors made possible not only increase the number of transistors in a chip, but also reach higher transistor switching frequency. Unfortunately this scaling brings an exponential growth in leakage power, which density began approaching the active power density. The problem is that this is a rigid limit since leakage power increase is the basic working principle of an MOS transistor [1]. While increasing processor frequency is getting harder, the ability of packing more transistors in each chip is still growing.

On the architectural layer, instruction level parallelism has also shown its limitations over the years, as at the program level, we have not seen recently greater performance improvements at the expense of large transistor counts. Due to this issues, modern processors are taking advantage of parallel architectures as can be seen in IBM Cell, a processor with 8 cores [2], in Intel Core 2 Quad [3] and AMD Phenom X4 Quad-Core, both with 4 complete execution cores [4].

Multi-core processors have the advantage to parallel process the same data set but this brings shared memory complications, since memory usually can only be accessed by a single core each time. Caches come as an optimization to reduce the number of memory access, the drawback is the cache coherence problem that is further explained.

In this paper a dual-core processor is designed and implemented focusing on this memory and caches aspects. It is structured as follows: Section 2 gives a MIPS32 quick overview; Section 3 presents the cache coherence and memory access problems and the architecture of the dual core MIPS; Section 4 discusses simulation and synthesis results; Section 5 presents conclusion and future work.

2. SINGLE CORE ARCHITECTURE

MIPS32 is a 32 bit RISC architecture widely used in embedded solutions such as Cisco routers, some SGI computers and video-games [5]. In this paper, the implemented processor is MIPS32 based with most relevant features like 5 depth pipeline, instruction and data caches and exception handling [6].

The processor is divided in the following main modules.

Bus: Responsible for connecting the processor caches to the main memory. Asserts that the memory will have only one access at a time.

Caches: Stores some requested memory data to take advantage of temporal and spatial location. Processor caches are described more detailed in sub-section 2.1.

Control: Set the necessary flags in order to execute the instruction correctly.

IF,ID,EX,MEM,WB: Pipeline stages, each module has the correspondent stage logic. *IF* fetches the next instruction, *ID* decodes it and set the necessary flags with control, *EX* is responsible for execution (ALU), *MEM* takes care of memory accesses and *WB* write the information to the registers.

IE Handler: Interruption and Exception handler, takes the proper action when an exception or interrupt is triggered. This action go from software routines to hardware interventions as resetting the processor.

2.1. Caches

The caches implemented in the processor are write-through non-allocate caches. When a store instruction is

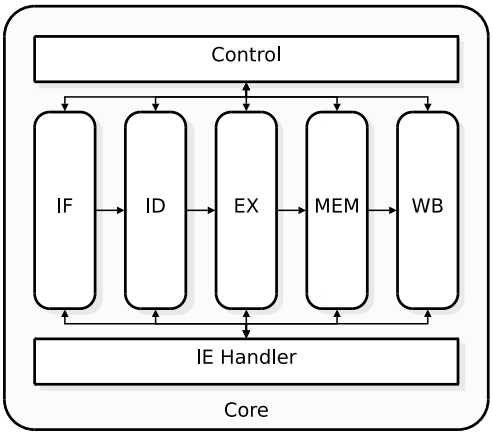

Figure 1. CPU Block Diagram

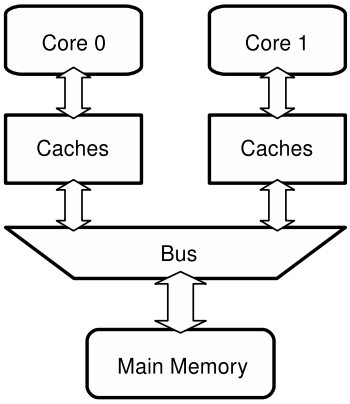

Figure 2. Processor memory architecture

executed, the data is written in the main memory and if the address is in cache its value is also updated.

Caches and memory were implemented in Verilog for simulation, for synthesis they are removed and external Static RAM is used for the caches and Dynamic RAM is used for the main memory.

3. DUAL CORE ARCHITECTURE

The single core MIPS32 can be adapted to be used in a dual core processor by taking especial care of the memory access and the cache coherence between the processors.

Both cores access the same main memory. This scheme bring some complications like sharing the memory access and maintaining coherence between processors caches [7].

3.1. Shared Memory Bus

The implemented processor is a centralized shared-memory multiprocessor. That is, all cores access the same main memory.

More than one core may need to access the memory at the same time but it can only answer to one query at once. So it is necessary to have a bus to control this simultaneous access.

Figure 2 presents the processor memory architecture.

In case of simultaneous access the bus is responsible for selecting one core to have the priority. This core can be selected by its id (the core with the smallest id has the higher priority) but this approach can lead to starvation of the other core. As an example if core 0 executes only load instructions and every one of them is responsible for a cache miss, if the core 1 needed to read data from memory, including instructions it would be locked forever.

A way to avoid this problem is to select the core in a pseudo-aleatory way. This can be done if the bus each clock cycle gives the priority to a different core, as in real applications memory access occurs in aleatory time, the priority will be given in an aleatory fashion.

If a processor is reading or writing, a flag is sent to the other processor making it wait until the bus is free.

3.2. Cache Coherence

In multi-processor design, a single cache bus is usually the system bottleneck. Reducing the use of bus is the key to achieve better performance. Better performance can also be reached if all processors have their own private cache [8].

With more than one core having its own cache, the same data can be stored in different places. One value that is changed by one cache should be changed in all other caches or there should be a mechanism to avoid wrong data use. This difficulty is known as the cache coherence problem.

The system must be coherent. A coherent system is the one where every read of memory data return its more recent value. In this paper this problem is a little simpler since the implemented processor has write-through caches (data is written in memory as soon as it is uploaded in cache) making the main memory always hold the correct value.

The problem is not completely solved because both cores can have the same address in their caches, if one modification is made by one core, the other one should be aware of it or should have its value updated.

Small-scale multiprocessors usually adopt hardware solutions for ensuring cache coherence. This solution is the cache coherence protocol. There are two types of protocols: Directory based and Snooping [9]. Directory based protocols stores the sharing status of a physical memory block in only one place, used for a larger number of processors. The Snooping protocol is the one used in this paper. In this protocol every caches snoops the bus to check if it has an invalid or old value.

If the Snooping protocol update values in the necessary caches when a value change is made it is said to be a *write update* or *write broadcast* protocol. The other type of snooping protocol is the *write invalidate* that is the one implemented for this paper.

The *write invalidate* protocol is the one that in every write to the memory invalidates the duplicates stored in other caches. If the core with the invalidated address try to read it, a cache miss will happen.

The Snooping protocol in this paper is implemented through the bus that connects all caches. All caches snoops this bus and invalidate broadcasted addresses. In this way

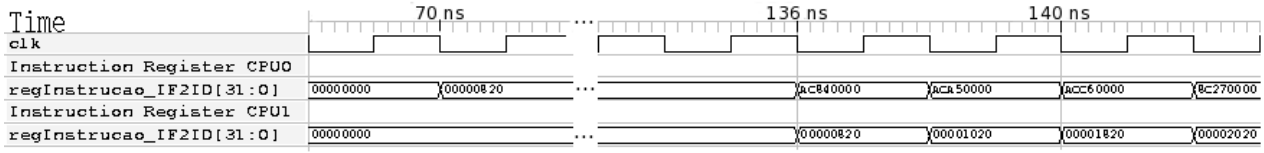

Figure 3. MIPS32 dual core normal execution

every cache is reported about data changes.

4. RESULTS

This section presents the results obtained by simulation and synthesis.

4.1. Simulation

Figure 3 is a waveform of the processor executing normally. At 70ns the first processor (CPU0) starts executing but the other one (CPU1) is still stalled. This happens due to the cache compulsory miss (the first time the data is read, in this case instructions). CPU1 then waits for the bus and at 136ns it starts executing concurrently to CPU0.

The test presented in figures 4 and 5 can summarize the dual core behavior of the bus and caches.

Figure 4 shows a moment after a cache miss in CPU0 (`miss_BUS0=1`), this indicates that the bus is used then to read data from memory. Data is being read when CPU1 executes an operation that leads to a memory access, being necessary to read the memory but the bus is already in use making it necessary to wait for its release (`waitDados_CPU1=1`).

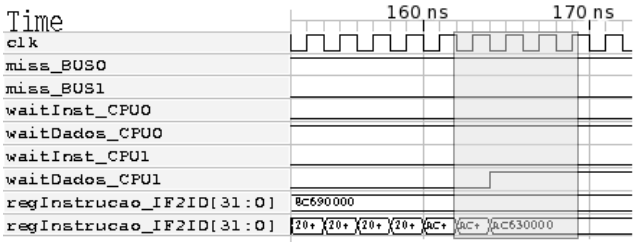

Figure 4. CPU1 tries to read during a CPU0 cache miss

Figure 5 highlights the moment when the bus is released and the instruction goes to the next pipeline stage (`regInstrucao_IF2ID=AC630000`) in a way that it can continue working normally.

4.2. Synthesis

The project was synthesized using the Xilinx ISE WebPack 9.2.04i [10] to Spartan3E XC3S1200E. Spartan3E is a series of low cost FPGAs. XC3S1200E FPGA provides 1.200.000 system gates.

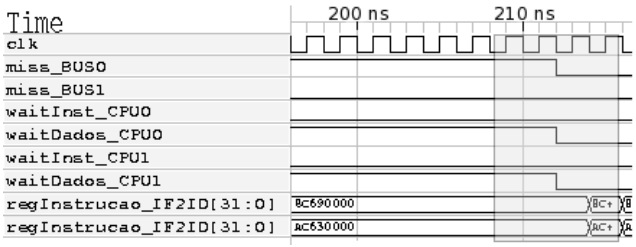

Figure 5. Bus release

Logic Utilization	Used	% Total
Number of Slice Flip-Flops	3,766	21%
Number of 4 Input LUTs	8,740	50%
Number of occupied Slices	5,720	65%
Total Number of 4 input LUTs	8,982	51%
Total equivalent gate count	104,072	8.67%

Table 1. Synthesis results

Table 1 presents the FPGA resources used.

Spartan 3 slices are composed of two Look-up Tables (LUT) function generators and two storage elements, along with additional logic. If one component of the slice is used, the whole slice is considered used, that justify the difference between slice utilization (in %) from system gate count (8.67%).

5. CONCLUSION AND FUTURE WORK

In this paper we designed and implemented a dual core MIPS32 processor. We discussed in more detail the simultaneous memory access and the snooping write-invalidate cache coherence protocol.

The discussed protocol is simple to implement and takes only a small part (898) of total system gates, proving to be a good candidate for simple dual core implementations. The main drawback of write-invalidate protocol is the fact that it doesn't scale for a large number of processors in a way that it can become a bottleneck [8, 11]. This happens because more cores will try to simultaneously access the memory in a way that processors will be in busy-wait until the bus is freed more frequently.

For future work the implemented protocol can be tested to check the increase in wait time caused by the growth in the number of cores. Other protocols can also be implemented, synthesized and compared to find a good trade-off in FPGA utilization and efficiency.

6. ACKNOWLEDGEMENTS

This work is supported by brazilian research agencies CNPq (grant PNM #141201/2005-3) and FAPEMIG (grant CEX-1485/06).

7. REFERENCES

[1] C. Johnson and J. Welser, "Future processors: flexible and modular," in *CODES+ISSS '05: Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, (New York, NY, USA), pp. 4–6, ACM, 2005.

[2] IBM Corp., "The Cell Architecture." Available at: http://domino.research.ibm.com/comm/research.nsf/pages/r.arch.innovation.html, 2005.

[3] Intel Corporation, "Product Brief: Intel Core 2 Quad." Available at: http://www.intel.com/portugues/products/processor/core2quad/index.htm, 2007.

[4] Advanced Micro Devices, Inc., "Key Architectural Features of AMD Phenom X4 Quad-Core Processors." Available at: http://vip.amd.com/us-en/Processors/ProductInformation/0,,30_118_15331_15332%5E15334,00.html, 2008.

[5] Mips Technologies Inc., "Mips Technologies Website." Available at: http://www.mips.com, 2008.

[6] MIPS Technologies, Inc., "MIPS32 Architecture For Programmers Volume II: The MIPS32 Instruction Set," 2005.

[7] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. 4 ed., 2007.

[8] J. Archibald and J.-L. Baer, "Cache coherence protocols: evaluation using a multiprocessor simulation model," *ACM Trans. Comput. Syst.*, vol. 4, no. 4, pp. 273–298, 1986.

[9] P. Stenström, "A Survey of Cache Coherence Schemes for Multiprocessors," *Computer*, vol. 23, no. 6, pp. 12–24, 1990.

[10] Xilinx, Inc., "Xilinx ISE WebPack." Available at: http://www.xilinx.com/ise/logic_design_prod/webpack.htm, 2008.

[11] T. Terasawa, S. Ogura, K. Inoue, and H. Amano, "A cache coherency protocol for multiprocessor chip," *Wafer Scale Integration, 1995. Proceedings., Seventh Annual IEEE International Conference on*, pp. 238–247, Jan 1995.