Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 6.017

IJCSMC, Vol. 6, Issue. 6, June 2017, pg.224 – 227

Enhancing the Performance of Direct Mapped Cache with Victim Cache using Structural Programming with only Nand Gates

Shravan.B.K.[1], Mahesh.B.Neelagar[2]
[1]Department of VLSI and Embedded Systems, Centre for PG Studies, VTU Belagavi, India
[2]Assistant Professor Department of VLSI and Embedded Systems, Centre for PG Studies, VTU Belagavi, India
[1] shravankoti2@gmail.com ; [2] neelagarmahesh@gmail.com

Abstract— Victim cache is a small cache which is used in conjunction with the primary cache to enhance the performance of the primary cache. This was proposed by Jouppi. This victim cache holds the data that are swapped out of the primary cache with the hope of this data being accessed in the next few clock cycles based on temporal and spatial locality. The victim cache used here is fully associative and follows pseudo LRU algorithm along with a direct mapped primary cache. The small 4 line victim cache of size 128B complements the 1KB primary cache which is present in conjunction to the 32KB main memory. The victim cache works on the pseudo least recently used algorithm. It swaps out data which is least recently used. The primary cache is direct mapped and thus it swaps out that particular line of data which is replaced by the data coming in from the main memory. This design is for a 32 bit processor to enhance its performance by reducing the timing delay as well as to reduce the silicon area which is illustrated in the results section.

Keywords— direct mapped cache, fully associative cache, Pseudo LRU, Instruction cache, victim cache

I. INTRODUCTION

Cache memory is the fastest memory in the hierarchy of memories. It is located closest to the CPU. The data is moved from the main memory to the cache which is accessed by the CPU. There are 2 types of cache memory one being the instruction cache and the other being the data cache. In this project we have dealt with only the instruction cache. The aim of the project was to enhance the performance of the instruction cache with the help of a concept called as victim cache which is smaller in size than the primary cache. The coding was done in Verilog with the help of Xilinx tool. The victim cache is smaller in size than the primary cache. The size of the main memory is 32kb. The size of the primary cache is 1kb. The aim was to design a 128 byte victim cache line for a 32 bit processor using structural programming using only NAND gates.

The cache used is a direct mapped cache which means a line in the memory can be moved to only one location in the cache. There are 2 other types of caches such as set associative and fully associative cache. In these kinds of caches we can move the data to any location in the cache from any location in the main memory. But there are extra bits to remember which is the least recently used (LRU) data. In our project we assume that the data access from the cache memory take one clock cycle whereas that from the main memory takes 5 clock cycles.

The victim cache used has is fully associative in nature. The swapping of data between the victim cache and the primary cache takes place within one clock cycle. The concept of designing with only NAND gates was

used to make the design easy understandable so that we get to know what happens when the design is synthesized and what are the internal contents of the blocks. The Nand gate was used as it is an universal gate. The design also included using of tristate buffers and D flip flops. These were the basic entities which were used to design the further blocks such as decoders, muxes,… The cache had instructions coming in from the instruction fetch unit and if there was a hit in the cache the data was moved out within one clock cycle and if it was not found then we waited for 4 clock cycles to access the data from the main memory. Effectively by using this victim cache approach theoretically we saved 4 clock cycles each time the data was accessed from either the primary cache or the victim cache thus increasing the speed of execution.

II. LITERATURE SURVEY

Norman.P.Jouppi proposed the technique of victim caching as an improvement over miss caching. According to him miss caching was effective technique at smaller sizes between 1K to 8K bytes. They effectively remove miss conflicts. As the percentage of miss conflict increases the size of the miss cache has to be increased to effectively deal with the misses. This is effective till 8K. But beyond that according to Jouppi it is better to save the miss in a victim cache. In the case of a miss the data which is removed from primary cache the data is saved in a smaller cache called victim cache.

He compared the performances of both victim caching and stream buffers. In case of victim caching a particular cache line is saved when it is removed from the primary cache with the hope of it being used again. While in case of stream buffers the data is prefetched with the hope of it being used. The two techniques create improvement in the performance of the CPU by different percentages. But when the 2 techniques are used together they improve the performance 143% according to Jouppi. By adding a small amount of hardware the conflict misses and the compulsory misses are brought down by a large amount. We have used the technique of victim cache in our project here with the aim of improving the speed of the processor by four clock cycles. He said that his approach was useful for cache sizes between 32-64 KB. [2]

In another paper by Hal Wasserman, based on the observations of Jouppi he tried to check the improvement in chip area with the help of victim cache approach. They observed that the improvement in performance was better at 32KB as against the finding of Jouppi of 4KB. They found that the victim cache size to be good at one fourth of the size of the primary cache. It worked well at 32KB rather than at 64KB.

Another factor that was improved during this experiment was the improvement in chip area Instead of adding another level of cache we can instead add a victim cache which saves the area by about 2 times. In this paper they have also discussed about the timing problems which are created due to the addition of victim cache approach. In my project I have only dealt with improvement in timing i.e. speed and not considered timing problems. The size of the cache selected is 1KB and the victim cache size is 128B. [1]

III. METHODOLOGY

Coding of the project is divided into different parts which will be later integrated using a main program.

A. BLOCK DIAGRAM

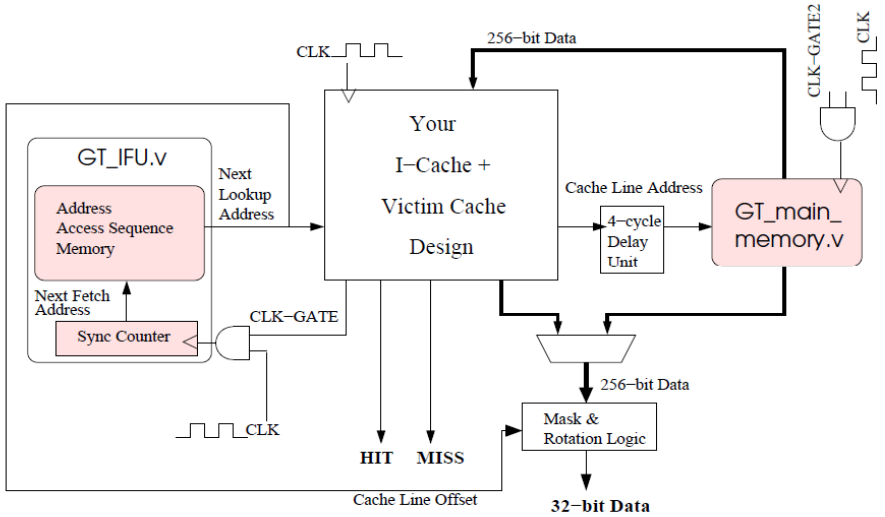

Fig 1 Block diagram of the Instruction cache with the Victim cache

B. DESIGN OF THE INSTRUCTION FETCH UNIT

The instruction fetch unit is the one which sends out instructions at each clock cycle. It is controlled by a gating signal which is coming from the control block. This is to ensure that the instructions are not sent out when the instruction is being fetched from the main memory. It ensures that the 4 clock cycle delay is executed without any problems. When there is no gating signal the instruction fetch unit sends out instructions. The instructions are initialized to different values for the purpose of simulation. The counter generates a signal which acts as a clock for the Instruction fetch unit which then drives out an instruction.

C. DESIGN OF INSTRUCTION CACHE

The instruction cache design is divided into sub modules such as instruction decoder, instruction control unit, instruction cache, tag comparator unit and instruction cache/victim cache selection module.

The instruction decoder takes 5 bits from the instruction stream and generates the line address in which the data is present. The least significant 4 bits selects which 4 byte data to be selected from a line in the cache. The next 5 bits in the instruction stream is used to select which line of the cache has to be selected which is of interest to the decoder unit.

The instruction cache by itself has 32 address lines and 279 data lines. 256 bits are the actual data whereas the other 23 bits are the tag address that is associated with the instruction to indicate the location in the main memory from where the data has arrived. The cache is built using a cache bit unit. The control signals here are cache enable which enables or disables the cache. Another signal read/write bar is used to either read from the cache if the read signal is 1 and to write to it if the signal is zero. Using this cache bit unit a byte unit is created having 8 cache bit units. Using 4 of these 8 bit units a 32 bit unit is created, which in turn is used to create a 256 bit unit to create a cache line. 32 such lines are created to create the instruction cache. The tag comparator unit is of 23 bits which compares the 23 tag bits of cache (257:278) with that of the instruction stream bit (31:10) to check in which line of the cache the data is present.

Then there is the control unit for the instruction cache. It generates control signals. If ic_hit=1 and if the clock delay is zero do nothing as the data is found. If the ic_hit=0 and vc_hit=1 then swap the data between the 2 caches. Also drive out data from the victim cache in the same cycle. If both hit signals are zero then gate the instruction fetch unit and enable the memory unit. Also start a counter to count the clock delay. At count of 3 write data from instruction cache to the victim cache so that the data is not over written. At count of 4 write the data from the memory to the instruction cache. Enable the instruction fetch unit and disable the memory unit. Also set the counter back to zero.

There is another sub module which selects either the instruction cache or the victim cache by using a 256 bit mux. There is a signal called c_m_select which when 1 selects the instruction cache and when zero selects the victim cache. Ic_hit signal is assigned to mux1_select to help select the proper cache.

The mask and rotation logic is used to drive out the relevant 32 bits out of the 256 bits in a line of cache. It masks all the irrelevant data by anding them with zero. The data is rotated to the left by as many bits and the 32 bit data is driven out.

D. DESIGN OF VICTIM CACHE

It is a smaller cache which is placed in conjunction with the primary cache. It is one eighth in size when compared with the primary cache. It has a total of 283 bits in one line of the cache which is divided into 256 bits of data, 23 bits of tag and 3 bits for the pseudo least recently used algorithm. It is a fully associative cache where in any data can be moved to any location unlike the direct mapped cache. It uses the pseudo LRU algorithm instead of the normal LRU algorithm. It uses 3 bits namely plru[2], plru[1] and plru[0]. The MSB bit is used to select which line the data is present whether in the 0^{th} and 1^{st} line group or the 2^{nd} and 3^{rd} line group. The plru[1] decides whether the data is in the 0^{th} or 1^{st} line and plru[0] decides whether it is in 2^{nd} or 3^{rd} line of the victim cache.

E. DESIGN OF MAIN MEMORY

It is 32KB (1024x256) in size. It has 10 address lines and 256 data lines. It is created and initialized with random values. It is enabled and disabled by a clock gating signal. Whenever data is found in either of the caches we gate the main memory so that vital clock cycles are saved. When the data is not found in both the caches then the data is taken from the main memory and after a delay of 4 clock cycles, generated by a counter, and written to the instruction cache and in the same time it is moved out of the mask and rotation unit.

IV. CONCLUSIONS

The project to enhance the performance in direct mapped cache with the assistance of victim cache has been completed using Xilinx ISE tool. The simulation of the main program was successfully carried out for the example sample of instructions taken up. There was an increase in speed as the total time of execution was reduced from 545ns to 385ns after the initial time taken to populate the cache. Theoretically each time the data

is accessed from the victim cache instead of the main memory 4 clock cycles are saved and hence a speed increase of 4 times is achieved.

The silicon area is also increased as the cache used is 1 KB (primary cache) plus 128B (victim cache) instead of using a 2 KB primary cache. The cache size proposed by Jouppi was one fourth of the size of the primary memory, but in this project we have tried to enhance the performance by using a victim cache which is only one eighth the size of the primary cache.

There is further scope for research which includes using victim cache at multiple levels of caches. We can also try to use level 3 or 4 caches as victim cache of the immediate higher level of cache. The direct mapped primary cache may also be replaced with a fully associative cache to further enhance the performance. The performance increase for a direct mapped level 1 cache with the help of victim cache approach has been completed.

ACKNOWLEDGEMENT

I have taken efforts in this Project report. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to Mr. Mahesh.B.Neelagar for his guidance and constant supervision as well as for providing necessary information regarding the Project & also for his support in completing the Project.

I am thankful to the external guide and the staff at KuVi Innovations Pvt. Ltd. for their constant support and encouragement.

I also thank our H.O.D. of VLSI and Embedded Systems Department Dr.T.C.Thanuja for providing her valuable suggestions and constant encouragement in completing the Internship.

I would like to express my gratitude towards my parents & members of VISVESVARAYYA TECHNOLOGICAL UNIVERSITY, BELGAVI for their kind co-operation and encouragement which helped me in completion of this Internship.

REFERENCES

[1] Hal Wasserman, *Victim Caching for Large Caches and Modern Workloads*, Univ. Of California, Berkley, Tech Rep, Spring 1996.

[2] Norman P. Jouppi, *Improving Direct Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers,* Digital Equipment Corporation Westem Research Lab, Palo Alto, CA.

[3] Digital Equipment Corporation, Inc, *VAX Hardware Handbook, volume I* – 1984, Maynard, Massachusetts.

[4] Farrens, Matthew K., and Pleszkun, Andrew R, *Improving Performance of Small On-Chip Instruction Caches*, The 16th Annual Symposium on Computer Architecture, IEEE Computer Society Press, May, 1999.

[5] Hill, Mark D, *Aspects of Cache Memory and Instruction Buffer Performance***,** Ph.D. Th., University of Califomia, Berkeley, 2007.

[6] Jouppi, Norman P., and Wall, David W, *Available Instruction-Level Parallelism For Super pipelined and Superscalar Machines*, Third International Conference on Architectural Support for Programming Languages and Operating Systems, IEEE Computer Society Press, April 2015.

[7] Jouppi, Norman P, *Architectural and Organizational Tradeoffs in the Design of the MultiTitan CPU*, The 16th Annual Symposium on Computer Architecture, JEEE Computer Society Press, May, 2012, pp. 281-289.

[8] Nielsen, Michael J. K., *Titan System Manual*, Tech. Rept. 86/1, Digital Equipment Corporation Westem Research Laboratory, September, 1986.

[9] Ousterhout, John, *Why Aren't Operating Systems Getting Faster* ***As*** *Fast As Hardware?*, Tech. Rept. Technote 11, Digital Equipment Corporation Westem Research Laboratory, October, 2014.

