ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

IJESRT
INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

HIGH THROUGHPUT CACHE CONTROLLER USING VHDL & IT'S FPGA IMPLEMENTATION

Khushbu Lalwani*[1] & Mayuri Chawla[2]
*[1]Department of ETC, RTMN University, NAGPUR
[2]Assistant Prof. Department of ETC, RTMN University, NAGPUR

DOI: 10.5281/zenodo.1012539

ABSTRACT

The world is now using multicore processors for development, research or real-time device purposes as they provide a better processing leading to better performance. This has attracted a number of researchers as the processors are embedded on a single chip and the performance can be easily amplified by saving the space, reducing power consumption, reducing the delay of the system. In the past years, a lot of technique shave emerged which proposes the optimization. One such important scope of optimization is the cache handling which has a considerable effect on the power, performance, and area of the processor. The problem with the cache which has to be overcome is the coherence which needs to be implementing to design a suitable and efficient technique. In this paper a new concept from which the cache controller is designed using the least recently used (LRU) cache replacement policy

KEYWORDS: Multicore Processor, Cache, Cache Optimization, LRU.

I. INTRODUCTION

The most recently utilized information or directions are kept in the cache with the goal that the data can be fetched at a quick rate enhancing the performance of the device. There is numerous level of cache with the last level being large and slow with respect to the primary level cache being quick and small. In the majority of the processor, initial level cache (L1) live in the processor and second level cache (L2) and third level cache (L3) are on a particular chip [1]. In multi-core processors, every processor has its own L1 cache. While last level cache is being shared by users of the cores [3]. The clock of the processor is a few hundred times speedier than the dormancy of fundamental memory [2].

Cache gives the support of diminishing this gap and it will improve the performance of the framework. A cache miss is the inability to locate the required direction or information in the cache. In that event when the information is not found in the cache then it would be brought into the cache from fundamental memory. A memory chain of importance appears in Figure 1 which was fetched [4]. For most of the part, cache misses are of three types: i) mandatory misses which happen when a memory area is fetched for the very first time, ii) conflict misses which happen because of lacking space when two squares are mapped in a similar area and iii) limit misses happens because of little space as cache can't hold each and every piece that we need to utilize.

Cache miss rate and Miss Penalty are the two main considerations which impact cache performance. The time required to deal with a cache miss is called cache penalty [5]. There are various techniques to diminish cache miss rates which incorporate casualty cache which is an area for impermanent of a cache line which is abolished from cache [6][7], Column cooperative caches lessen miss rate of direct mapped caches, prefetching of cache lines. Caches misses can be dodged by understanding the elements which can bring about misses and afterward they can be expelled by the software engineers from their applications utilizing distinctive CPU profilers and by adjusting and redesigning of the information. Cache penalty can be lessened by utilizing multi-level cache. Cache performance can likewise be enhanced by decreasing cache hit time. In the multi-level cache, more often the main level is kept small and speedier while the second level is bigger and much slower than the first level. On the off chance that we increment the cache pipeline stages, we can diminish the crevice between cache survey time and processor process duration.

There are different mapping techniques present which help data from main memory in mapping it to the cache which is used by the processor which have a direct impact on speed and performance.

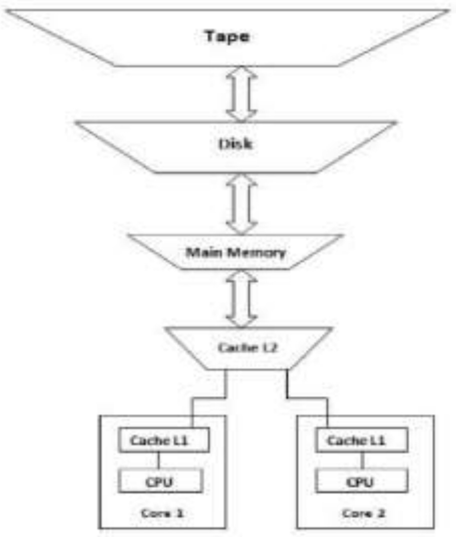

Figure 1: Multi-Core Memory Hierarchy

In this paper, we are going to demonstrate an LRU based technique for cache optimization. The next section of the paper illustrates the related work following with the proposed technique, setup, and results. Lastly, conclusion concludes the paper.

II. LITERATURE REVIEW

The performance of the CPU and main memory is enhanced using cache memory. Approximately 43% of total processing power is consumed by caches. A lot of various techniques have been proposed for energy efficient caches [5-10]. The articles studied are majorly based on various cache parameters in single-core architectures. Whereas there is no suitable solution for multicore cache memory.

A tradeoff between the complex cache and the power utilization by the cache is contemplated in [11]. An execution and power investigation strategy is displayed in [12] for multi-core frameworks. It doesn't address the issues in regards to various cache associations. A systematic model is created in [13] for power estimation and memory gets to time. This model is not reasonable for multi-core frameworks. A power-aware guideline cache is acquainted in [14] with lessening dynamic vitality utilization in the direction cache yet is not reasonable for multi-core frameworks with various coaches at various levels.

III. LEAST RECENTLY USED (LRU)

The references of the cache that were accessed in the recent past can be accessed again in near future. Once the cache memory is accessed it will access again which gives rise to the least recently used (LRU) algorithm.

The LRU monitors how a page has been utilized and keeping the pages that reference is occurring the most in the operations. The LRU approach depends on the rule of the area which expresses that program and information references inside a procedure tend to group. The Least Recently Used trade strategy chooses that cache memory for substitution which has not been referenced for a very long time. For quite a while, LRU was thought to be an ideal online strategy. The issue with this approach is the trouble in execution. One approach is to tag each cache part with its last reference; this would need to be done at every memory reference, for both direction, and information.

Time		1	2	3	4	5	6	7	8	9	10
Requests		c	a	d	b	E	b	a	b	c	d
0	a	a	a	a	a	A	a	a	a	a	a
1	b	b	b	b	b	B	b	b	b	b	b
2	c	c	c	c	c	E	e	e	e	e	d
3	d	d	d	d	d	D	d	d	d	c	c

Time Page last used: a = 2, b = 4, c = 1, d =3. The request for memory, a = 7, b = 8, e = 5, c = 9, d =3.

There are a few rules for LRU which starts with, the memory assigned in the cache for reference is assigned to a specific age and reference bit too. The age bit keeps the track of time the memory is occupied in the cache. The third rule is that the value of age bit should be between numbers of frames n to 2n in memory. When a memory is occupied the correspondent age bit is decremented by some constant number. When a hit occurs the age bit will be reset to the initial value.

The seventh rule is that when the memory is full the reference and age bit are checked and if there is a bit not recently used it is changed. The memory part is replaced with the oldest part present in the system which is not required in near future. If the reference bit is set to 1 then it is not chosen for replacement as it will be accessed in near future. Lastly, if all the reference bit are set to 1 then the oldest memory part is replaced.

The LRU is totally dependent upon hardware assistance. For a perfect LRU keep a time stamp for each page with the time of the last success instruction. If yes then throw out the LRU instruction. The problem with this part is that, since the OS is performing the LRU while changing the OS have to check all the instructions which is a lot expensive. A list of all the instructions is recommended to be made where the list starts with the most used instructions and end with the least recently used instructions. If the instruction of the end list is accessed then it is to be adjusted accordingly. This operation is also very expensive for the system. On each access, the hardware sets the reference to 1 also, it is set to 0 with varying time and LRU. There are also additional bits maintained, around 8 more bits are maintained. When the memory is accessed, shift the byte right, placing a 0 in the high order. Whereas when a fault occurs the lowest numbered page is kicked out.

IV. PROPOSED CACHE

To optimize the cache we are going to propose a cache using least recently used (LRU) method. The figure below shows the basic processor architecture:

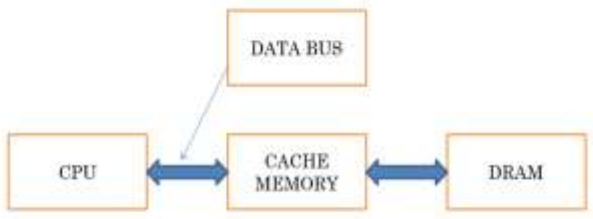

Figure 2: Block Diagram of basic processor architecture with cache memory.

The cache is implemented using VHDL, and is made with four important parts first is the RAM, following with the cache, Cache Controller and the interface as shown in the figure below. As observed from the above figure the cache is supported by RAM and Cache controller.

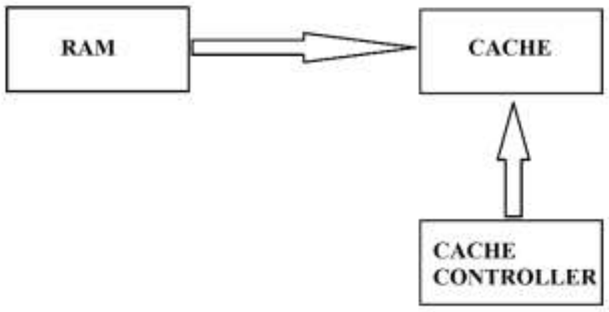

Figure 3: Cache Architecture

The cache controller helps the cache to communicate with the main memory. The controller takes the input and selects the data from the memory and transfers it to the cache for the system to use. The RAM is used for knowing which instruction is being used for processing. The Ram, Controller, and cache make a good cache. The diagram below illustrates the modules of the cache designed:

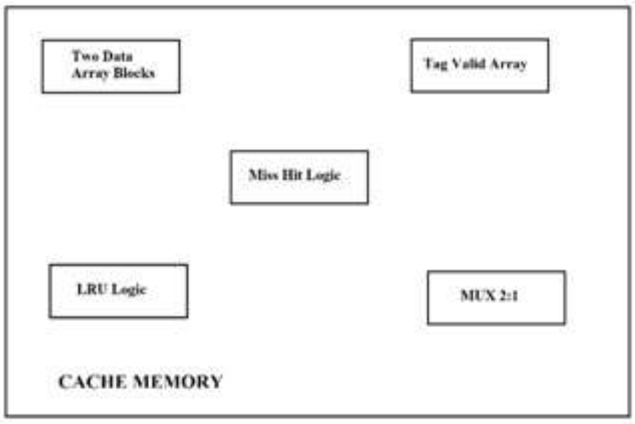

Figure 4: Cache Memory Modules

As observed from the above figure the cache contains five major modules:

a. LRU Logic: This module contains the LRU logic which is explained in the paper above, the LRU takes care of the performance and the area efficiency of the cache.
b. Miss Hit Logic: when the requested data is available in the cache it is called hit and when the data is not found it is called miss. Therefore, a miss and hit logic are required for the cache.
c. Tag Valid Array: this is used for accessing the data from the data array and maintaining the bits. For example, the age bit and referenced bit are held for accessing the data.
d. Two Data Array Blocks: the data array contains the data which is stored for faster access. In this system, there will be two data arrays which will enhance the memory power of the designed cache.
e. MUX: the MUX is used to select the proper line for reading and write operation of the data.

V. RESULTS

In this section of paper, we are going to illustrate the observations recorded. The RAM size created is about 1KWord, whereas the word size is about32 bits. The Cache Block Size designed is of 64 Words. The number of cache blocks designed is 2.

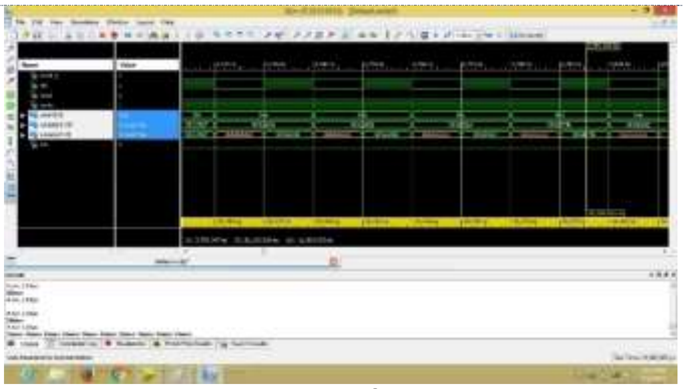
Figure 5: Simulation of proposed system

As observed by the above the above figure illustrates the simulation of the system designed. The operations performed are addr[9:0], wrdata[31:0] and rddata[31:0], with values 0bd, 003d873b and 003d873b respectively.

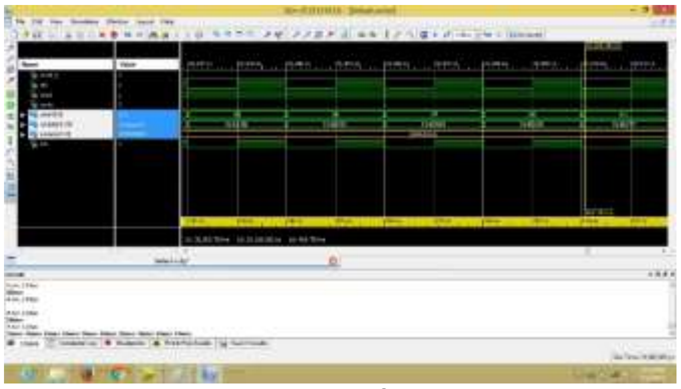
Figure 6: Simulation 2 of proposed system

As observed by the above the above figure illustrates the simulation of the system designed. The operations performed are addr[9:0], wrdata[31:0] and rddata[31:0], with values 001, 014daf57 and UUUUUUUU respectively.

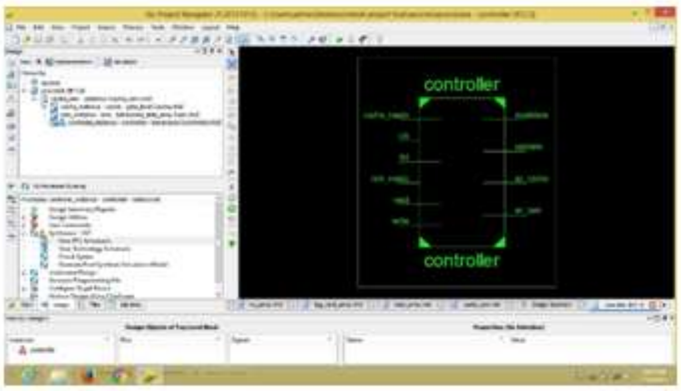

Figure 7: RTL of Cache Controller

As observed from the above figure, it shows the RTL of the controller with pins cache_ready, clk, hit, ram_ready, read, write, invalidate, validate, wr_cache and wr_ram.

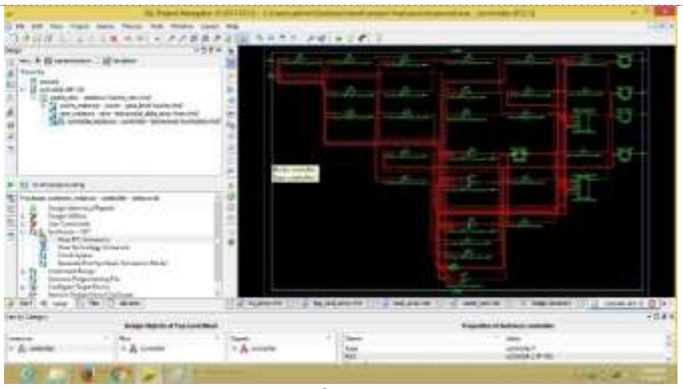
Figure 8: RTL of the Cache Designed

As shown in the above figure it shows the cache design.

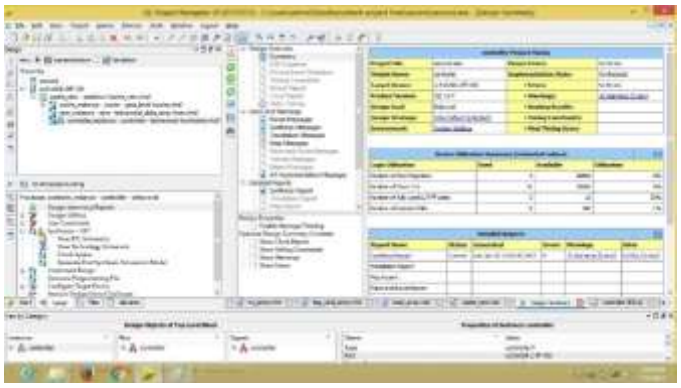
Figure 9: Area saved by the proposed technique

The figure above shows the area saved by the proposed algorithm the detailed device utilization is shown in the following table:

Table 1: Device utilization Summary

Logic Utilization	Used	Available	Utilization
Number of slice registers	3	28800	0%
Number of Slice LUTs	10	28800	0%
Number of fully used LUT-FF pairs	3	10	30%
Number of bounded IOBs	8	480	1%

As observed from the above table the number of slice registers and number of lice LUT are utilized less than 1%. The number of fully used LUT-FF pairs are utilized 30% and finally, the number of bounded IOBs is used 1%.

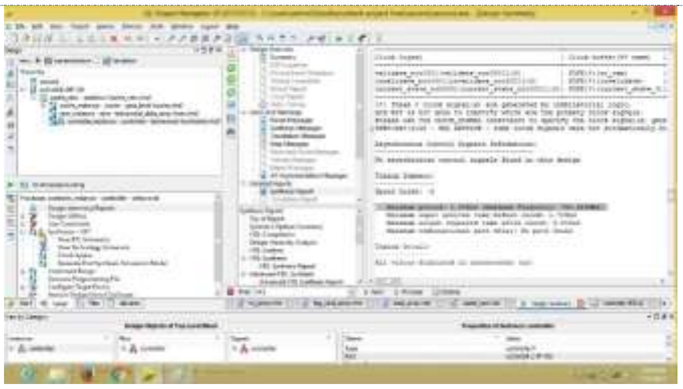

Figure 10: Frequency used by the proposed system.

The above figure shows the frequencies used by the technique. As observed the minimum period is 1.309ns whereas the maximum frequency is 763.82 MHZ. the minimum input arrival time before the clock is 1.709ns. The maximum output required time after the clock is 3.040ns and lastly maximum combinational path delay is not found in the system.

VI. CONCLUSION

In this paper we have proposed an LRU based cache, the system is better than the traditional system because the LRU saves the area. The optimization of the system is based on coding. The system designs RAM size of1KWord, with word size = 32 bits. The Cache Block Size is 64 Words and Number of cache blocks designed is 2. The system reduces the device utilization as low as 1% and finally, the delay is reduced as low as 1ns

VII. REFERENCES

[1] J. S. Yadav, M. Yadav, and A. Jain, "CACHE MEMORY OPTIMIZATION," International Conferences of Scientific Research and Education, vol. 1, no. 6, pp. 1–7, 2013.
[2] H. Dybdahl, "Architectural Techniques to Improve Cache Utilization" Diss. Ph.D. thesis, Norwegian University of Science and Technology, 2007.
[3] X. Ding and K. Wang, "ULCC : A User-Level Facility for Optimizing Shared Cache Performance on Multicores." Acm big plan notices, Vol. 46, No. 8, ACM, 2011.
[4] S. Paper, R. Sawant, B. H. Ramaprasad, S. Govindwar, and N. Mothe, "Memory Hierarchies-Basic Design and Optimization Techniques Survey on Memory Hierarchies – Basic Design and Cache Optimization Techniques," 2010.
[5] M. Zhang and K. Asanovic, "Highly-Associative Caches for Low-Power Processors", Appears in Kool Chips Workshop, 33rd Int.Symp. on Microarchitecture, 2000.
[6] J. Montanaro et al., "A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor", IEEE JSSC, Vol. 31(11):1703–1714, 1996.
[7] G. Albera and I. Bahar, "Power and performance tradeoffs using various cache configurations", In Power Driven Microarchitecture Workshop at ISCA98, 1998.
[8] N. Bellas, I. Hajj, and C. Polychronopoulos, "Using dynamic cache management techniques to reduce energy in a high-performance processor", In ISLPED, 1999.
[9] K. Inoue et al., "Way-predicting set-associative cache for high performance and low energy consumption", In ISLPED, 1999.
[10] U. Ko, P. Balsara, and A. Nanda, "Energy optimization of multilevel cache architectures for RISC and CISC processors", IEEE Trans. VLSI Systems, Vol. 6(2), 1998.
[11] P. Hicks, M. Walnock, and R.M. Owens, "Analysis of Power Consumption in Memory Hierarchies", Proc. Int'l Symp. Low Power Electronics and Design, pp. 239, 1997.
[12] R. Bergamaschi, et al., "Exploring power management in multi-core systems", In Proc. ASPDAC, 2008.
[13] R.E. Ahmed, "Energy-Aware Cache Coherence Protocol for ChipMultiprocessors", Proc. CCECE'06, 2006.
[14] C.H. Kim, et al., "First-Level Instruction Cache Design for Reducing Dynamic Energy Consumption", In SAMOS-2005, 2005.

[15] Hongliang Gao, Chris Wilkerson, "A Dueling Segmented LRU Replacement Algorithm with Adaptive Bypassing", Intel Corporation.
[16] Mukesh Kumari, Ashish Kumar, Neha, "SIMULATION OF LRU PAGE REPLACEMENT ALGORITHM FOR IMPROVING PERFORMANCE OF SYSTEM", Int.J.Computer Technology & Applications, Vol 4 (4),683-685, 2013.
[17] Predrag R. Jelenkovic´ and Ana Radovanovi, "Optimizing LRU Caching for Variable Document Sizes", Department of Electrical Engineering Columbia University, 2003.
[18] Priyanka Yadav, Vishal Sharma, Priti Yadav, "Cache Memory – Various Algorithm", International Journal of Computer Science and Mobile Computing, Vol.3 Issue.9, September- 2014, pg. 838-840

CITE AN ARTICLE

Lalwani, K., & Chawla, M. (2017). HIGH THROUGHPUT CACHE CONTROLLER USING VHDL & IT'S FPGA IMPLEMENTATION. *INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY, 6*(10), 346-353.