DLX HOTOKADA:
A Design and Implementation of a 32-Bit Dual Core Capable DLX Microprocessor with Single Level Cache

Darryl Aldrin M. Dioquino, Katrina Joy S. Rosario, Homer F. Supe, Jestoni V. Zarsuela
Anastacia P. Ballesil and Joy Alinda Reyes
{dmdioquino, ksrosario, hfsupe, jvzarsuela, apballesil, jpreyes1}@up.edu.ph
Intel Microprocessors Laboratory
Department of Electrical and Electronics Engineering
University of the Philippines-Diliman

***Abstract* — Data access in main memory units can be sufficient for processors but due to demands for faster computers nowadays, implementation of multiple cores as well as the usage of a cache to increase performance, are necessary. These two solutions were implemented using a 32-bit pipelined DLX microprocessor, resulting to a dual core capable (DCC) DLX with single-level cache in a Uniform Memory Access Architecture type. This project made use of the Shared Cache System divided into an Instruction Cache and a Data Cache to solve processor structural hazards due to coincident instruction and data access.**

The structural model was implemented using Verilog HDL and Synopsys® Design Tools in producing the gate level netlist and layout of the microprocessor in a 0.25µm CMOS process.

The implemented microprocessor was then characterized in terms of speed, area, and power consumption. The insertion of shared caches for the microprocessor enhanced system performance by an average of 80%. However, due to insertion of caches, the synthesized design drastically gained a 1420% increase in area and a 1370% in power compared to the dual core DLX processor without the cache.

I. INTRODUCTION

To meet the demands for faster and more powerful processing capabilities, engineers have explored several areas of computer architecture. One of them is multi core architecture wherein two processors are fabricated on a single chip to increase system performance by increasing the degree of parallelism between programs. Another approach is to use a smaller but faster memory (relative to the main memory) in order to achieve a greater average in serviced instructions per clock cycle. This smaller and faster memory is called a cache and the central idea of its design lies on the theory of *locality of reference*. Locality of reference manifests itself in two forms and its concept can be understood by these. The first form is *spatial locality* which states that if a program accesses a block of memory, say block *j* it will be likely accessing the adjacent block, either block *j+1* or block *j-1*. The second one is called *temporal locality*. The idea of temporal locality is that the blocks accessed by a program will be likely accessed again in the future. Therefore, since most programs display a high degree of locality of reference, inserting a cache in between processors and lower levels of memory is expected to reduce memory clock cycles and improve program execution speeds.

In the University of the Philippines Intel Microprocessors Laboratory, separate researches in the DLX Dual Core Microprocessors and Single-Level Caching were done by two groups of undergraduate researchers. Therefore this project aims to fuse the two researches by inserting a cache for the Dual Core Capable DLX Microprocessor and testing the design in terms of area, speed and power to see the improvement of inserting caches in dual core architectures.

A. DLX Microprocessor

The DLX (pronounced as "deluxe") is a five-stage pipelined processor developed by John L. Hennesy and David A. Patterson, with the purpose of using it as an instructional model for studying computer architecture.

DLX is a simplified version of the *MIPS* (Microprocessors without Interlocked Pipelined Stages). DLX is pipeline efficient and operates a simple load and store instruction set. It is also known as a *GPR* (General Purpose Register) machine containing thirty two 32-bit registers. The GPRs can be utilized as thirty two single-precision registers (32-bits) or as sixteen double-precision floating point registers (64-bit). DLX also uses byte-addressable memory accesses in *Big Endian* mode. It has 32 bits of address lines making it capable of addressing 2^{32} unique addresses in memory. DLX instructions are 32 bits in length and are assumed to be aligned (separated by wordlength addresses). GPR access can be made by bytes (8 bits), halfwords (16 bits) or words (32 bits).

B. Multicore Architectrures

Different multicore architectures exist and are each is suitable with different programs and applications. These architectures can be classified into two: *NUMA (Non-Uniform Memory Access)* and *UMA (Uniform Memory Access)* architectures. NUMA architectures have a local and remote memory for both processors. Local memory is memory "owned" by the respective processor and remote memory is memory nearer (and "owned by") to other processors. NUMA architectures have issues with memory data coherence since processors modify data on their own memory that is possibly being accessed in a different local memory. Hence, NUMA architectures are most compatible with programs that have high degrees of parallelism.

Uniform Memory Access (UMA), on the other hand, uses a global memory for all processors. Though lower level memory data coherence is resolved, the problem of bus traffic in the shared (global) memory arises. Thus, these architectures are suited for programs that have high data dependence and for multicores that are few in number (around 2 - 4).

C. Cache Algorithms and Policies

Cache placement policies can be categorized into three; *direct mapping, associative mapping* and *set-associative mapping.* Direct mapping allows certain blocks of memory to be mapped on a specified block in the cache. On the other hand, associative

978-1-4244-2182-4/08/$25.00 ©2008 IEEE.

mapping gives total freedom in mapping main memory blocks with cache blocks. In between them is set-associative mapping. Set-associative mapping maps memory data by sets (groups of data greater than or equal to one block) wherein it emulates the direct mapping scheme by allowing only certain locations to be mapped in a set. Inside the set however, placement of allowed memory blocks are totally free adapting for this case associative mapping.

There are also schemes devised for replacing blocks in the cache (i.e. the cache is full and data is needed to be stored there). Among them are *Random replacement*, *LRU (Least Recently Used) Algorithm* and *FIFO (First-in-First-Out) Algorithm*. As the word implies, Random Replacement randomly replaces a block without regard for anything except for the result it obtains by its *Random Generator Circuit* used to determine which block to replace. LRU algorithm however takes into account the history of the blocks accessed (either read or written on) in the cache and replaces the block that was not accessed for longest time. For the FIFO scheme, the block that gets to be replaced is the oldest block that was brought in the cache (not necessarily the least recently accessed).

For writing policies there are two common approaches, *Write Through* and *Write Back*. In Write Through algorithm, when data is written in the cache, it is reflected in lower memory (i.e. main memory) as well. On the other hand, with Write Back policy, writes are done only in the caches. Written data in the caches are only reflected in lower memory if the cache block that is written on is found to be the candidate for replacement in the cache.

II. METHODOLOGY

A shared memory architecture was assumed in the project and was also adapted in the design of the caches. Since there are only two cores, the shared memory and cache system was not expected to greatly hamper the performance of multi-core architectures with caches. Sharing memory data also removes the burden of data coherence brought about by designing local memories for both processors.

The group exploited the simplicity of the DLX processor's load and store instructions by designing two types of caches to be shared: the data and the instruction cache. Creating a separate memory for data and instructions reduces structural hazards of the Dual Core Capable DLX Microprocessor since it allows the access of instruction and data memory simultaneously. Bus arbiters were also added to facilitate memory access in shared environments. There were two levels of arbitration done in the project. The first one was the arbitration for sharing the caches. This was realized through the Instruction and Data Cache arbiters. The second arbitration was between the caches and the main memory. Since main memory is also global, a block was inserted to assist in the sharing of the caches with the main memory. Figure 1 shows the block diagram of the whole Dual Core Capable DLX with Single Level Cache.

A. Implementation

The group used Verilog HDL (Hardware Description Language) to code the designs for the project. Implementing the project started with coding the designs in Register Transfer Level (RTL). After which, tests were conducted to validate if the behavior of the project coincided with what was intended in the design. After RTL simulation was passed, the blocks were synthesized in order to map the design with standard cells and to produce the project's schematic. Next, the synthesized project was tested. In testing, timing constraints brought about by the mapped gates were introduced. After passing the test in synthesis level, the project commenced with placement and routing to produce and optimize the project's layout. This was facilitated using Astro® Place and Route Tool. The produced layout was again tested to validate the functionality of the project. Figure 2 illustrates the design methodology of the project.

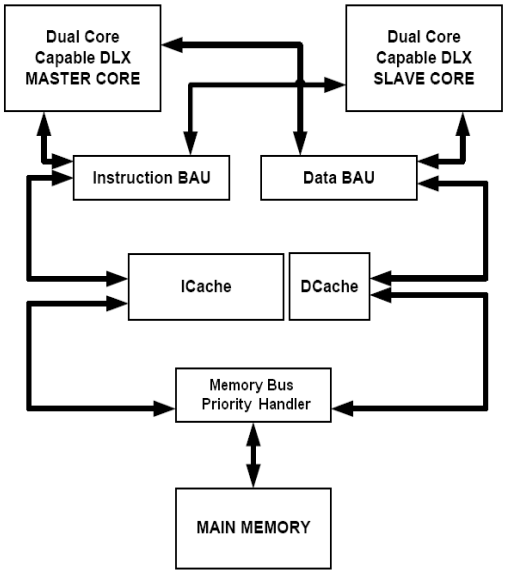

Figure 1. Block Diagram of HOTOKADA DLX

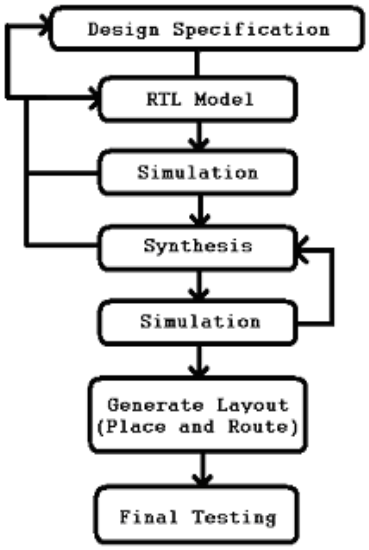

Figure 2. Design Methodolgy

B. Testing / Verification

Register Transfer Level Testing: The group members of the project were divided into two teams, a cache design group and a DLX microprocessor group. For both teams, several levels of testing were implemented starting from testing per block all the way to the final integration. Both design groups started with testing individual blocks. After verifying individual blocks, the whole cache system was tested against design blocks that emulated the processor's behavior. For the DLX processor, verification was done per pipeline stage, validating its functionality through the 5 pipeline stages of the DLX processor. After that stage was passed, the processor was tested against its instructions set. After verifying the processor against individual instructions, the processor's

functionality and correctness was validated against sample programs. Processor issues and hazards were also tested in the design. Cases of data forwarding, stalls and hazards were exhausted in order to affirm the correctness of the processor's behavior. Arbiter blocks were tested in parallel with the processors. Arbiter testing utilized dummy signals to verify and test the functionality of how they facilitate memory sharing. As mentioned, there were two arbiters in the project and so there were two tests of arbitration made. One was a test on the arbiter of the processors between the caches and the other one was a verification of the arbiter between the caches and the main memory. When all of these tests were done, the whole project was tested against benchmark programs like GCF, Multiplication and Factorial operations. Likewise thread synchronization tests were also done. To facilitate testing, two instantiations of the Dual Core Capable DLX were made, each representing a core. This was done for a better evaluation of how caches improve performance in dual core architectures in different types of processing scenarios like parallel programs and consumer-producer threads. Anticipating this convention, the arbiters of the processors to the caches were designed to switch between memory requests instead of making the processors communicate with themselves. This was to reduce the problem to single core memory accesses while stalling the other processor. That is, by switching between requests, the caches at any given time will see a single block accessing its ports.

Gate Level Testing / Netlist Simulation: The standard delay file (SDF) of the project was extracted at synthesis and was used in testing the Gate Level correctness of the project. The same tests in the Register Transfer Level were used to verify the functionality of the system against the SDF's introduced delays. The group took note of how the behavior deviated from the ideally timed RTL simulation and made the adjustments in the coding style of the project in order to meet timing constraints. Thus, the group passed Gate Level testing by repeatedly recoding the project, retesting and verifying its functionality against RTL and Gate Level simulations.

Layout Level Testing: This was the final level of testing done with the project. With additional routing delays from a new SDF file, the project was tested if it still held the correctness it had from the two previous tests. Results showed that the project was still correct. Also, acquisition of maximum allowable frequency of the project, area, and power consumption was done in Layout Level. For power consumption, *Synopsys*® Prime Power was used.

III. RESULTS AND ANALYSIS

Two configurations were designed by the group to compare the improvement in performance of the shared single-level cache for the Dual Core Capable DLX microprocessor. The first configuration was two DLX processors without a cache system – directly accessing main memory. The other configuration was the group's project the Dual Core Capable DLX microprocessor with a single-level cache. This configuration had a lower level of cache (L2 cache) before it accessed the main memory. *Synopsys* VCS was used to obtain the speed by which the two configurations finished instruction execution. *Synopsys* Design Vision and Prime Power were used to extract area and power respectively.

A. Speed

To facilitate testing 4 benchmark programs used: GCF, Factorial, Multiplication and Thread synchronization. Also, the main memory was given a latency of 5 clock cycles. An average of 80% speedup in program execution time between the base microprocessor and DLX HOTOKADA was achieved.

B. Power

Power consumption was measured using a sample program that evaluates 12!. DLX HOTOKADA's processor power consumption fares well with its predecessor (DLX 7). The discrepancy in power consumption however, can be attributed to the caches and the design modifications needed to integrate them with the processor. Figure 3 shows the distribution of power among DLX HOTOKADA's major blocks.

C. Area

One of the obvious drawbacks of inserting caches is the drastic increase in area. For DLX HOTOKADA, the caches consumed a combined percentage of 93%. The great disparity is presented by splitting the caches into two types and the amount of storage that they were designed to have. Likewise speed enhancing designs for the caches also increased the combined area for the project. Table 1 shows the percentage of the area among the major blocks of DLX HOTOKADA.

D. Cost-effectiveness of Cache Insertion

The insertion of caches increased program execution speed for the dual core DLX processor at the expense of greater power and area consumption. If speed, power and area are used as criteria in supporting caches for the dual core DLX, then the benefit in speed enhancement must be greater than the additional cost of power and area for the insertion of caches to be justified. Thus, the group tabulated the disparity in program execution (using the same 12! program) between DCC HOTOKADA with the main memory only and the DCC HOTOKADA with Cache (see Figure 4 and Figure 5). In general however, as the gap in main memory latency and cache latency increases, speed enhancement also increases. Equation (1) verifies this statement.

$$speed\ up = \frac{execution\,time\,without\,cache}{execution\,time\,with\,cache} \qquad (1)$$

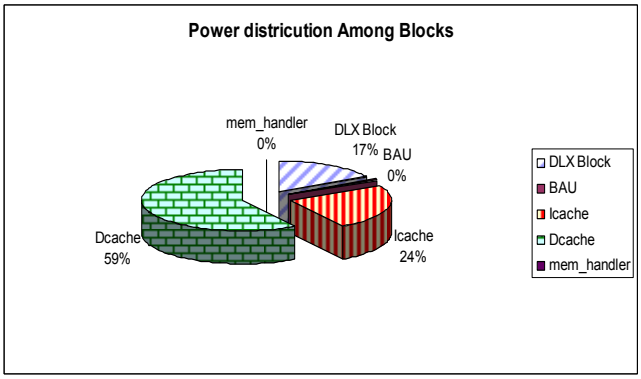

Figure 3. Power Distribution of DLX HOTOKADA

	AREA (sq. µm)	PERCENTAGE (%)
DCC DLX	804493.375	7.283108576
Bus Arbitration Units	39836.1289	0.360637963
Instruction Cache	2775755	25.1290137
Data Cache	7425932	67.22723979

Table 1 Area Occupied by Dual-Core Capable DLX with Single-Level Cache

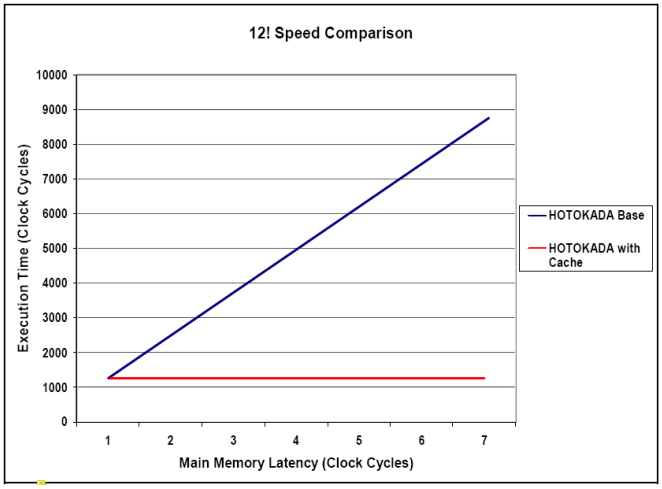

Figure 4. Comparison between the execution time of HOTOKADA Base Core and the HOTOKADA processor with cache memory (in clock cycles)

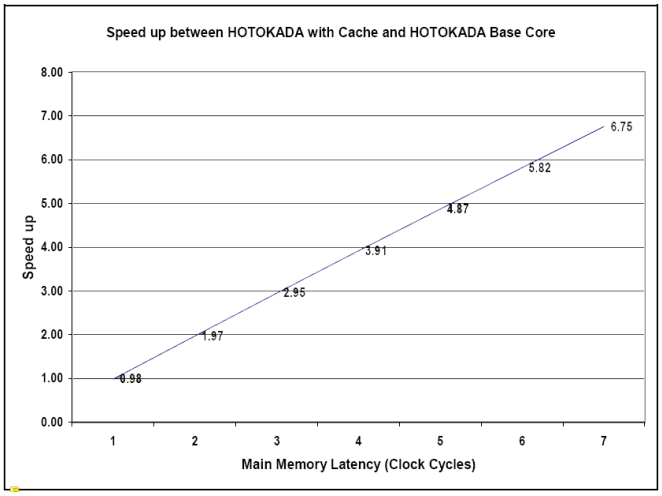

Figure 5. Speed up of inserting caches for the HOTOKADA DCC DLX (in clock cycles)

IV. CONCLUSION

The project was able to achieve its objective by increasing program execution speed for a dual core capable DLX microprocessor by as much as 80%. However, an insertion of cache memory system contributed to a larger chip area as much as 1,420 % increase in total area. Also, the project resulted in higher power consumption due to additional blocks needed to support the insertion of caches.

The cache system improved program execution speed as expected but there are still a lot of enhancements that can be done. First the architecture of the caches can be changed to a shared L1 cache to a separate L1 cache per core. Though the cache system was split, (data and instruction cache) the cache system itself was still shared. There was still a bottleneck since the processors were not able to maximize the high degree of parallelism presented by the cache's locality of reference and the purpose of dual-core architecture itself.

V. RECOMMENDATIONS

Future researches may want to explore a Dual-Core DLX microprocessor with a separate cache per core (dedicated cache) as their next project. Though they will be dealing with more issues like snooping and coherence of data they will be resolving others like bus traffic brought about by cache sharing. Future researchers are also encouraged to extend the project to greater number of cores and to improve the processor's design, increasing its speed in executing instructions and handling hazards (i.e. data forwarding). Other cache algorithms may also be tried (i.e. random replacement instead of LRU policy). Moreover, future researchers may also explore minimizing area and power for the current project. As always, area and power will be an issue in microprocessors especially as they get more powerful and minimizing them in microprocessor designs saves money and resources for all parties (customers, producers) involved.

REFERENCES

[1] Motlagh, DeMara, "Memory Latency in Distributed Shared-Memory Multiprocessors", IEEE, 1998.

[2] C. Hamacher, Z. Vranesic and S. Zaky, "Computer Organization", 5th edition, New York: McGraw-Hill, 2002, pp. 314-337

[3] D. Ancajas, E. Opelinia, A. Sepillo, W. Sumalia and W. Tan, "Dual Core Capability of 32-bit DLX Microprocessors", University of the Philippines, Diliman, Department of Electrical and Electronics Engineering, October 2006, unpublished

[4] J. Bautista, G. Bolivar, J. Ingking and M. Laviete, "High-Level Design Implementation and Characterization of a 32-Bit 5-Stage Pipelined DLX Microprocessor with Single Level Cache", University of the Philippines, Diliman, Department of Electrical and Electronics Engineering, October 2005, unpublished

[5] A.J. Smith, "Cache Memories", *Computing Surveys,* Vol.14, No.3, pp. 473-530, September 1982

[6] R.L. Lee, P. Yew and D. Lawrie, "Multiprocessor Cache Design Considerations", Center for Computer Research and Development, University of Illinois at Urbana-Champaign, 1987

[7] J.L. Henessy and D.A. Patterson, "Computer Organization and Design The Hardware and Software Interface", 2nd edition, San Francisco, California, Morgan Kaufmann Publishers, Inc.

[8] D. Hovemeyer, W. Pugh and J. Spacco, "Atomic Instructions in Java", Department of Computer Science, University of Maryland, College Park, MD 20742 USA