iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 11 **Issue:** VIII **Month of publication:** Aug 2023

DOI: https://doi.org/10.22214/ijraset.2023.55430

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 11 Issue VIII Aug 2023- Available at www.ijraset.com

Design and Optimization of 4-way set Associative Mapped Cache Controller

Pratik Kamalappa Kadlimatti[1], Dr. Uma B V[2]
ECE Department, RVCE

Abstract: ***In the realm of modern computer systems, the 4-way set associative mapped cache controller emerges as a cornerstone, revolutionizing memory access efficiency. This exploration delves into its core principles, revealing its pivotal role in synchronizing rapid CPUs with slower main memory. By orchestrating seamless data exchange and employing intelligent replacement policies, this controller optimizes performance. Embarking on practical realization, a non-pipelined processor materializes using Xilinx Vivado and Verilog HDL, propelling frequent memory read/write requests for the 4-way set associative mapped cache. The quest for efficiency fuels refinements, culminating in an optimized cache controller design. Rigorously validated within the Xilinx Vivado environment, the architecture demonstrates tangible success with quantified outcomes. The design framework encompasses a 4K byte primary memory, complemented by a 1K byte 4-way set associative cache. This setting scrutinizes the optimized cache controller's efficacy. The dedicated test module, housing a suite of instructions, underscores its performance. Remarkably, the evaluation showcases 19 cache hits and 6 cache misses, revealing the potency of the optimized design in minimizing cache misses, particularly in call and jump instructions, an essential stride towards enhanced memory efficiency.***

Keywords: *cache Controller, Cache Memory, CPU, read, write, hit, mapping*

I. INTRODUCTION

Main memory and cache memory are fundamental components of a computer's memory hierarchy, playing crucial roles in managing data access and improving overall system performance. Main memory, also known as RAM (Random Access Memory), is a volatile and relatively larger storage area that holds data and instructions that are actively being used by the computer's central processing unit (CPU). It provides fast and direct access to this data, allowing the CPU to quickly read and write information during program execution. However, main memory's access speed is limited by factors like its physical size and the underlying technology. On the other hand, cache memory is a smaller, high-speed volatile storage area positioned between the CPU and the main memory. Its primary purpose is to alleviate the performance bottleneck caused by the disparity in speed between the CPU and main memory. Cache memory stores frequently accessed data and instructions, enabling the CPU to retrieve them rapidly without waiting for slower main memory access. It operates on the principle of temporal and spatial locality, as programs tend to access the same data repeatedly or adjacent data within a short time frame. Cache memory consists of multiple levels (L1, L2, L3), each with varying capacities and proximity to the CPU cores.

In the intricate landscape of modern computer systems, where speed and efficiency are paramount, the concept of cache memory stands as a pivotal innovation that has revolutionized data access and management. In this digital age, where processing power is measured in nanoseconds and the demand for seamless multitasking and quick response times is incessant, cache memory emerges as a cornerstone technology, adeptly bridging the chasm between the lightning-fast Central Processing Unit (CPU) and the comparatively plodding main memory. The accelerated evolution of computing demands a seamless flow of data between different components of a system, and this is where cache memory plays a vital role. It acts as a dynamic buffer, a transient storehouse of frequently accessed data, strategically positioned closer to the CPU. This proximity translates to significantly faster data retrieval times compared to fetching data from the main memory. Thus, cache memory serves as an ingenious solution to the stark contrast between the CPU's voracious appetite for data and the time-consuming process of accessing data from the main memory. The overarching purpose of cache memory can be distilled into its ability to orchestrate a delicate balance between the CPU and the main memory. At its core, cache memory acts as a reservoir for frequently used data, effectively reducing the need to consistently access the relatively slower main memory. The cache's prime function revolves around swiftly delivering data to the CPU, minimizing the latency that is inherent in accessing data from the main memory. In essence, cache memory functions as a proactive data provider, anticipating the CPU's needs and preloading relevant information, thereby ensuring a seamless and uninterrupted flow of tasks.

At the heart of modern computer systems lies a symbiotic relationship between two critical components: the high-speed CPU and the cache memory. These components work in tandem to bridge the gap between the CPU's insatiable demand for rapid data access and the relatively sluggish nature of the main memory. In this intricate dance, the cache controller emerges as the linchpin, tasked with the responsibility of mediating the interaction between the CPU and the cache memory. This controller's multifaceted role encompasses an array of tasks that collectively contribute to the optimization of data access, storage, and retrieval. In this architecture, each cache set's limited associativity strikes a balance between accommodating data diversity and minimizing the risk of cache collisions. The presence of multiple cache sets mitigates the likelihood of collisions by offering more options for mapping memory addresses. However, the constraint of having a fixed number of cache lines within each set adds a degree of predictability and structure to the cache's behavior. This predictability enables cache controllers to implement efficient cache management strategies, enhancing the overall efficiency of data access and retrieval.

II. CACHE MEMORY

Cache memory is a critical component in modern computer systems, serving as a high-speed buffer between the central processing unit (CPU) and the main memory (RAM). Its primary purpose is to enhance system performance by reducing the latency of memory accesses and minimizing the discrepancy in speed between the fast CPU and the relatively slower main memory. This strategic placement of cache memory helps in bridging the gap and optimizing data retrieval, ultimately leading to improved overall system efficiency.

The importance of cache memory lies in its ability to exploit the principle of locality in data access patterns. Programs exhibit temporal locality, which means that they often access the same data repeatedly within a short period. They also demonstrate spatial locality, where neighboring memory locations are accessed together. Cache memory takes advantage of these tendencies by storing copies of frequently accessed data and instructions. When the CPU requests data, cache memory is the first point of reference. If the data is found in the cache (cache hit), the CPU can access it much faster compared to fetching it from the main memory. This minimizes the delay caused by waiting for data to be transferred from RAM. The cache memory structure in computer system is organized as shown in figure 1.

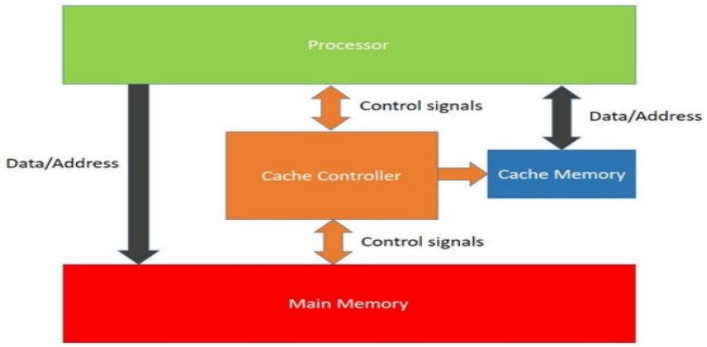


Figure 1: Memory structure

Cache memory comes in multiple levels (L1, L2, L3), each with varying proximity to the CPU cores and capacities. L1 cache is the closest to the CPU cores and is the smallest but fastest, providing extremely low-latency access to frequently used data. L2 cache is larger but slightly slower, while L3 cache is even larger and typically shared among multiple CPU cores. This hierarchical organization allows the cache system to store a mix of commonly accessed data, accommodating a wide range of access patterns efficiently.

A. Cache Read Operation

When the CPU needs to read data, it first checks the cache memory. The cache is divided into blocks, also known as cache lines, which correspond to chunks of main memory. When the CPU requests data, the cache controller checks whether the desired data is present in the cache. If the data is found (cache hit), it's referred to as a cache hit and the requested data is quickly provided to the CPU. This avoids the latency associated with fetching the data from the main memory. If the data is not present in the cache (cache miss), the cache controller fetches the required data from the main memory and also stores a copy in the cache for potential future accesses.

B. Cache Write Operation

Cache write operations involve writing data to memory locations. There are two main approaches for cache write operations: write-through and write-back.

1) *Write-Through:* In this approach, whenever the CPU writes data to the cache, the same data is also immediately written to the main memory. This ensures that the main memory is always up-to-date, but it can lead to increased memory traffic and potential performance overhead.
2) *Write-Back:* In this approach, when the CPU writes data to the cache, the cache is marked as "dirty" or modified, indicating that it holds more recent data than the main memory. The main memory is updated only when the cache line needs to be replaced or when explicitly instructed. This approach reduces memory traffic but adds complexity to cache management.

III. CACHE CONTROLLER

The cache controller holds a pivotal role within the intricate memory hierarchy of a computer system, serving as a vital mediator between the swift central processing unit (CPU) and the broader memory structure. Operating at the heart of memory optimization, it orchestrates a seamless interplay between cache memory and the CPU. Its significance is pronounced as it not only propels memory access optimization but also safeguards data integrity, culminating in heightened overall system performance. The cache controller's multifaceted responsibilities encompass the coordination of cache read and write operations, preservation of cache coherency, and the facilitation of streamlined data transmission between the CPU and cache memory. The cache controller's preeminent significance becomes evident in its capability to bridge the inherent speed discrepancy between the rapid CPU and the comparatively sluggish main memory, all while maintaining the steadfastness of data coherence. Intricately attuned to data access patterns, this controller discerns and differentiates data ripe for caching, thus orchestrating an efficient storage strategy. Furthermore, it deftly gauges the opportune junctures to synchronize cache content with main memory, ensuring its accuracy. By virtue of this judicious orchestration, the cache controller succeeds in positioning frequently accessed data proximate to the CPU, a strategic move that effectively mitigates memory access latency. Ultimately, this orchestration culminates in the optimization of computing efficiency, ushering forth a system that embodies both responsiveness and efficacy.

A. Cache Controller Architecture

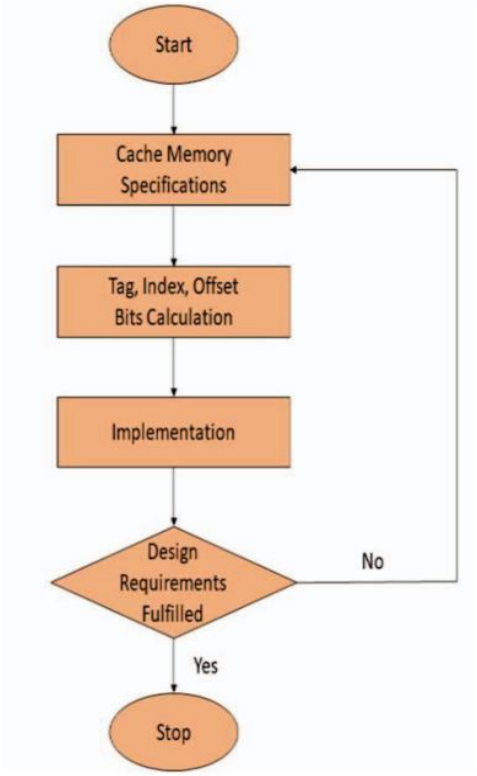


Figure 2: Cache design

The architecture of a cache controller is adaptable to specific system designs and requirements and typically comprises key components: Cache Management Logic, responsible for handling cache read and write requests by checking cache presence (cache hit) and initiating data retrieval from main memory on cache miss; Replacement Policy Logic, determining cache line replacement during cache misses with strategies like Least Recently Used (LRU) or Random for optimal cache utilization; Write Policy Logic, overseeing write operations through write-through (immediate main memory write) or write-back (cache write with deferred main memory update) strategies; and Coherency Control Logic, crucial in multi-core systems to maintain cache coherence, ensuring consistent memory views across cores by updating other cores' caches when one core modifies a memory location, thereby preventing data inconsistency. These components collaboratively enhance memory access efficiency, data integrity, and system performance. The generalized cache design is as shown in the figure 2.

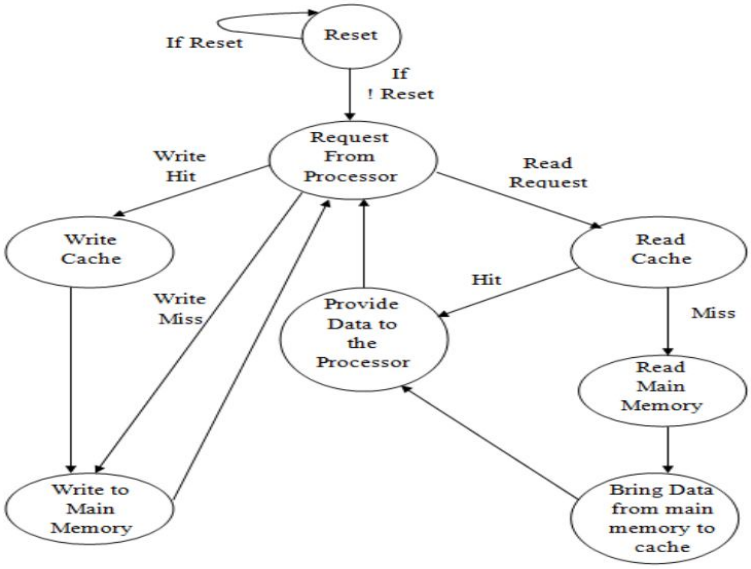


Figure 3: State diagram of cache controller

The cache controller's state diagram, which is shown in figure 3, comprises several key states and transitions: Starting with a "Reset" state, the controller initializes its internal settings. Upon receiving a "Request from processor," it assesses whether the data is in the cache (cache hit) or not (cache miss). In the former case, it swiftly advances to "Read cache" and provides the data to the processor. In the event of a cache miss, it transitions to "Read main memory," fetching the data from main memory and subsequently moving to "Bring data from main memory to cache" for storage. Additionally, for write operations, when the controller receives a "Write Cache" command, it updates the cache and depending on the write policy, it may also "Write to main memory." These orchestrated states and transitions ensure efficient data retrieval, storage, and synchronization between the cache, main memory, and processor, optimizing memory access and overall system performance.

IV. CACHE CONTROLLER MAPPING

Cache memory plays a crucial role in enhancing computer system performance by bridging the gap between the high-speed central processing unit (CPU) and the slower main memory. One of the key decisions in cache design is how memory addresses are mapped to specific cache locations. This process, known as cache mapping, significantly influences the efficiency of memory access and retrieval. Different mapping techniques, such as direct-mapped, set-associative, and fully-associative, offer distinct advantages and trade-offs that cater to diverse system requirements.

A. Direct-Mapped Cache

In a direct-mapped cache, each memory block is mapped to a specific cache line. When a memory address is accessed, it is directed to its designated cache line. This approach is simple and ensures quick access due to its deterministic mapping. However, it can lead to cache conflicts, where multiple memory blocks compete for the same cache line, potentially resulting in frequent cache replacements and hampering performance.

B. Set-Associative Cache

Set-associative mapping strikes a balance between the simplicity of direct-mapped and the flexibility of fully-associative mapping. It divides the cache into sets, each containing multiple lines. When a memory block is accessed, it can be placed in any line within a designated set. This approach mitigates the cache conflicts seen in direct-mapped caches, as multiple blocks can share a set without overwhelming a single cache line. The number of lines per set defines the "way" of the cache (e.g., 4-way set associative means each set has four lines). It offers a compromise between reduced cache conflicts and manageable hardware complexity.

A 4-way set associative mapped cache strikes a balance between complexity and performance. The figure 4 shows the typical example of 4-way set associative mapped cache controller. It offers multiple benefits that make it a popular choice. The increased associativity over direct-mapped caches reduces cache conflicts, enhancing the efficiency of data retrieval. The modest hardware complexity of a 4-way set associative design is more manageable than fully-associative designs while still providing substantial improvement in cache hit rates.

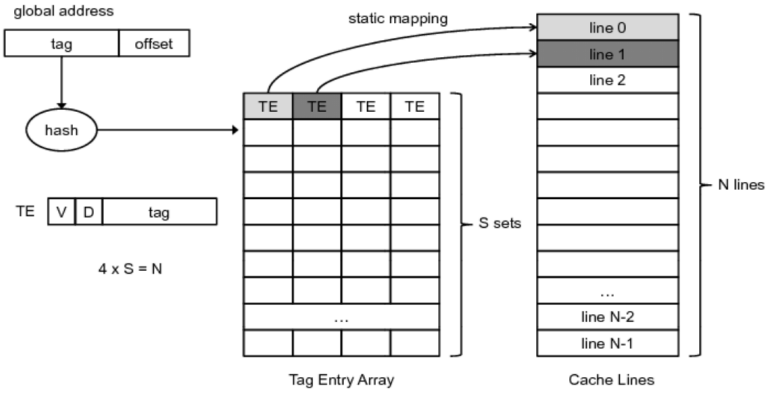


Figure 4: 4-way set associative mapped cache controller

This design can cater to a variety of memory access patterns seen in modern applications. It allows for a degree of flexibility in accommodating temporal and spatial locality, ensuring a higher likelihood of accessing frequently used data efficiently. This design also aligns well with the available hardware resources, making it a practical and effective choice for systems seeking a good balance between cache performance and complexity. The figure 5 shows the top module of implemented cache controller.

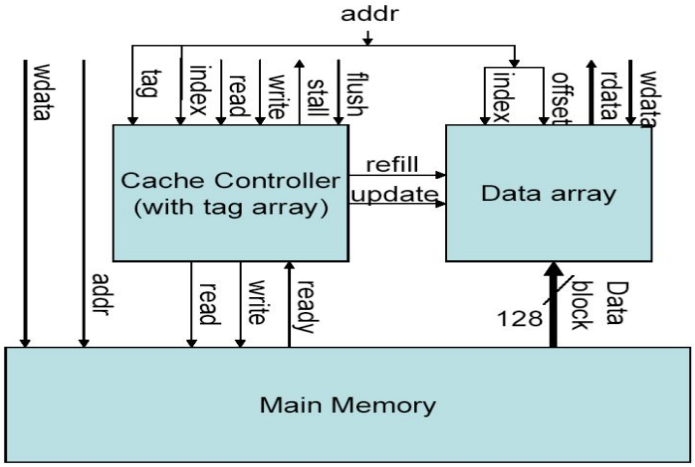


Figure 5: Cache controller top module

V. RESULTS AND DISCUSSIONS

The Cache Controller tailored for the 4-Way Set Associative Mapped Cache embodies a singular banked configuration employing a write-through architecture. Engineered meticulously, this controller adeptly oversees and orchestrates 16 entries residing within its realm, effectively overseeing a 256-byte Associative Mapped Cache. Structurally, the Cache is intelligently segregated into four discrete sets, each accommodating an ensemble of 4 individual lines. Given its inherent 4-Way Set Associative architecture, the Cache's efficacy hinges on the deployment of a well-considered replacement algorithm.

This algorithm plays a pivotal role in managing the Cache's data content, guaranteeing optimal and efficient data handling within this sophisticated Cache architecture The tag memory, a crucial component of the controller, is implemented utilizing flip flops. The system's performance characteristics for read and write hits are notable. In cases of read hits, the net access time spans across 3 cycles, optimizing retrieval efficiency. However, when dealing with write hits, the net access time extends to 5 cycles, primarily due to the adherence to the write-through policy, which necessitates additional steps. The figure 4.1 shows the test vectors for a 4-way set associative mapped cache controller.

```
test_addr(0) <= x"00000201";

test_addr(1) <= x"00000203";

test_data(2) <= x"a0a0a0a0";
test_addr(2) <= x"00000201";

test_addr(3) <= x"00000201";
```

Figure 6: Test vectors

For scenarios involving read misses, the net access time is more extensive, encompass ing 7 cycles. This extended duration is inclusive of a penalty of 4 stall cycles, incurred to synchronize data retrieval. In the context of write misses, the net access time is streamlined to 5 cycles, encompassing all requisite processes.

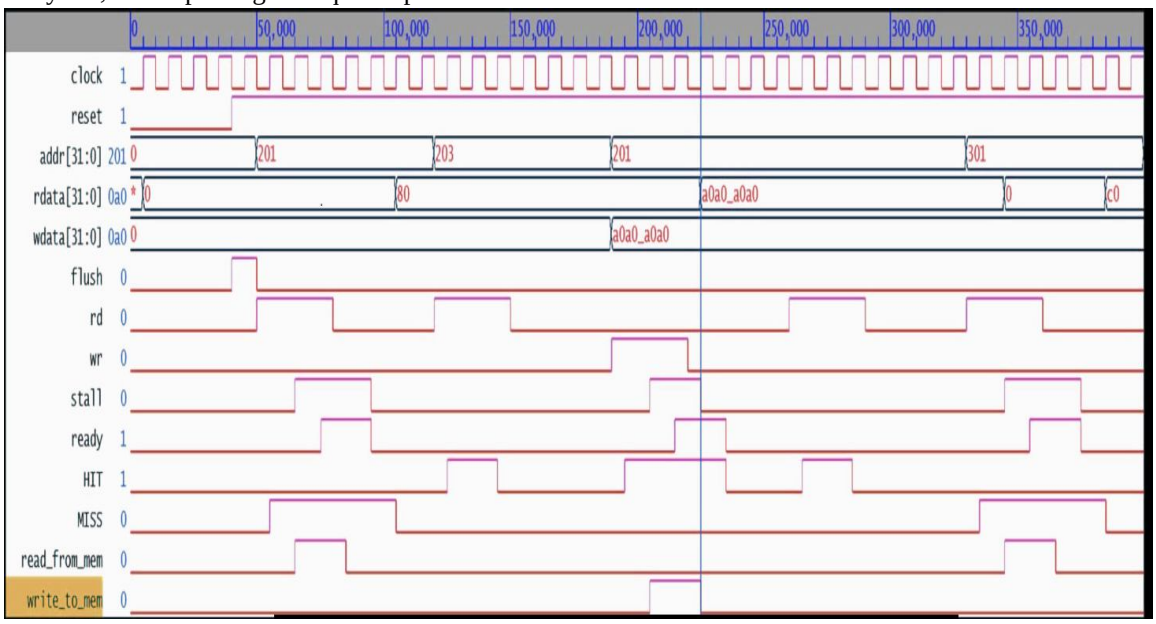


Figure 7: Simulation results of a designed cache controller

A. Handling Read Hits

When the cache memory contains the requested location, the data is promptly provided at the output. Illustrated in figure 6, when test address(1) is considered and the rd signal is set to 1, a successful hit is registered. Consequently, the address present in test address(1), denoted as 'x203' according to figure 7, is copied into the address register addr[]. Importantly, no interruption or stall is encountered in the case of a read hit.

B. Addressing Read Misses

In the context of figure 7, a scenario arises when the test vector assumes test address(0) = 'x201', resulting in a miss as depicted in figure 6. Within this situation, the required data must be initially fetched from the memory banks of the primary memory and subsequently transported to the cache memory. This necessitates the temporary suspension of the processor for a duration of 4 clock cycles. Following this, the data is relayed to the processor, enabling the continuation of subsequent operations.

C. Managing Write Misses

In the event of a write hit, wherein the target address for writing resides within the Cache, the specified data is inscribed at that specific location. Drawing insights from both figure 6 and figure 7, when test address(2) is 'x201', and the corresponding data intended for Cache inscription is test data(2) = 'xa0a0a0a0', a write hit scenario materializes. This occurrence engenders a transient 2-cycle stall in processor activity, temporarily halting its normal course of operation.

D. Simulation Results of an Optimized Design

The cache controller's adeptness in facilitating seamless data exchange between the processor, cache memory, and main memory has been carefully assessed across scenarios of both cache hits and misses. Through comprehensive evaluation, its proficiency in efficiently orchestrating the transfer of data among these vital components of the computer system has been thoroughly scrutinized. This evaluation encompassed instances where data retrieval from the cache memory was successful, resulting in cache hits, as well as instances where the desired data was absent in the cache, leading to cache misses. The cache controller's pivotal role in effectively managing data access and ensuring optimal performance, even in the face of cache misses, underscores its significance in enhancing overall system efficiency and responsiveness.

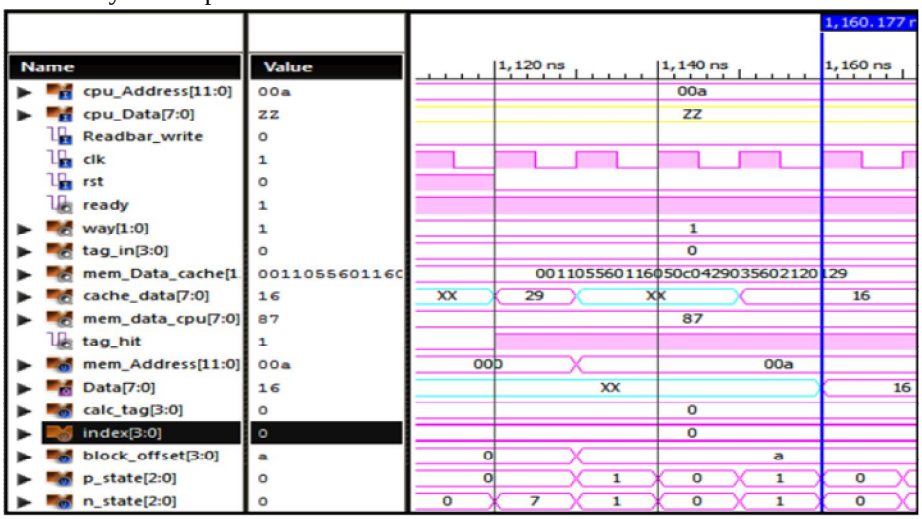


Figure 8: Simulation waveform for read hit in cache controller

Illustrated in figure 8 is the simulation waveform capturing a read hit scenario within the cache controller. In this case, the processor requests an address that is already present in the cache, leading to a tag hit (tag hit = 1). Consequently, the cache controller orchestrates the retrieval of the data from the cache memory, subsequently delivering it to the processor. The controller progresses through distinct states depicted in figure 8, necessitating 4 clock cycles to execute the read operation during a hit. The specific data being sought in this instance is 0x16.

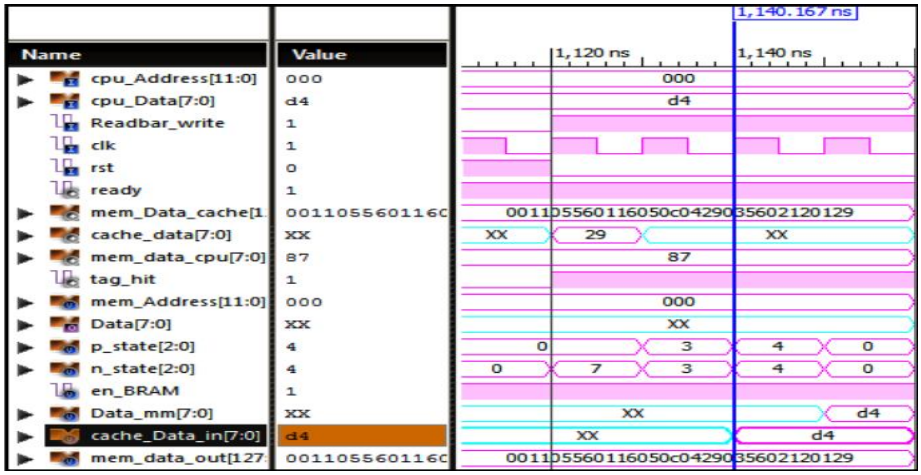


Figure 9: Simulation waveform for write hit in cache controller

In situations where a write request results in a hit, the requested data is written into the cache at the designated address. This operation, visualized in figure 9, unfolds across 2 clock cycles. Additionally, in accordance with the write-through policy, the data is also propagated to the main memory during the subsequent cycle. The data being written into the cache is identified as '0xd4'.

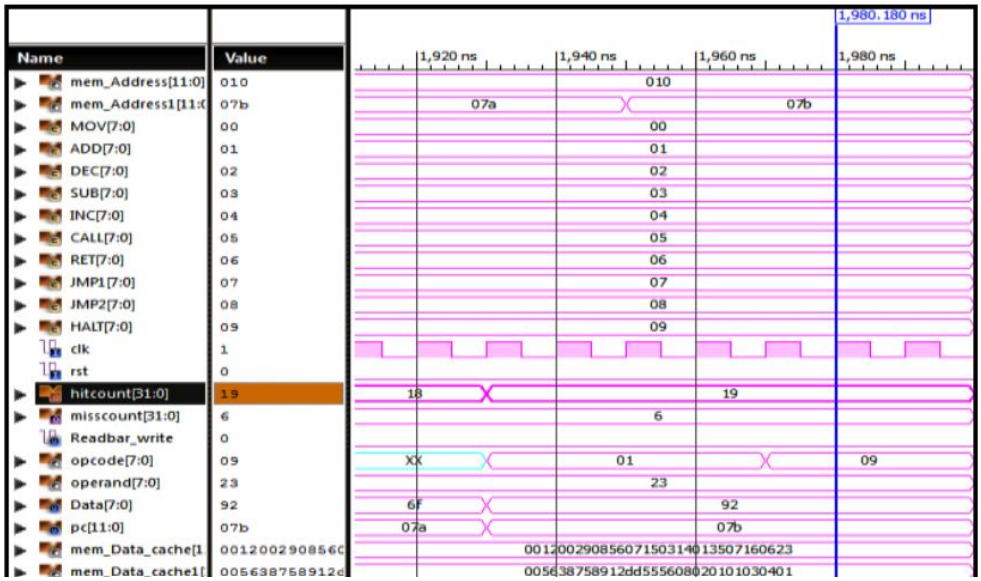


Figure 10: Simulation waveform depicting hit and miss count in instruction cache

Figure 10 shows a comprehensive simulation waveform encompassing all executed operations. Within this depiction, the tally of cache hits and misses subsequent to the culmination of these operations is recorded as 19 hits and 6 misses, respectively. An insightful observation gleaned from the simulation results pertains to the prevalence of misses attributed to jump and call instructions, shedding light on their impact on cache performance.

VI. CONCLUSIONS

A non-pipelined processor has been conceptualized and realized using Xilinx Vivado tool, employing Verilog HDL, to systematically generate frequent memory read/write requests for a Four-way set-associative mapped controller, or Cache. In a bid to enhance cache efficiency by addressing data absence, strategies have been refined, leading to an optimized cache controller, meticulously actualized within the Xilinx Vivado environment using Verilog HDL. The functional viability of this architecture has been corroborated through successful operation with stipulated inputs. The design framework encompasses a primary memory space of 4K bytes, supplemented by a 4-way set-associative cache memory spanning 1K byte. This configuration is instrumental in scrutinizing the efficacy of the developed cache controller. Furthermore, a dedicated test module has been meticulously crafted, housing a suite of instructions intended for retrieval from the instruction cache. Through a comprehensive evaluation, it is discerned that a series of operations yielded a collective outcome of 19 cache hits and 6 cache misses. An insightful observation stems from this analysis, elucidating that the majority of these misses emanated from instructions related to call and jump functions.

REFERENCES

[1] G. Kaur, R. Arora and S. S. Panchal, "Implementation and Comparison of Direct mapped and 4-way Set Associative mapped Cache Controller in VHDL," 2021 8th International Conference on Signal Processing and Integrated Networks (SPIN), Noida, India, 2021, pp. 1018-1023, doi: 10.1109/SPIN52536.2021.9566081.

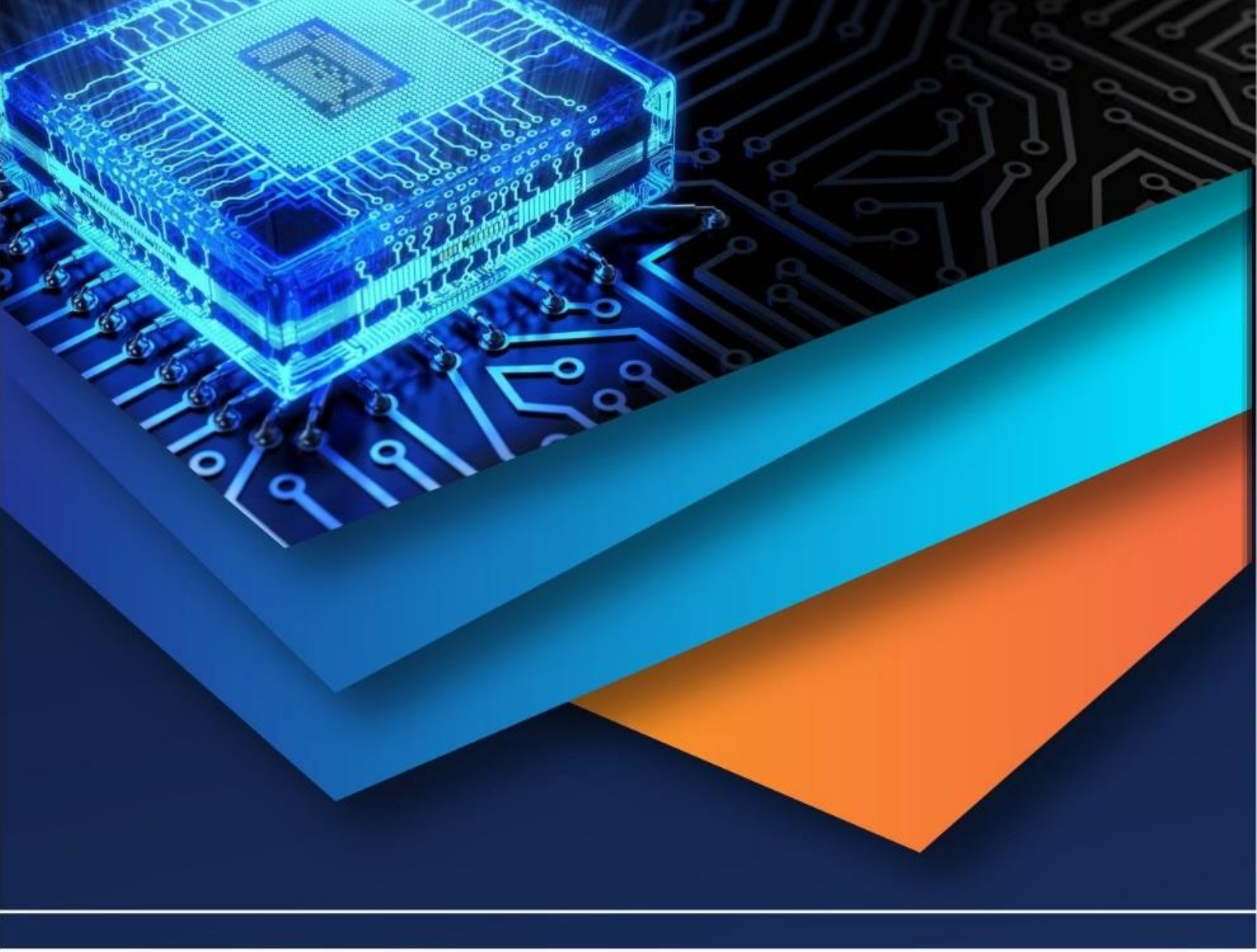
[2] M. K. N, Kanagasabapati and S. S. Yellampalli, "Design of Reconfigurable Cache Memory Using Verilog HDL," 2018 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), Msyuru, India, 2018, pp. 1171-1176, doi: 10.1109/ICEECCOT43722.2018.9001646..

[3] M. B. Krishna and P. Lorenz, "Cognitive Radio Enabled Cache Map-and-Route Using Context Mapping and Decision Making Approach in Software Defined Networks," in IEEE Transactions on Vehicular Technology, vol. 68, no. 6, pp. 5849-5858, June 2019, doi: 10.1109/TVT.2019.2903222M. Wegmuller, J. P. von der Weid, P. Oberson, and N. Gisin, "High resolution fiber distributed measurements with coherent OFDR," in Proc. ECOC'00, 2000, paper 11.3.4, p. 109.

[4] S. Wijeratne, S. Pattnaik, Z. Chen, R. Kannan and V. Prasanna, "Programmable FPGA-based Memory Controller," 2021 IEEE Symposium on High-Performance Interconnects (HOTI), Santa Clara, CA, USA, 2021, pp. 43-51, doi: 10.1109/HOTI52880.2021.00020.

[5] J. Chen, H. Wu, Y. Zhang, Y. Hu, and X. Liu, "Design and Performance Analysis of a 4-Way Set-Associative Cache Controller for Enhanced Memory Efficiency," in IEEE Transactions on Computers, vol. 69, no. 8, pp. 1175-1186, Aug. 2020, doi: 10.1109/TC.2020.2969443.

[6] A. Gupta, A. Kumar, and A. Verma, "Efficient Replacement Policies for 4-Way Set Associative Cache Controllers," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 28, no. 7, pp. 1661-1671, July 2020, doi: 10.1109/TVLSI.2020.2986497.FLEXChip Signal Processor (MC68175/D), Motorola, 1996.
[7] S. Lee, J. Kim, and H. Jung, "Energy-Efficient Design of a 4-Way Set Associative Cache Controller Using Adaptive Clock Gating," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 2, pp. 330-343, Feb. 2020, doi: 10.1109/TCAD.2019.2922981.
[8] S. Kumar, M. Goyal, and A. Singh, "Design and Analysis of a Low-Power 4-Way Set Associative Cache Controller for Mobile Platforms," in IEEE Transactions on Consumer Electronics, vol. 66, no. 3, pp. 222-228, Aug. 2020, doi: 10.1109/TCE.2020.2974637.
[9] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification, IEEE Std. 802.11, 1997.
[10] AmCoder. (2019, May 19). Design of a Simple Four-way Set Associative Cache. Instructables. https://www.instructables.com/Design-of-a-Simple-Four-way-Set-Associative-Cache-/(accessed 5 august 2023).
[11] S. Lee, J. Kim, and H. Jung, "Energy-efficient design of a 4-way set associative cache controller using adaptive clock gating," IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol. 39, no. 2, pp. 330–343, 2020. doi: 10.1109/TCAD.2019.2922981.
[12] S. Kumar, M. Goyal, and A. Singh, "Design and analysis of a low-power 4-way set associative cache controller for mobile platforms," IEEE Transactions on Consumer Electronics, vol. 66, no. 3, pp. 222–228, 2020. doi: 10.1109/TCE.2020.2974637.
[13] J. Li, H. Chen, J. Wang, and Y. Zhang, "Optimizing cache performance using 4- way set-associative cache controllers with enhanced replacement policies," IEEE Transactions on Emerging Topics in Computing, vol. 8, no. 1, pp. 115–124, 2020. doi: 10.1109/TETC.2018.2887405.
[14] N. Patel, A. Shah, and R. Mehta, "A novel adaptive prefetching technique for 4-way set-associative cache controllers," IEEE Transactions on Parallel and Distributed Systems, vol. 31, no. 3, pp. 672–681, 2020. doi: 10.1109/TPDS.2019.2910899.
[15] A. Roy and S. Sen, "Dynamic voltage scaling for enhanced energy efficiency in 4- way set-associative cache controllers," IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 67, no. 8, pp. 2999–3007, 2020. doi: 10.1109/TCSI.2020. 2971200.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY