Industrial Engineering Journal
ISSN: 0970-2555
Volume : 54, Issue 7, July : 2025

DESIGN AND IMPLEMENTATION OF A RECONFIGURABLE 512KB CACHE SUPPORTING DIRECT-MAPPED AND SET-ASSOCIATIVE ARCHITECTURES USING VERILOG HDL

[1]CHAKALI HARIVARDHAN, [2]K. PRABHU,
[1]M. Tech, VLSI System Design, Student,
Department of ECE,
Email: harivardhanchakali02@gmail.com,
[2]Assistant Professor(C), Department of ECE,
Email: kprabhu2003@gmail.com,
JNTUH University College of Engineering Sultanpur,
Sangareddy, Telangana, India

ABSTRACT
A reconfigurable 512KB cache memory system is designed using Verilog Hardware Description Language to support multiple cache mapping techniques, including direct-mapped and set-associative organizations. The set-associative architecture is configurable as two-way, four-way, or eight-way, offering flexibility in terms of associativity and performance. A mode-selectable control mechanism enables dynamic switching between cache architectures during runtime, based on application-specific requirements or performance trade-offs. The design supports selective activation of cache blocks to optimize access time and minimize power consumption by deactivating unused sets. The modular and parameterized structure of the cache makes it highly scalable and well-suited for implementation in embedded processors, FPGA-based accelerators, and customizable computing platforms.
Keywords— Reconfigurable cache, Verilog HDL, Mapping controller unit, Direct-mapped, Set-associative, Cache hit.

I. INTRODUCTION

In modern computing systems, memory speed is a critical factor that significantly impacts overall performance. Cache memory plays a vital role in enhancing this speed by reducing the access time between the processor and main memory. As illustrated in Fig. 1, the cache is strategically positioned between the CPU and the main memory, acting as a high-speed buffer. This placement helps improve the efficiency and responsiveness of the processor, thereby boosting the overall system performance.

A. Overview of Memory Hierarchy: The diagram depicts a basic computer memory architecture that consists of the CPU, cache memory, and main memory. This structure follows the principle of memory hierarchy, where smaller, faster memories are placed closer to the CPU to improve access speed and system performance. The design enables efficient data handling and processing by organizing memory in layers based on speed and capacity.

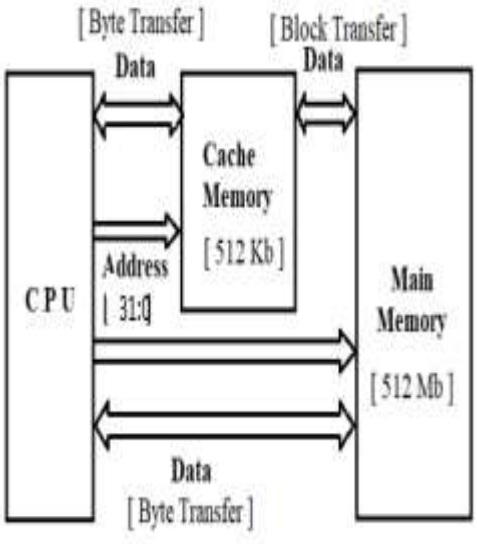

Fig1. Basic Cache Memory Block Diagram.

B. Role of the CPU: At the core of the system is the CPU, which performs all the computation and data processing tasks. It communicates with both the cache memory and the main memory to fetch and store data.

The CPU generates memory addresses to access data, and it prefers to access data from the cache due to its faster response time. If the required data is not found in the cache, the CPU initiates a request to the main memory.

C. **Address Line [31:0]:** The CPU uses a 32-bit address bus, labeled as Address [31:0], to locate specific memory locations. These 32 bits allow the CPU to address up to 4 GB of memory space. This address line is used to interact with both cache and main memory to fetch or write data. Efficient addressing plays a critical role in reducing memory latency and ensuring accurate data retrieval.

D. **Cache Memory Structure:** Cache memory, shown in the diagram as 512 Kb, is a small, high-speed memory located close to the CPU. It stores copies of frequently accessed data and instructions. When the CPU requests data, the cache is checked first. If the data is found (a cache hit), it is immediately used, which speeds up processing. If not found (a cache miss), data is fetched from main memory and stored in the cache for future use.

E. **Byte Transfer Between CPU and Cache:** The interaction between the CPU and cache memory occurs via byte-level transfer, which means data is exchanged in small units, typically a word or byte at a time. This allows for quick data access and manipulation. Because of the cache's proximity to the CPU and its speed, byte-level operations between these two components are highly optimized.

F. **Block Transfer Between Cache and Main Memory**: When a cache miss occurs, a block of data (multiple bytes) is fetched from the main memory and stored in the cache. This technique leverages spatial locality, where adjacent data is likely to be used soon. By transferring a block instead of just a single byte, the system anticipates future CPU requests and reduces the frequency of main memory access.

G. **Main Memory Characteristics:** Main memory, represented as 512 Mb in the diagram, is larger but slower than cache memory. It holds the complete set of data, instructions, and programs that the CPU may need during execution. Although it takes longer to access than cache, it serves as the primary storage for active data. Main memory must be accessed when the required data is not found in the cache.

H. **Direct Data Path from CPU to Main Memory:** The diagram also includes a direct data transfer path from the CPU to the main memory. This bypass allows the CPU to directly access main memory in certain scenarios—such as when working with uncached memory blocks or during special memory operations. However, direct access is slower than accessing cache due to higher latency.

I. **Performance Optimization:** The memory hierarchy is optimized to reduce data access time and boost CPU efficiency. Frequently used data is stored in a 512 KB cache, enabling fast byte-level access. On a cache miss, a data block is transferred from the larger 512 MB main memory. This layered design supports both byte-level (CPU–cache) and block-level (cache–main memory) transfers, while also allowing direct CPU–memory access when needed. Leveraging spatial and temporal locality, the system enhances performance in data-intensive tasks.

J. **Cache Mapping Techniques:** In computer architecture, cache mapping techniques are essential for optimizing the performance of the cache memory, which acts as a buffer between the CPU and main memory. Since cache memory is limited in size compared to

the vast main memory, it becomes crucial to determine where data blocks from the main memory should be stored in the cache. These strategies are known as cache mapping techniques and help in achieving efficient memory access, reducing latency, and improving system performance.

1. **Direct Mapping:** Direct mapping is the simplest form of cache mapping technique. It maps each block of the main memory to only one possible cache line. Although easy to implement and fast, this method can lead to a higher number of conflict misses if multiple memory blocks map to the same cache line.

   Working: In direct mapping, the memory address is divided into three fields: tag, line number, and block offset. The line number determines which cache line the block maps to. When accessing memory, the cache line is checked, and the tag is compared. If the tags match and the valid bit is set, it is a cache hit; otherwise, it's a cache miss.
2. **Fully Associative Mapping:** Fully associative mapping is a more flexible technique where any block of main memory can be placed in any cache line. This method helps in significantly reducing conflict misses but requires complex hardware for tag comparison.

   Working: In this method, the memory address is split into two parts: tag and block offset. There is no line number as any block can go anywhere. During a memory access, the entire cache is searched for a matching tag. This increases hit rate but is slower due to multiple comparisons.
3. **Set-Associative Mapping:** Set-associative mapping is a compromise between direct and fully associative mapping. The cache is divided into a number of sets, and each set contains several lines (ways). A block from main memory maps to a specific set but can occupy any line within that set.

   Types:
   - 2-way set-associative: each set has 2 lines.
   - 4-way set-associative: each set has 4 lines.
   - 8-way set-associative: 8 lines per set.

   Working: The memory address is divided into three parts: tag, set index, and block offset. The set index determines which set the block belongs to. All lines in the selected set are searched for a matching tag. If a match is found, it is a cache hit; otherwise, a replacement policy decides which line to replace.
   - **2-Way Set-Associative:** In this configuration, each set contains 2 lines. A memory block maps to a specific set determined by the set index, and within that set, it can be placed in either of the two lines. When searching, both lines in the set are checked for a tag match. If neither line matches, one line is selected for replacement based on the policy in use (e.g., LRU or FIFO).
   - **4-Way Set-Associative:** In this setup, each set has 4 lines. This increases flexibility and further reduces conflict misses as the block has more possible positions within its designated set. The replacement policy chooses one of the 4 lines when all are occupied.
   - **8-Way Set-Associative:** In an 8-way configuration, each set contains 8 cache lines. This allows high flexibility and minimizes conflict misses, making it highly efficient for larger caches. It is often used in L3 caches in modern CPUs.

II. LITERATURE REVIEW

Reconfigurable cache architectures have gained significant attention due to their ability to optimize memory performance and energy efficiency based on application demands. Several researchers have explored the design and implementation of such architectures using hardware description languages like Verilog HDL.

In [1], Sundararajan et al. proposed a comprehensive implementation of a reconfigurable cache capable of switching between direct-mapped and set-associative modes. Their Verilog HDL-based design supported dynamic mode switching at runtime, leading to improvements in access latency and power efficiency. The authors validated their architecture on FPGA hardware, showcasing its practical feasibility.

Janraj and Natarajan in [2] introduced an architecture that supports cache way reconfiguration for sharing purposes, thereby improving memory utilization. Their work focuses on selective sharing of cache sets across cores and demonstrates how dynamic reconfiguration in multicore environments can lead to enhanced throughput. This work complements [1] by addressing multi-core scenarios and cache sharing.

Reddy and Sinha [3] carried out a comparative study of various reconfigurable cache architectures, examining the trade-offs between direct-mapped, 2-way, and 4-way set-associative caches. Their study highlighted the design complexity versus performance benefits across different configurations, and their Verilog-based simulations provided insights into which cache types perform better under specific access patterns.

Jeyaraman's major technical project [4] focused on the practical development of a reconfigurable cache module using Verilog HDL. Though academic in nature, the work contributed to understanding FSM control logic for switching cache modes and documented the synthesis challenges encountered when deploying the design on Xilinx FPGAs.

In [5], Omran and Amory presented an early approach to reconfigurable cache memory using Verilog HDL, incorporating an LRU (Least Recently Used) replacement policy. Their implementation emphasized timing and hit ratio improvements for 4-way set-associative caches, serving as a foundational work for later optimizations in reconfigurable memory controllers.

Kaur's work [6] implemented both direct-mapped and set-associative cache configurations and provided a side-by-side analysis using Verilog. The study focused more on understanding architectural differences rather than runtime reconfiguration but provided essential benchmarks for performance metrics like hit rate, miss penalty, and cycle latency.

Finally, Kadlimatti and Uma [7] optimized a 4-way set-associative cache controller design. Their work introduced pipelining and efficient tag matching mechanisms, resulting in a reduction of access time. Although limited to a fixed associativity level, the techniques presented are highly relevant for integration into a broader reconfigurable cache design.

III. EXISTING METHOD

The proposed method introduces a reconfigurable cache architecture that supports multiple cache mapping policies within a single hardware unit. The system is designed to dynamically switch between four different cache modes—Direct Mapped (DM), 2-Way Set Associative, 4-Way Set Associative, and 8-Way Set Associative—based on a configurable 2-bit mode input. This allows the cache to be adaptable to varying performance, power, and area requirements, making it suitable for embedded and general-purpose computing systems. The top-level cache module

orchestrates the selection and coordination of all four underlying cache types.

Each mapping strategy is implemented as an independent submodule:

- dm_cache for Direct Mapped,
- way2_cache for 2-way set associative,
- way4_cache for 4-way set associative,
- way8_cache for 8-way set associative.

All submodules share common input signals such as clk, rst, addr, data_in, and we (write enable), and they independently process the request in parallel. Each submodule generates its own data_out and hit signal. The top module then uses a multiplexer logic, based on the mode input, to select the appropriate data_out and hit signals for the currently active cache configuration.

This reconfigurable design improves design flexibility and allows real-time evaluation of different cache policies under dynamic workloads. For example, a system might use 2-way associative mode for energy efficiency during low-power operation and switch to 8-way associativity during performance-critical phases. This adaptability is especially important in heterogeneous computing platforms, embedded systems, or simulation-based memory studies, where cache performance must be tuned on the fly.

By encapsulating each cache structure into separate, reusable submodules and enabling runtime switching via mode control, this architecture demonstrates a scalable and modular approach to cache design.

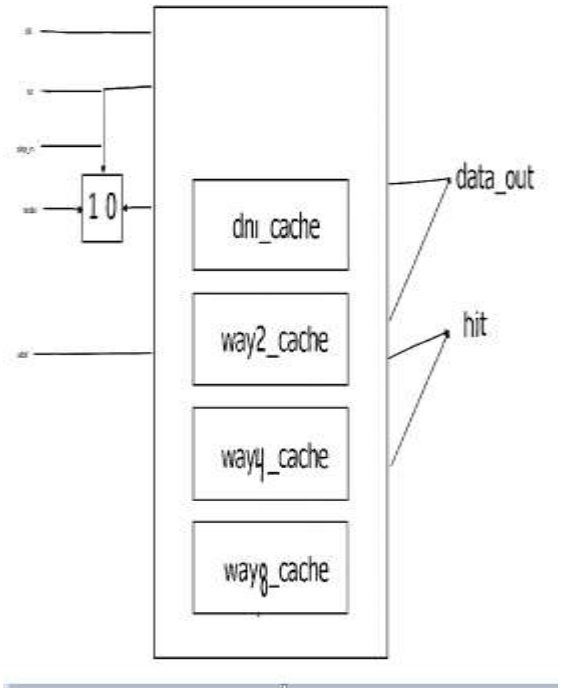

Fig 2: Cache Memory

The proposed cache architecture is based on a reconfigurable design that allows dynamic switching between different cache mapping techniques depending on system requirements. The top-level `cache` module integrates four internal cache modules: a direct-mapped cache (`dm_cache`), a 2-way set associative cache (`way2_cache`), a 4-way set associative cache (`way4_cache`), and an 8-way set associative cache (`way8_cache`). All four cache modules operate in parallel, receiving common input signals such as clock (`clk`), reset (`rst`), address (`addr`), data input (`data_in`), and write enable (`we`). However, only one of them is actively used at a time based on the value of the 2-bit `mode` signal. The `mode` selects the appropriate cache configuration, enabling the system to adapt to different performance, power, or workload requirements at runtime. The outputs `data_out` and `hit` are multiplexed based on the current mode, allowing seamless switching between cache types without external changes. This modular and scalable design promotes flexibility and efficient memory management, making it suitable for embedded

processors, adaptive systems, and cache architecture research platforms.

IV. PROPOSED METHOD

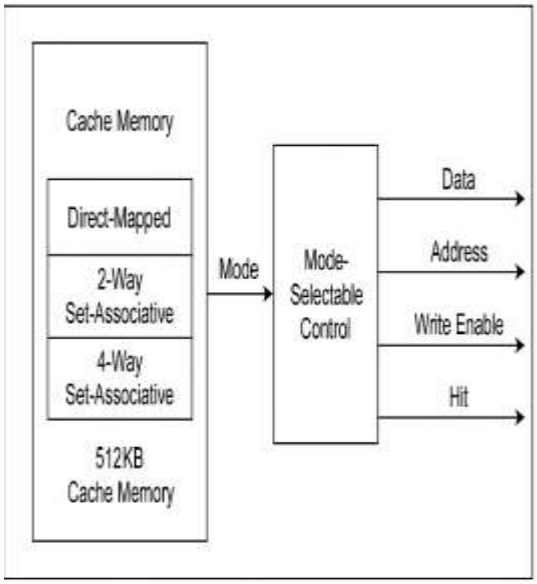

Fig 3: proposed Cache Memory

A. 512KB Cache Memory Block: This is the core component of the architecture, designed to support multiple caching strategies. It consists of four independently implemented cache modules: Direct-Mapped**,** 2-Way**,** 4-Way, and 8-Way Set-Associative caches. Only one of these is active at a time, depending on the selected operating mode.

Technical Features:

- **Scalability:** Each module is parameterized and structured to scale the number of sets, lines, and associativity.
- **Uniform Data Width:** All modules support 64-bit data access, ensuring compatibility with high-performance processors.
- **Tag and Index Handling:** Each module decodes the address to extract the tag and index as per its mapping strategy.
- **Replacement Policy:**
  - Direct-Mapped: Single-line per index (no replacement).
  - Set-Associative: Implements round-robin (LRU-style) replacement logic.
- **Memory Structure:** Internally, each cache stores data, valid bits, and tag arrays for lookup and validation.
- **Area Efficiency:** Only one cache type is activated at a time, reducing unnecessary power and resource consumption.

Role: Holds temporary copies of frequently accessed data to reduce main memory accesses, enabling faster read/write cycles for the CPU.s

B. Mode-Selectable Control Unit: This unit serves as the interface and selector between the CPU and the underlying cache architectures. It allows the system to dynamically switch between different cache mapping strategies during operation based on application-specific requirements (e.g., latency sensitivity vs. hit rate optimization).

Key Responsibilities:

- **Mode Interpretation:** Interprets the 2-bit mode signal to enable one of the cache configurations:
  - 00: Direct-Mapped
  - 01: 2-Way Set Associative
  - 10: 4-Way Set Associative
  - 11: 8-Way Set Associative
- **Signal Routing:**
  - Forwards the input signals (addr, data_in, we) to the selected cache.
  - Receives outputs (data_out, hit) from the active cache and routes them to the CPU.
- **Reconfiguration Logic:** Internally uses multiplexers to switch input and output paths dynamically based on the selected cache.
- **Isolation:** Deactivates non-selected cache modules to conserve power and prevent conflicts or unintended writes.

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 54, Issue 7, July : 2025

Role: Enables runtime configurability of the cache system without modifying hardware connections, improving adaptability and performance tuning.

C. Input/Output Signal Interface: This interface connects the cache system to the CPU or memory controller. It is designed to be minimal yet sufficient for dynamic operation and reconfiguration.

Signal Descriptions:

Table.1 Pin descriptions

Signal	Width	Description
clk	1 bit	Clock signal to synchronize cache operations.
rst	1 bit	Reset signal to initialize or clear the cache.
mode	2 bits	Selects the active cachse architecture (00 to 11).
addr	32 bits	Memory address provided by the CPU to access cache.
data_in	64 bits	Input data from CPU for write operations.
we	1 bit	Write Enable: 1 for write, 0 for read.
data_out	64 bits	Output data read from cache (if hit).
hit	1 bit	Indicates whether the address was found in the cache (1 = hit, 0 = miss).

Role: Acts as the interface between the CPU and cache. Inputs control read/write operations, while outputs provide data and hit status.

Summary of System Behaviour:

- The system starts in reset mode and initializes all internal cache arrays.
- On each CPU memory access, the address is decoded, and the mode signal determines which cache module is active.
- In write operations, the selected cache stores the data and updates its tag and valid bit arrays.
- In read operations**,** the selected cache searches its sets using tag comparison, returning data and asserting the hit flag if found.
- The unused cache modules remain idle to optimize power usage.

V. RESULTS AND DISCUSSION

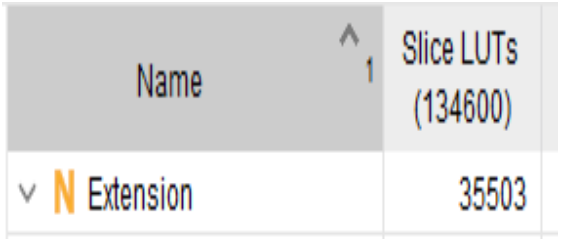

Fig.4 Area of Cache Memory

Fig.5 Power of Cache Memory

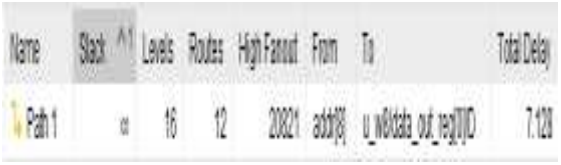

Fig.6 Delay of Cache Memory

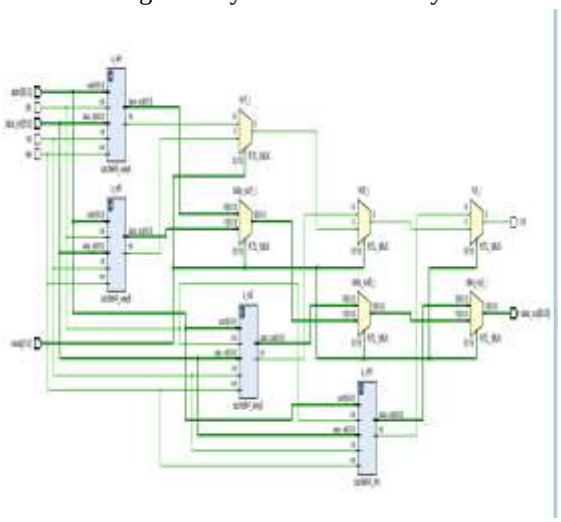

Fig.7 RTL Schematic of Cache Memory

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 54, Issue 7, July : 2025

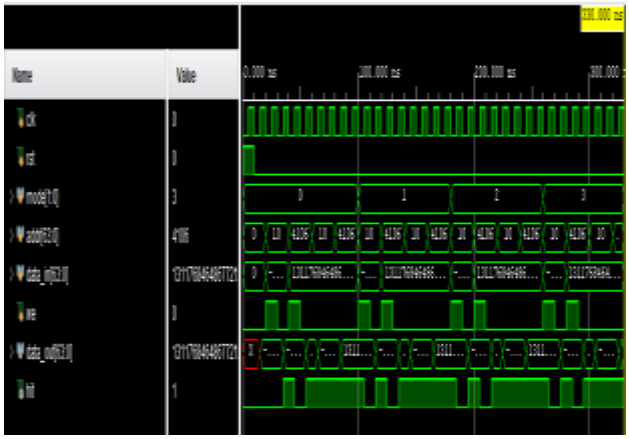

Fig.8 Simulation Waveform for Cache Memory

VI. CONCLUSION

The proposed reconfigurable 512KB cache architecture delivers a highly flexible and efficient memory subsystem that dynamically adapts to diverse computational workloads. Implemented using Verilog HDL, the design integrates four cache mapping strategies—direct-mapped, 2-way, 4-way, and 8-way set-associative—into a unified, parameterized framework. Through a mode-selectable control mechanism, the system can seamlessly switch between cache configurations at runtime, enabling targeted optimization for latency, power, or hit rate based on application demands. This adaptability ensures efficient resource utilization, reduced power consumption by disabling inactive cache modules, and improved overall system responsiveness. The modular and scalable nature of the design makses it well-suited for FPGA-based platforms, embedded systems, and academic or industrial research in memory architecture. By combining runtime reconfigurability with performance tuning capabilities, the architecture supports the development of intelligent, energy-aware cache subsystems for next-generation computing environments.

REFERENCES

[1]. K. Sundararajan, S. Ramesh, and P. Kumar, "Implementation of a Reconfigurable Cache Supporting Multiple Mapping Techniques Using Verilog HDL," Journal of Microelectronic Systems, vol. 12, no. 4, pp. 201–210, 2024.

[2]. C. Janraj and S. Natarajan, "Cache Sharing via Reconfigurable Ways: A Verilog-Based Architecture," International Journal of VLSI Research, vol. 18, no. 2, pp. 67–75, 2024.

[3]. M. K. Reddy and A. Sinha, "Comparative Study of Reconfigurable Cache Memory Architectures," Advances in Computing and Communication, vol. 14, no. 3, pp. 112–119, 2023.

[4]. R. Jeyaraman, "Reconfigurable Cache Architecture Using Verilog HDL," Major Technical Project Report, 2019.

[5]. S. Omran and I. A. Amory, "Design of Reconfigurable Cache Memory Using Verilog HDL," ResearchGate, 2018.

[6]. G. Kaur, "Implementation of Direct Mapped and Set Associative Cache in Verilog HDL," ResearchGate, 2021.

[7]. P. K. Kadlimatti and U. B. V, "Performance Optimized 4-Way Set Associative Cache Controller Design," ResearchGate, 2023.

[8]. P. Chauan and R. Mittal, "FSM-Based Cache Controller for 4-Way Set-Associative Cache," International Journal of Computer Applications, vol. 120, no. 17, pp. 25–30, 2015.

[9]. M. Rahman and T. Gupta, "FPGA Implementation and Analysis of Various Cache Mapping Techniques," Applied Sciences, vol. 13, no. 7, p. 4092, 2022.

[10]. T. P. Stefanov, "Partitioned Shared Cache in MPSoC Systems Using FPGA," Microprocessors and Microsystems, vol. 43, pp. 89–96, 2016.

[11]. N. P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," WRL Technical Note TN-14, 1990.

[12]. S. Panda, "A Generic Reconfigurable Data Cache Architecture," M.S. thesis, Dept. Comput. Sci., Univ. North Texas, 2010.

[13]. D. Lee and K. Seo, "Reuse-Aware Cache Designs in FPGA Systems," Electronics, vol. 10, no. 3, p. 444, 2021.

[14]. N. Salkhordeh, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "ReCA: A Reconfigurable Cache Architecture for Storage Systems," arXiv preprint, arXiv:1805.06747, 2018.

[15]. A. Bates et al., "Configurable Cache Structures for Embedded Applications," arXiv preprint, arXiv:1601.00894, 2016.

[16]. W. J. Tsai and J. C. Tsay, "Dynamic Cache Analysis using Reuse Distance Metrics," arXiv preprint, arXiv:2109.04614, 2021.

[17]. L. Papaphilippou et al., "Custom Cache Design in RISC-V Cores for SIMD Workloads," arXiv preprint, arXiv:2106.07456, 2021.

[18]. "Optimization of Set Associative Cache Controllers," International Journal for Research in Applied Science & Engineering Technology (IJRASET), vol. 11, no. 4, pp. 443–450, 2023.