Chapter

New Content Addressable Memory Architecture for Multi-Core Applications

Allam Abumwais and Mahmoud Obaid

Abstract

The future of massively parallel computation appears promising due to the emergence of multi- and many-core computers. However, major progress is still needed in terms of the shared memory multi- and many-core systems, specifically in the shared cache memory architecture and interconnection network. When multiple cores try to access the same shared module in the shared cache memory, issues arise. Cache replacement methods and developments in cache architecture have been explored as solutions to this. This chapter introduces the Near-Far Access Replacement Algorithm (NFRA), a new hardware-based replacement technique, as well as a novel dedicated pipeline cache memory design for multi-core processors, known as dual-port content addressable memory (DPCAM). The experiments show that the access latency for write/read operations of a DPCAM is lower than that of a set-associative (SA) cache memory, with the latency of a write operation staying the same regardless of the size of the DPCAM. It is estimated that the power usage will be 7% greater than a SA cache memory of the same size.

Keywords: multi-core processor, shared cache, cache architecture, dual port CAM, replacement algorithm

1. Introduction

The purpose of the special purpose shared memory architecture discussed in this chapter is to allow multiple cores of a multi-core processor to access a cache memory simultaneously, thus decreasing access latency compared to set-associative (SA) caches. This proposed architecture is based on CAM and a new replacement algorithm. In Section 1, the introduction of shared memory types in computer design is discussed, and Section 2 covers the architectures of the DPCAM and the Near-Far Access NFRA. Section 3 provides functional and timing simulation results, power estimation analysis, and an FPGA implementation of the DPCAM.

Multi-core ICs package multiple processors into a single device. Many-core systems, an evolution of multi-core technology, provide intense parallel processing capabilities for a large number of cores. In order for many-core systems to work, shared memory must be used to communicate between the cores. However, this shared memory can become problematic if multiple cores attempt to access it simultaneously. To address this issue, there have been various studies conducted in the literature that aims to reduce latency and power usage when accessing shared memory. Two potential methods for this are improving cache replacement algorithms and optimizing cache architecture.

Many multi-core systems utilize Associative Memory (AM) cache as a way to share memory [1, 2]. The architecture of enhanced caching seeks to facilitate parallel searching and faster retrieval [3]. In contrast, replacement algorithms are employed to aid the cache controller in deciding which data to eliminate in order to make space for new data [4, 5]. Moreover, an effective replacement algorithm can reduce the latency of cache access. Content addressable memory (CAM) is a type of AM that accesses memory locations by comparing tags (parts of the content) rather than calculating the address and has certain properties that make it suitable for use as a shared memory [3, 6, 7]. The use of CAM memory in shared memory for multi-core systems is interesting, as demonstrated by other relevant articles that have recently been published by the authors [8, 9].

1.1 Types of shared cache memory

In contrast to traditional memory architectures, such as Static Random Access Memory (SRAM) and Dynamic RAM (DRAM), which use unique addresses to retrieve and store data, content-operated memory (COM) uses a different approach. COM allows stored data to be accessed based on part of its content, instead of an address [1]. COM is used in a variety of digital computer applications, from branch prediction techniques to very-high-speed parallel systems, to perform two primary memory-related operations: writing (storing data) and reading (accessing the correct corresponding data) when the address is not known [3]. The major application of COMs is packet switching routing and classification on network systems [10]. It is anticipated that COM memory will be used in upcoming applications for non-CMOS next-generation electronic devices [3]. COM memory architectures can be divided into two main categories: AM and CAM. Both of these types of memory perform the same functions, but they do so in different ways.

AM memory is further divided into three categories: direct-mapped (DM), set associative (SA), and fully associative (FA). Each of these memory types has different restrictions on where data can be written, as well as different replacement algorithms that are used. DM memory only allows for one location for a particular data item. FA memory allows for data to be mapped to any location. SA memory allows for a set of possible locations for data to be stored. In the following subsections, a brief overview of each of these three main types of cache memory will be provided.

1.1.1 Fully associative memory

The FA cache memory design stores the address and data in the same cache location, and compares the incoming address with all addresses stored within each location. As shown in **Figure 1**, this type of caching architecture is associated with high performance in comparison with its size; however, its design complexity is a major drawback. To counteract this, Random, First in First out (FIFO), and the Least Recently Used (LRU) algorithms are employed to determine where data should be stored [2].

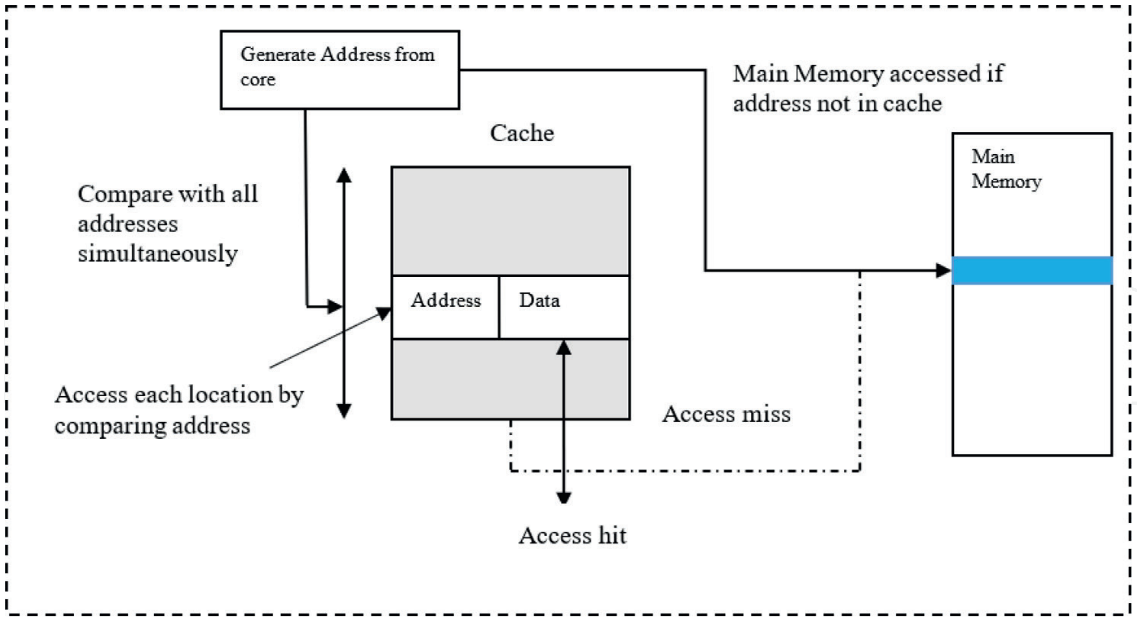
Generate Address from core
Main Memory accessed if address not in cache
Cache
Main Memory
Compare with all addresses simultaneously
Address
Data
Access each location by comparing address
Access miss
Access hit

Figure 1.
FA architecture.

FA caches are rarely utilized in multi-core processors due to their lower cache hit rate. Whenever a new memory is referenced to the same cache location, the cache line is replaced, leading to an increased miss rate [1, 2].

1.1.2 Direct mapped memory

In this type of system, the main memory is divided into blocks, and the cache is divided into a set of lines. This means that each cache line can hold one block of the main memory. Rather than storing the full address in the address field, only a part of the address bits is stored alongside the data field [1, 2] shown in **Figure 2**. Direct mapped caching has the benefit of being both simple and cost-effective to implement;

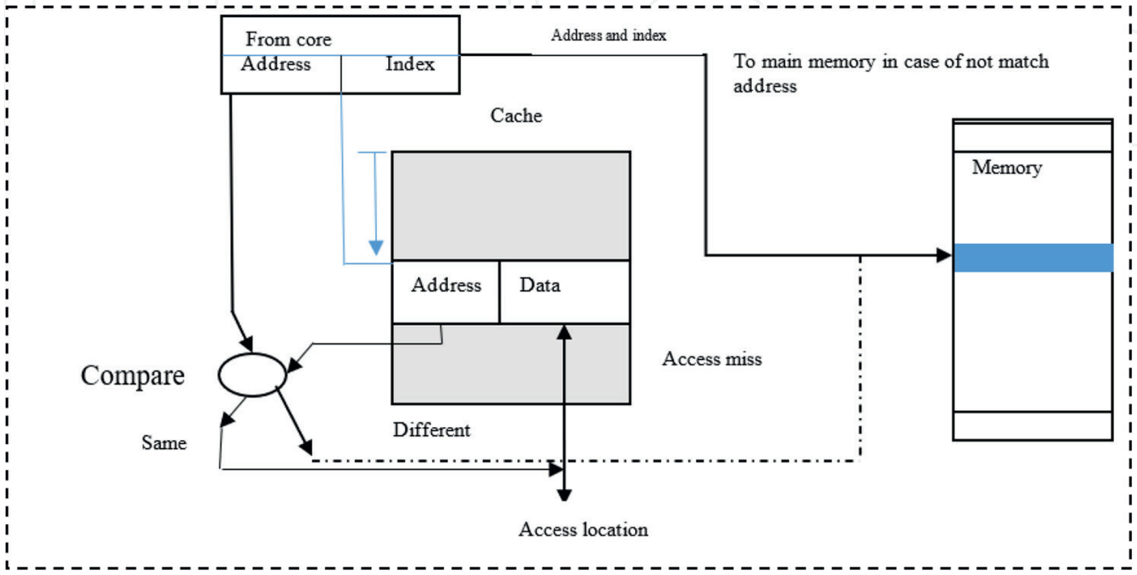

Figure 2.
Direct mapped cache memory.

however, if access to different locations with the same index is attempted, its performance will suffer.

1.1.3 Set-associative memory

Set-associative (SA) caching is a hybrid between full associativity and direct mapping. It splits the cache into a set of lines, allowing one block of main memory to be stored in n potential sets. Compared to a FA cache, it is less complex and can provide better performance since multiple addresses can be stored under the same index. However, its cost increases as the set size grows, as well as its access latency since it has to compare each address in all sets after its index is generated. Despite this, many commercial multi-core systems still use SA caches due to their improved performance [1, 2]. **Figure 3** represents the SA cache memory.

In both FA mapping and SA mapping, there are multiple options for where data can be stored, so replacement algorithms must be used to decide which location should be chosen.

1.1.4 Content-addressable memory

CAM is a type of memory whose locations can be accessed by comparing tags that are parts of the contents, rather than supplying their addresses. In some ways, CAM is similar to direct memory (DM) in its form; both allow for the instant retrieval of an output based on the input. However, both DM and CAM use different methods to facilitate the parallel search and quick storage [2, 3]. DM prevents the storage of particular data in just one location; conversely, CAM has no bounds on where data can be stored. Similarly, CAM and FA are comparable in that they both have no constraint on where data can be saved. Additionally, they both use analogous update and replacement strategies such as random, FIFO, and LRU to replace data when memory is full or the data becomes no longer useful. These algorithms will select a line that is unlikely to be needed in the near future, from all the lines stored in memory [5].

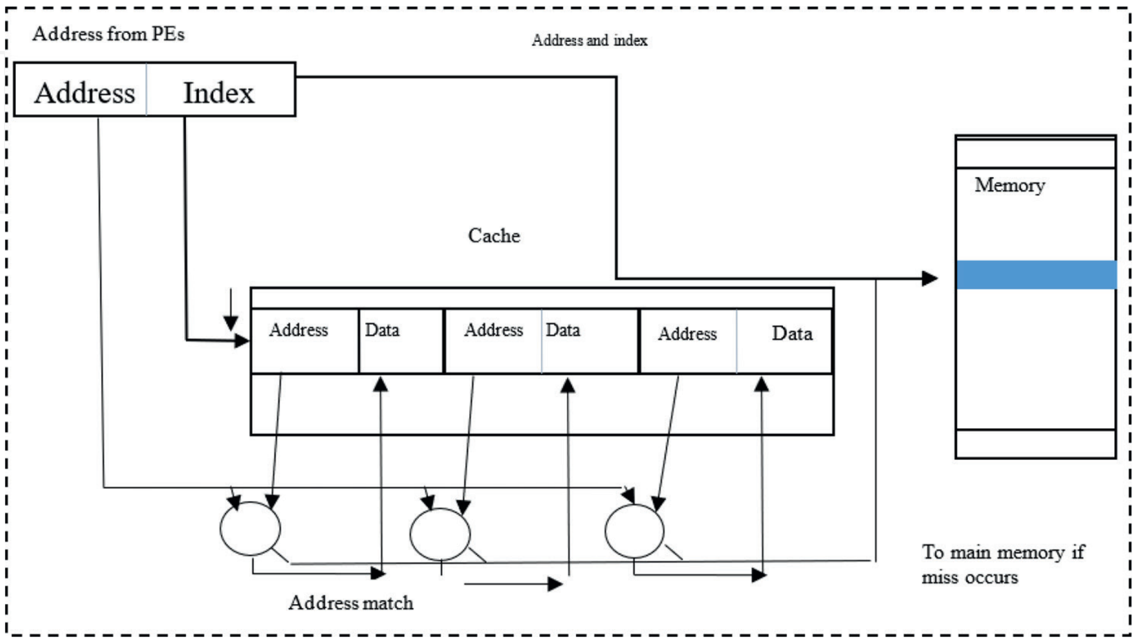

Figure 3.
SA cache memory.

Transactional memory (TM) is a new, emerging type of memory associated with CAM. It is not much different from CAM, but it is used to allow data sharing between processors in a distributed system. TM is used as autonomous storage memory with various hardware components [11, 12].

CAM memory is used for a variety of applications, including image processing, signal processing, pattern recognition, switching network techniques, and parallel processing systems. Unlike traditional SRAM, CAM memory searches through the content of data rather than its address, allowing for parallel and simultaneous search. This makes it a powerful tool that can quickly search through memory contents [3, 13–15].

A unique tag is assigned to each data in a CAM. To read the data, a read signal and the tag are applied to all locations at the same time, and then, the applied tag is compared with all of the previously stored tags. If a match is found, the data in the matched location is selected, output on the data bus, and read by the core. **Figure 4** displays the architecture of a CAM with a single port. It was not previously possible to make CAMs as a standalone memory in any system because a large number of pins were required; however, with the advances in semiconductor technology and FPGAs, researchers are now able to implement CAMs in FPGAs [7, 16]. These types of memory improve the search rate and reduce the processing latency and sometimes the power consumption.

The SA cache is the most popular architecture type used as shared memory in multi-core systems [1, 3]. It still suffers from many problems such as increasing access latency and contention if more than one core tries to access the same shared memory simultaneously. These problems are solved using the proposed DPCAM architecture.

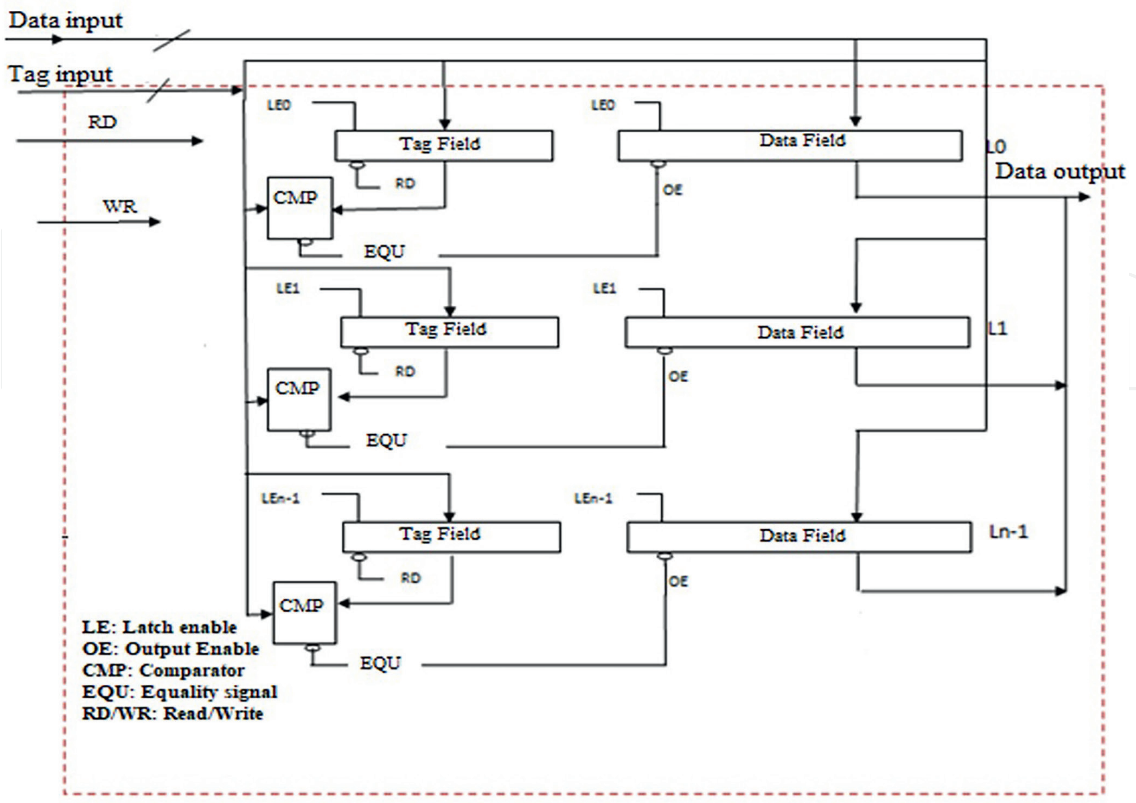

Figure 4.
CAM architecture.

2. Proposed DPCAM

As demonstrated in **Figure 5**, a Dual-Port Content Addressable Memory (DPCAM) can be used with a separate, pipelined shared cache. A Tag Field, Data Field, control unit (CU), comparator (CMP), Tag Field, Data Field, and two ports (Ds_{31}-Ds_0 for writing and Dd_{31}–Dd_0 for reading) are all included in the cache. The core sends Data source [Ds_{31}–Ds_0] and Tag source [Ts_{15}–Ts_0] to be written to the chosen cache line during the Store Back (SB) stage. The cached data is read to the destination data bus [Dd31–Dd0] during the Operand Fetch (OF) step, while the core simultaneously sends the Tag destination [Td_{15}–Td_0] for comparison with each cache line. Both ports have the ability to operate simultaneously.

The Data Field and the Tag Field are the two components that make up a cache line (L). The common data is kept in the Data Field, and each Data Field's specific tag (data and version number) is kept in the Tag Field. Depending on the sort of architecture the CAM is used in, the length of each field can be altered. A 24-bit tag, for instance, can hold up to 16 Mega versions of shared material. For reading operations, a 2 × 1 CMP is included that compares the tags from the OF stage [Td_{15}-Td_0] to those kept [Ts_{15}–Ts_0] in the cache lines.

The CU of the DPCAM design is a crucial component that is responsible for both managing the writing process and executing the replacement algorithm. Its goal is to generate an active signal in a cyclic pattern for each cache line. The control circuit is used to select which position to write the data to, as illustrated in **Figure 6**. The locations are chosen in order and are rewritten if needed to update their contents. This is accomplished by employing a collection of D Flip Flops (D-FF), each of which points

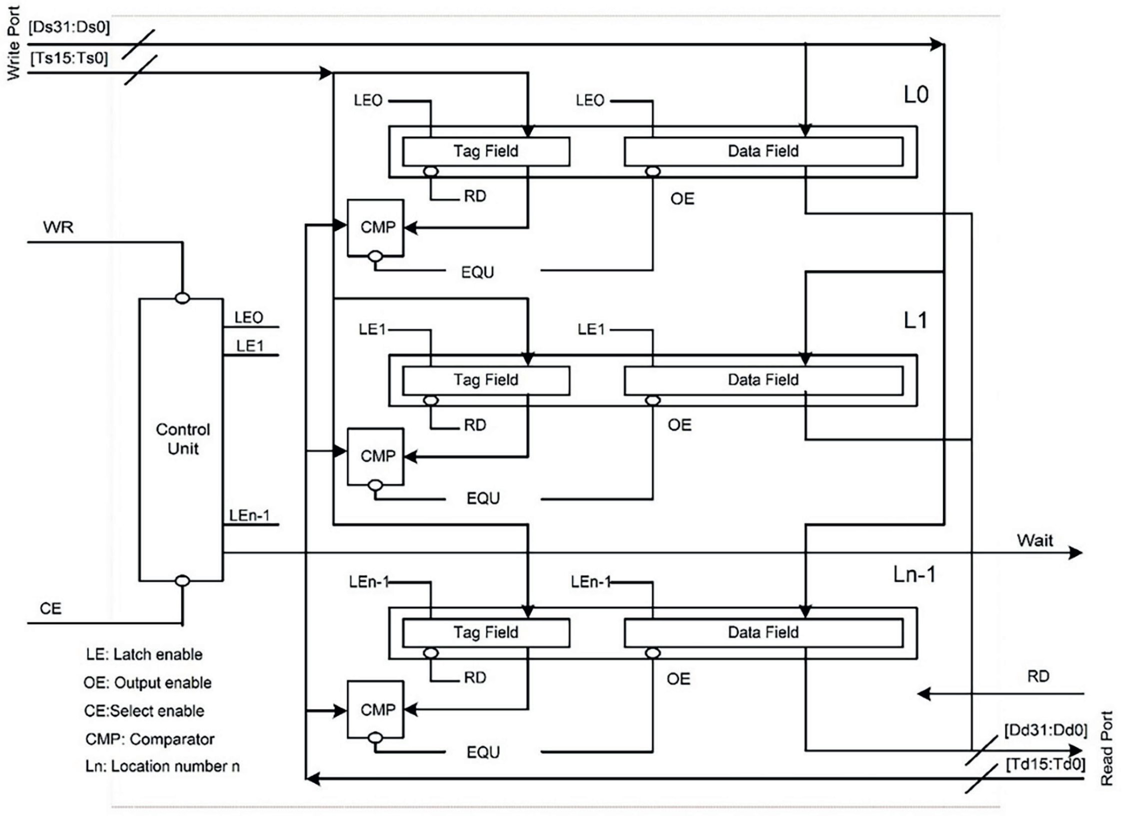

Figure 5.
DPCAM design.

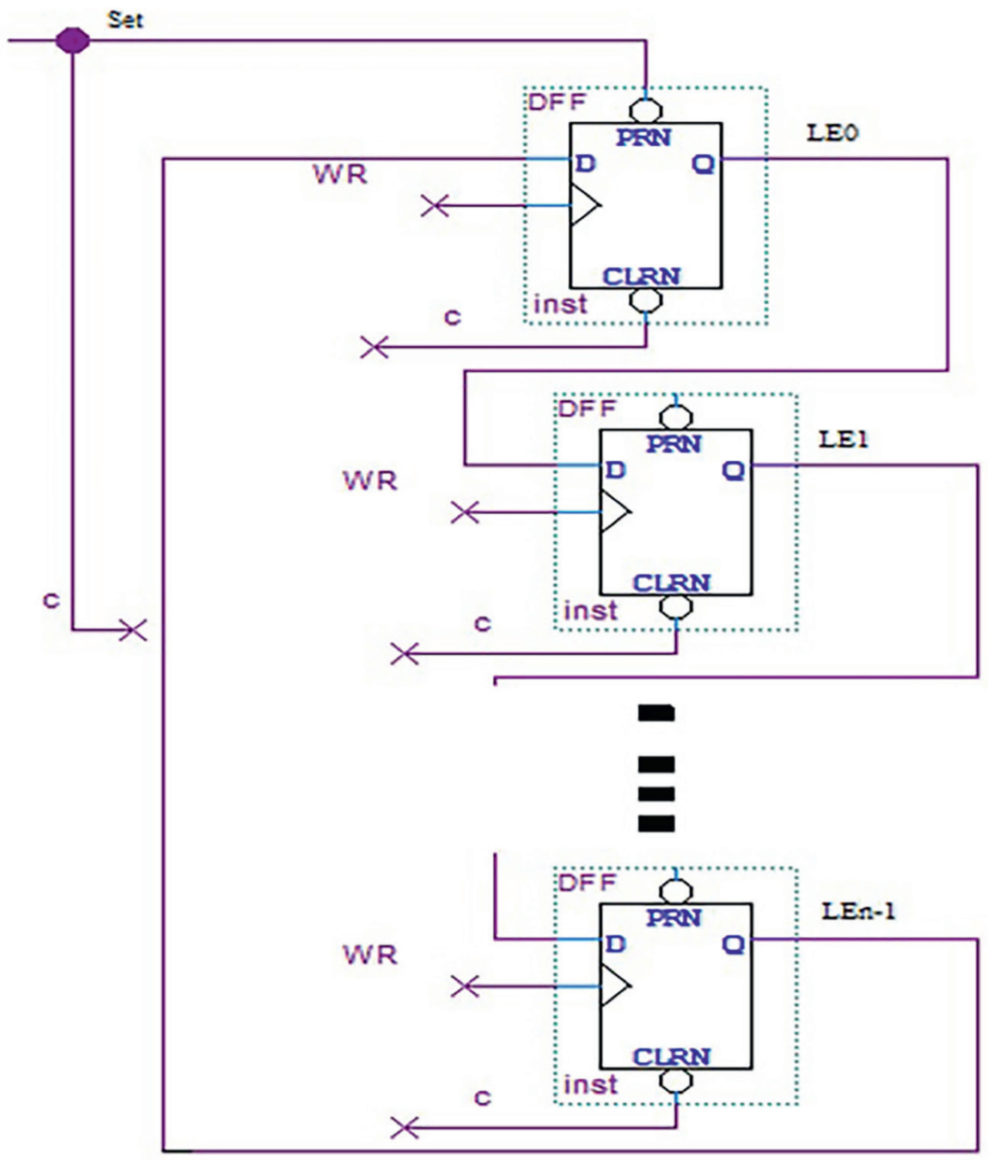

Figure 6.
Control unit.

to an associated DPCAM location. The first writing operation will be done on line L0 when the system is initially powered on, with the pointer indicating the first line, LE0. After writing to the current line, the pointer will shift to the next line and so on until the n − 1th line (Ln − 1) is reached.

The Write (WR) signal is transmitted by the CU from the first port (write port) in DPCAM, which is responsible for the writing operation. The stage buffer (SB) unit supplies the source data [Ds_{31}–Ds_0], the source tag [Ts_{15}–Ts_0], and an active low WR signal. The control circuit will then switch the LE to LE1 in order to write to line 1 whenever the WR signal reaches its negative edge (marking the end of the writing procedure). The output (OF) unit of the reading core will send an active high read (RD) signal and the tag destination (Td_{15}–Td_0) to all Tag Fields during the reading operation. The applied data's tag is compared to the tags kept in the memory lines. If a match is found, the CMP of each memory line will output an output enable (OE) signal. The data kept in the Data Field is then output *via* this signal to the destination data bus [Dd_{31}–Dd_0] for the reading core's OF unit to read. If the same memory address is requested for both reading and writing, the CU will give priority to the writing process and signal a WAIT to the reading operation. Both read and write ports can operate simultaneously if separate memory locations are requested for reading and writing, which lowers the cache access latency. While the SB unit of the writing core delivers the data [Ds_{31}–Ds_0] and the tag [Ts_{15}–Ts_0] to the precise position designated by the CU, the OF unit of the reading core concurrently transmits the destination tag [Td_{15}–Td_0] and the RD signal to all tag fields. This makes it possible to read the stored data from the target data bus [Dd_{31}-Dd_0].

The proposed architecture features a new, small DPCAM in place of the cache controller, which collects data from lower-level memory and increases access latency. Close-access data is stored in the main DPCAM module, as indicated in **Figure 7**, and far-access data is stored in the new module. As far-access data is used less often than near-access data, the far-access module is generally smaller. To illustrate, a four-core processor with a 64 KiB shared DPCAM can store 2 K operands, each composed of eight bytes of data plus a tag, before the data must be rebuilt.

The NFRA algorithm is implemented at the hardware level to reduce cache access latency. This technique involves writing a CU and pointer to position Lx, followed by instructions that write their operands to Lx + 1 to Ln − 1. After reaching the last position, the pointer returns to LE0 and overwrites the previous data and tags. Compared to a complex algorithm in the cache controller, this method is used for both near-access and far-access modules and has lower costs and access overhead. Moreover, it facilitates the storing, loading, and retrieving of near-access and far-access data/tags from various DPCAM modules [1]. According to the migration principle, the far-access module can be activated as needed and then switched to an inactive mode to conserve power when not in use [17, 18].

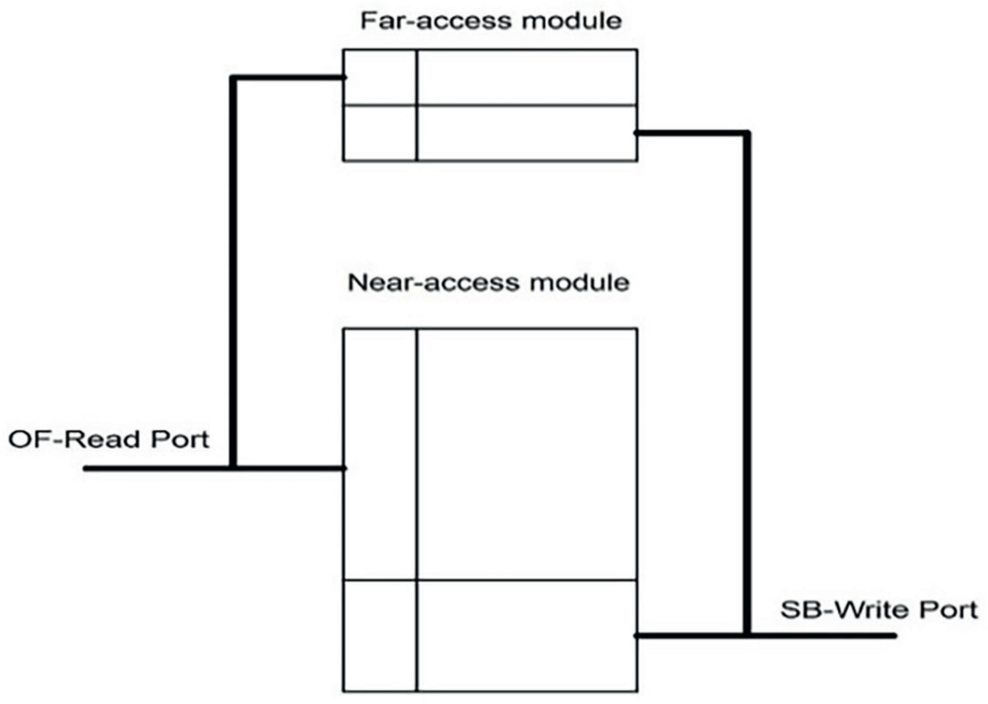

Figure 7.
Near-access, far-access DPCAM modules.

3. Implementation of the DPCAM and performance analysis

Quartus Prime 19.1 was utilized in the development, testing, and validation of DPCAM using Intel's FPGA Cyclone V family with 28 nm technology. ModelSim, Intel's design and simulation software, was employed to construct and examine DPCAM as a single memory [19]. In order to determine if DPCAM could replace the shared cache in the memory hierarchy of a multi-core CPU, two cores were utilized to measure the latency of read and write operations. Block schematics and Verilog Hardware Description Language (VHDL) files were implemented to develop the device, while Model Sim and Vector Wav File (VWF) were used for functional and timing simulation verification and debugging. The Power Analyzer Tool was also employed to evaluate the DPCAM's static and dynamic power consumption, and a tests-bench was created to simulate and analyze the latency of its reading and writing operations. To compare the performance of DPCAM, the SA cache, the most popular architecture type used as shared memory in multi-core computers, was employed [1, 2].

3.1 Functional assessments

The test-bench program was used to simulate the operations of the DPCAM and evaluate its latency and power usage. Firstly, it reset the CU and then created random 16-bit tags and 32-bit data to carry out write operations. Read/write signals were generated until the end of the simulation time and the output for the read operations was generated by comparing the 16-bit tags with the stored tags. It was used to compare the SA cache with the LRU replacement mechanism, and DPCAM with NFRA. The usage of the test-bench program is depicted in **Figure 8**. Through numerous simulations, it was demonstrated that the DPCAM's reading, writing, simultaneous read-write, CU, and replacement algorithms all operated as intended. **Figure 9** displays a 10 ns clock period of multiple clock cycles for reading and writing operations to the 64 KiB DPCAM. The CU was set to the first location in the first interval (0–10 ns) with the written data (out). The processor loaded the relevant tag (tagd) of the written data (outI) and used the RD signal to read the data from the desired out DPCAM location during the second interval (10–20 ns) using the WR signal. As soon as the RD signal went high, the processor output busses (outE) released the stored information. During interval 4 (30–40 ns), multiple DPCAM locations underwent read and write operations, with a write operation taking priority over the read operation. As a result, new data with a tag of [0]13 was written to the target location, while data previously written with a tag of [0]12 was accurately read. In interval 5 (40–50 ns), both read and write operations were performed on the same spot simultaneously.

3.2 Latency assessments

The performance of the DPCAM with regard to timing evaluation was evaluated using the Intel Quartus Prime Timing Analyzer. For each component of the design, this tool reports on all necessary data times, data arrival times, and clock arrival times using industry-standard constraint and analysis methodologies (Intel, 2021). The access latency for read and write operations in the DPCAM structure was measured using the Timing Analyzer, and real signal arrivals were compared to the design restrictions.

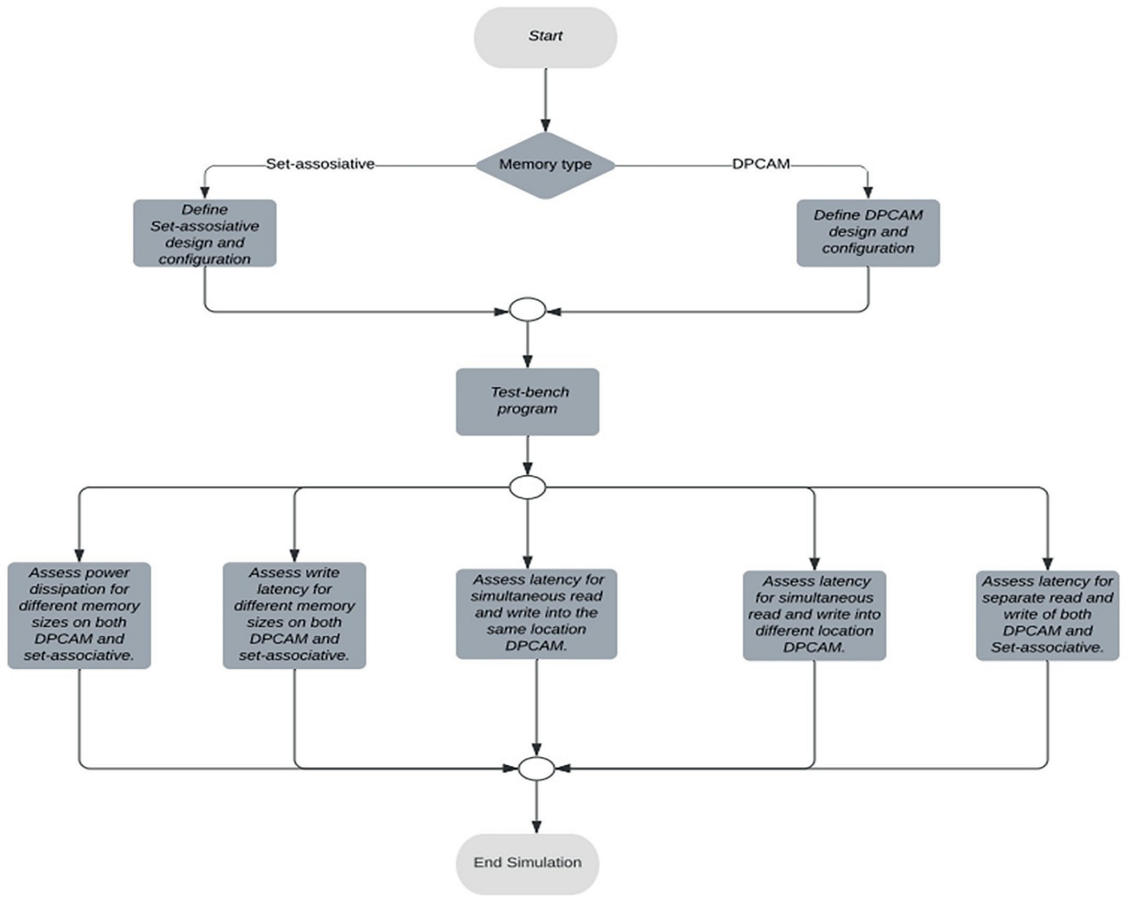
Start
Memory type
Set-assosiative
DPCAM
Define Set-assosiative design and configuration
Define DPCAM design and configuration
Test-bench program
Assess power dissipation for different memory sizes on both DPCAM and set-associative.
Assess write latency for different memory sizes on both DPCAM and set-associative.
Assess latency for simultaneous read and write into the same location DPCAM.
Assess latency for simultaneous read and write into different location DPCAM.
Assess latency for separate read and write of both DPCAM and Set-associative.
End Simulation

Figure 8.
Test-bench program.

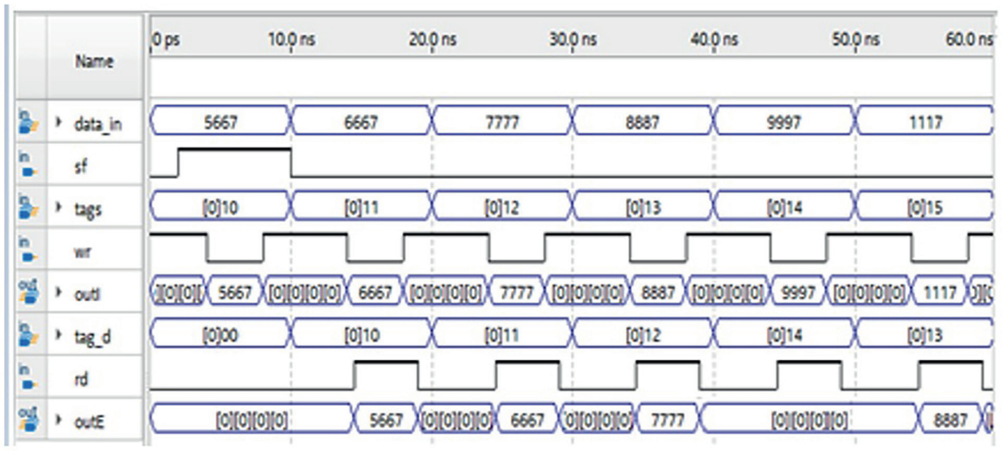

Figure 9.
Functional assessments.

To determine the delay of read and write operations, a 64 KiB near and distant DPCAM module timing simulation was performed using Intel's Cyclone V FPGA, as shown in **Figure 10**. After running the simulator 100 times, it was found that the average delay time for writing on DPCAM was 0.9529 ± 0.03393 ns. The WR signal was then turned off. The average latency for a read operation was found to be 1.1782 ± 0.08830 ns when the tag ([0]10) in the second interval was compared with

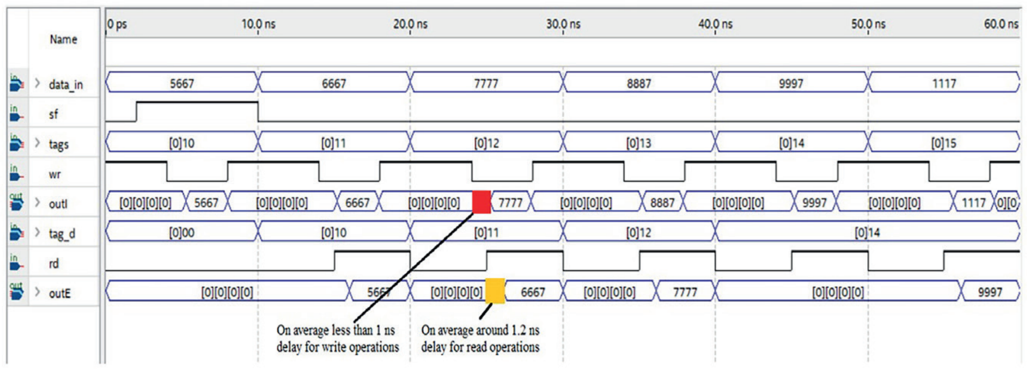

Figure 10.
Latency assessments.

the tags in all places with the RD signal. The fourth and fifth intervals were used to assess the latency of simultaneous read and write operations, and the average delay time was calculated to be the time between initiating a request for data and the actual data transfer for a single operation and the time between two requests for simultaneous write and read operations.

- $l_{SDL} = \max\left(l_{WR} + l_{RD}\right)$ SDL is referred to simultaneous access RD/WR operations to the different memory lines
- $l_{SSL} = t_{CL} + l_{RD}$ SSL is referred to simultaneous access RD/WR operations to the same memory lines

Where l_{WR} is a latency of a write operation, l_{RD} is a latency of a read operation and t_{CL} is the cycle time.

Hundred simulations in two separate modes with write and read operations to unique or similar memory locations were run. The write and read latency for the latter mode were 0.98280.0412 ns and 1.2226 ± 0.09446 ns, respectively, with an overall latency of l_{SSL} = 11.2226 ± 0.09138 ns according to the T-test and 95% confidence interval. The average write latency was 1.9434 ± 0.0382 ns, and the average read latency was 2.15840.1056 ns, according to tests with a 64 KiB four-way SA cache. Unfortunately, due to SA cache limitations, simultaneous read and write operations could not be tested. The tested DPCAM has a lower read latency than the tested SA cache; this is because DPCAM compares the incoming tag directly with the stored tag, whereas SA caches must use an index to access the location with a tag to compare to, which increases the latency. Generally, a cache memory based on AM has a latency of around 2 ns for 64 KiB [20], 1.66 ns for AM with 1KiB, and 1.69 ns for 4-way set associative with 2 KiB, which is used in cache controllers [21]. However, the write latency for a cache memory based on AM typically exceeds 2 ns for 64KiB [20].

Using FPGA technology, a comparison of the write latency between a typical four-way set associative cache and a DPCAM design was made for equivalent-sized caches. Because the CU points directly to the memory location and does not need to generate the address of the following write site as is necessary for the AM cache memory, simulations have shown that DPCAM has a low and consistent write latency for variable memory sizes. As seen in **Figure 11** and **Table 1**, the write latency difference between DPCAM and the SA cache widens as memory capacity grows.

In order to assess the latency of write and read operations, the NFRA replacement method used by DPCAM and the LRU algorithm employed by the set associative cache memory were compared. The results showed that the set associative cache had an average latency of 0.9529 ± 0.03393 ns for a write operation and 1.1782 ± 0.08830 ns for a read operation, whereas the DPCAM recorded a lower access latency for a size of 64 KiB, with a latency of 1.9434 ± 0.0382 ns for a write operation and 2.1584 ± 0.1056 ns for a read operation.

3.2.1 Descriptive statistics

About 100 times were spent running the simulator with various test-bench values, documenting the latency for write and read operations as well as for simultaneous read and write operations into distinct memory locations and the same memory location. In order to determine the minimum, maximum, mean, and standard error for DPCAM and SA architecture, data analysis was done using SPSS and *T*-test tools. **Table 2** displays descriptive data for write latency in DPCAM, **Table 3** describes descriptive statistics for read latency, **Table 4** describes descriptive statistics for simultaneous read and write operations into distinct memory locations, and **Table 5** describes descriptive statistics for simultaneous read and write operations into the same memory regions. Similar descriptive statistics for write and read latency in set associative are shown in **Tables 6** and 7. The write and read operations between DPCAM and SA memory were

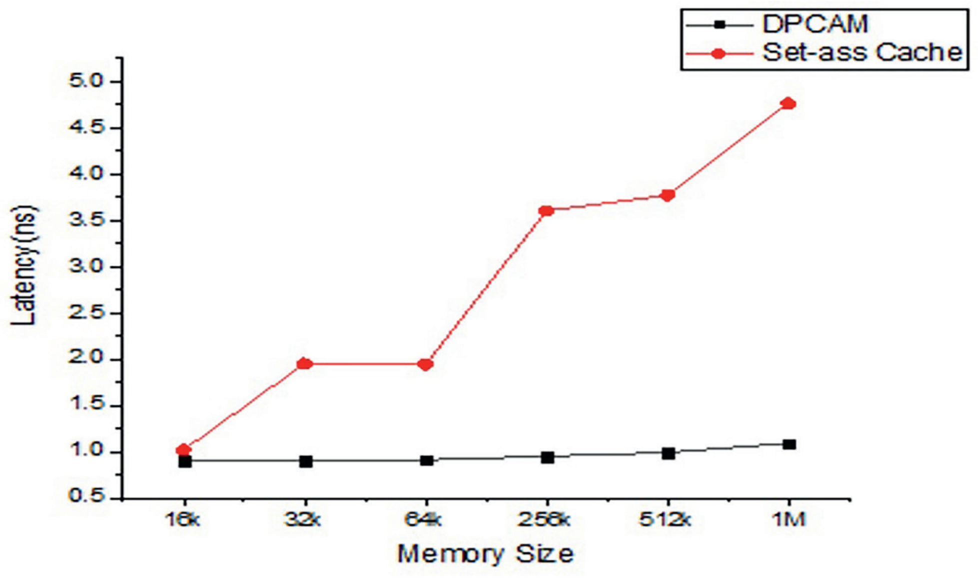

Figure 11.
Write access latency (ns).

Size	DPCAM	Cache	Size	DPCAM	Cache
16 K	0.90	1.02	256 K	0.95	3.60
32 K	0.90	1.95	512 K	0.99	3.77
64 K	0.91	1.94	1 M	1.09	4.76

Table 1.
Write operation access latency (ns).

	N	**Minimum**	**Maximum**	**Mean**	**Std.**	
	Statistic	**Statistic**	**Statistic**	**Statistic**	**Std. error**	**Statistic**
Write DPCAM	100 100	0.89	1.10	0.9529	0.00339	0.03393

Table 2.
Descriptive statistics for write latency in DPCAM.

	N	**Minimum**	**Maximum**	**Mean**	**Std.**	
	Statistic	**Statistic**	**Statistic**	**Statistic**	**Std. Error**	**Statistic**
Read DPCAM	100 100	1	1.35	1.1782	0.00883	0.0883

Table 3.
Descriptive statistics for read latency in DPCAM.

	N	**Minimum**	**Maximum**	**Mean**	**Std.**	
	Statistic	**Statistic**	**Statistic**	**Statistic**	**Std. error**	**Statistic**
Write/read different location DPCAM	100 100	1	1.40	1.2262	0.00945	0.0945

Table 4.
Descriptive statistics for simultaneous access latency Rd/Wr operations in the same DPCAM location.

	N	**Minimum**	**Maximum**	**Mean**	**Std.**	
	Statistic	**Statistic**	**Statistic**	**Statistic**	**Std. Error**	**Statistic**
Write/read same location DPCAM	100 100	0.91	1.10	0.9828	0.00412	0.0412

Table 5.
Descriptive statistics for simultaneous access latency Rd/Wr operations in the same DPCAM location.

compared using the T-test, and it was discovered that DPCAM had lower write and read latencies than those of SA memory with a 95% confidence interval.

3.3 Estimation of a power dissipation

Power management is essential to achieving better size, performance, and affordability while maintaining a high power density as chip technology continues to get smaller. The Quartus simulator's Power Analyzer Tool estimates power dissipation with an accuracy of 10% to make sure components use the right amount of power and enhance the design [22]. Based on the waveform file generated by Model Sim while

	N	Minimum	Maximum	Mean		Std.
	Statistic	Statistic	Statistic	Statistic	Std. error	Statistic
Write SA	100 100	1.88	2.10	1.9434	0.00382	0.0382

Table 6.
Descriptive statistics for write latency in SA.

	N	Minimum	Maximum	Mean		Std.
	Statistic	Statistic	Statistic	Statistic	Std. error	Statistic
Read SA	100 100	1.95	2.35	2.1584	0.01056	0.1056

Table 7.
Descriptive statistics for read latency in SA.

simulating the script with the DPCAM design at the gate level, the Power Analyzer Tool was used to assess static, dynamic, I/O, and overall power consumption.

This section compares and evaluates the power dissipation of DPCAM and four-way SA caches with various memory capacities. The DPCAM dissipates electricity through near-far access modules. Static power, which is the leakage power of the functional unit on the FPGA excluding the I/O port, is the thermal energy used on the chip. Dynamic power is the amount of energy used when a unit is in use or when a signal is changing. The pins, which power components on and off-chip and have an impact on dynamic power, produce I/O power [22].

In **Figure 12**, the static, dynamic, and I/O power dissipation of DPCAMs and SA caches can be compared. From **Figure 12a**, it is evident that DPCAMs have a higher static power dissipation than SA caches. This is because increasing the size of the DPCAMs leads to the complexity of the hardware created by the CU and internal wires covering a larger surface, resulting in an increased static power dissipation. **Tables 8** and **9** further provide a comparison of the static, dynamic, I/O, and total power dissipation of DPCAMs and SA caches, respectively, for different sizes. **Figure 12b** compares the dynamic power dissipation of DPCAM and SA. It can be observed that when the size is less than 512 K, the dynamic power of DPCAM is similar to that of SA. However, after 256 K, it increases significantly due to

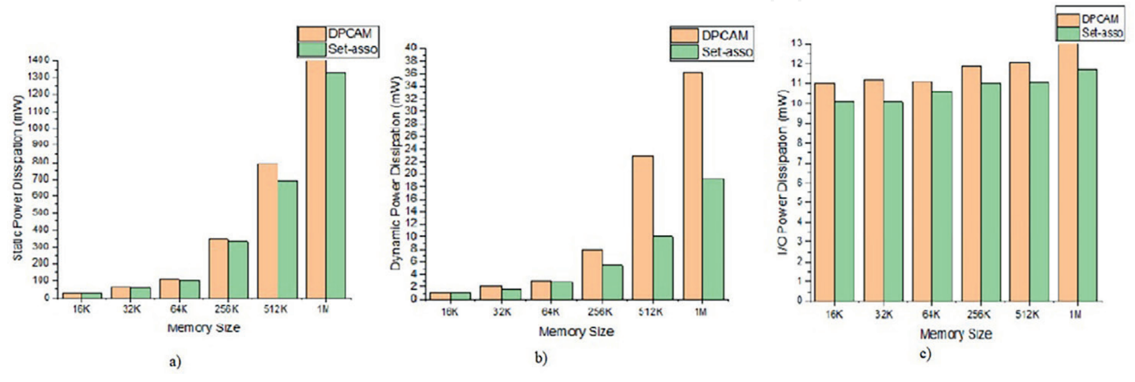

Figure 12.
Power dissipation with variation in memory size: (a) Compared the static power dissipation between DPCAM and SA. (b) Compared dynamic power dissipation. (c) It is compared I/O power dissipation.

	Power in milliwatts (mW)			
Size	**Static**	**Dynamic**	**I/O**	**Total**
16 K	32.214	1.13	11	44.344
32 K	64.33	2.14	11.21	77.68
64 K	107.57	2.99	11.10	121.66
256 K	349.5	7.98	11.88	369.28
512 K	796.2	22.90	12.07	831.17
1 M	1411.10	39.26	13.21	1463.57

Table 8.
DPCAM power dissipation.

	Power in (mW)			
Size	**Static**	**Dynamic**	**I/O**	**Total**
16 K	28.9166	1.12	10.1	40.1366
32 K	57.33	1.62	10.1	69.05
64 K	99.41	2.79	10.6	112.8
256 K	334.750	5.48	11	351.23
512 K	696.261	10.021	11.025	771.307
1 M	1325.310	19.28	11.737	1356.326

Table 9.
SA memory power dissipation.

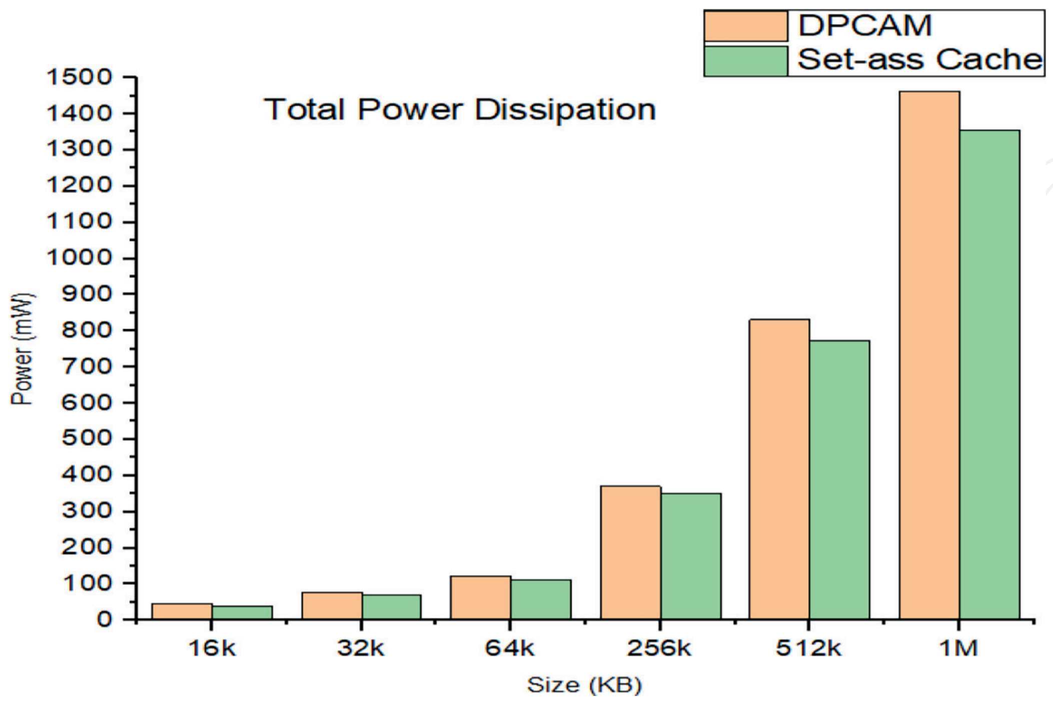

Figure 13.
Total power dissipation with variation in size.

numerous active locations being accessed during read operations. **Figure 12c**'s comparison of I/O power dissipation reveals that the DPCAM's I/O power is comparable to the SA with varied sizes, since the off-chip pins remain constant regardless of the internal memory capacity. **Figure 13** indicates that the total power used by DPCAM is only marginally higher than that of the SA cache, at around 7%. This small increase in power dissipation can be managed through power-saving techniques, such as those found in refs. [17, 18, 23–25], thus not prohibiting the adoption of DPCAM in multi-core systems.

4. Conclusion

A design of a special purpose-shared memory architecture based on CAM and a replacement algorithm has been presented in this chapter. This architecture was designed to enable multi-core processors to access the cache memory with lower latency than the traditional SA cache. It should be stressed that while all of the previous replacement algorithms do not make use of cache hardware architecture, they increase non-computational times for updating the location and introduce new access overhead. In order to lower the cost overhead and complexity of the cache controller, Near-Far Access Replacement Algorithm (NFRA) is also suggested and implemented as a hardware component inside the DPCAM module.

Testing the design on a Cyclone V Intel FPGA showed that the DPCAM could replace the shared cache in the memory hierarchy of a multi-core processor. The DPCAM had an average latency of 1.2 ± 0.09138 ns for reading operations and 0.9679 ± 0.0642 for writing operations, which is better than other types of shared memory. Furthermore, the access latency for a write operation was almost the same regardless of the memory size. Although the DPCAM consumes more power than the SA memory, some power-saving techniques can be used to reduce this amount.

Author details

Allam Abumwais* and Mahmoud Obaid
Computer Systems Engineering, Arab American University, Jenin, Palestine

*Address all correspondence to: allam.abumwais@aaup.edu

References

[1] Patterson DA, Hennessy JL. Computer Organization and Design The Hardware Software Interface. 2nd ed. United States: Morgan kaufmann; 2020

[2] Stallings W. Computer organization and architecture. In: Wu H-K, Lee SW-Y, Chang H-Y, J, editors. Designing For Performance. 9th ed. United States: Pearson Education; 2013

[3] Karam R, Puri R, Ghosh S, Bhunia S. Emerging trends in design and applications of memory-based computing and content-addressable memories. Proceedings of the IEEE. 2015;**103**(8):1311-1330

[4] Olanrewaju RF, et al. A study on performance evaluation of conventional cache replacement algorithms: a review. In: 2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC). IEEE; 2016

[5] Priya BK, Kumar S, Begum BS, Ramasubramanian N. Cache lifetime enhancement technique using hybrid cache-replacement-policy. Microelectronics Reliability. 2019;**97**:1-15

[6] Abumwais A, Ayyad A. The MPCAM based multi-core processor architecture: A contention free architecture. WSEAS Transactions on Electronics. 2018;**9**:105-111

[7] Irfan M, Cheung RC, Ullah Z. High-throughput re-configurable content-addressable memory on FPGAs. In: Proceedings of the 2019 International Conference on Information Technology and Computer Communications. 2019

[8] Abumwais A, Amirjanov A, Uyar1 K, Eleyat M. Dual-port content addressable memory for cache memory applications. Computer, Material & Continua. 2021;**70**(3):4583-4597

[9] Abumwais A, Obaid M. Shared cache based on content addressable memory in a multi-core architecture. CMC-Computers, Materials & Continua. 2023;**74**(3):4951-4963

[10] Cheriton DR. U.S. Patent No. 9,111,013. Washington, DC: U.S. Patent and Trademark Office; 2015

[11] Nakaike T, Odaira R, Gaudet M, Michael MM, Tomari H. Quantitative comparison of hardware transactional memory for Blue Gene/Q, zEnterprise EC12, Intel Core, and POWER8. ACM SIGARCH Computer Architecture News. 2015;**43**(3S):144-157

[12] Papagiannopoulou D, Marongiu A, Moreshet T, Benini L, Herlihy M, Bahar RI. Hardware transactional memory exploration in coherence-free many-core architectures. International Journal of Parallel Programming. 2018;**46**:1304-1328

[13] Bhattacharya D, Bhoj AN, Jha NK. Design of efficient content addressable memories in high-performance FinFET technology. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2014;**23**(5):963-967

[14] Imani M, et al. Digitalpim: digital-based processing in-memory for big data acceleration. In: Proceedings of the 2019 on Great Lakes Symposium on VLSI. 2019

[15] Martyshkin AI, Salnikov II, Pashchenko DV, Trokoz DA. Associative co-processor on the basis of programmable logical integrated circuits for special purpose computer systems. In: 2018 Global Smart Industry Conference (GloSIC). IEEE; Nov 2018. pp. 1-5

[16] Ullah I, Ullah Z, Lee JA. Ee-tcam: An energy-efficient sram-based tcam on fpga. Electronics. 2018;**7**(9):186

[17] Luo JY, Cheng HY, Lin C, Chang DW. TAP: reducing the energy of asymmetric hybrid last-level cache via thrashing aware placement and migration. IEEE Transactions on Computers. 2019;**68**(12):1704-1719

[18] Ofori-Attah E, Bhebhe W, Opoku Agyeman M. Architectural techniques for improving the power consumption of noc-based cmps: A case study of cache and network layer. Journal of Low Power Electronics and Applications. 2017;**7**(2):14

[19] Cyclone V Device Overview. Available from: https://www.intel.com/content/www/us/en/docs/programmable/683694/current/cyclone-v-device-overview.html

[20] Cargnini LV, Torres L, Brum RM, Senni S, Sassatelli G. Embedded memory hierarchy exploration based on magnetic random access memory. Journal of Low Power Electronics and Applications. 2014;**4**(3):214-230

[21] Chauan P, Singh G, Singh GJ. Cache controller for 4-way set-associative cache memory. 2015;**129(1):8887**

[22] Quartus Handbook. Volume 3: Verification. Available from: https://www.mouser.com/pdfdocs/qts-qps-5v3.pdf

[23] Adegbija T, Gordon-Ross A. PhLock: A cache energy saving technique using phase-based cache locking. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2017;**26**(1):110-121

[24] Park J, Lee M, Kim S, Ju M, Hong J. MH cache: A multi-retention STT-RAM-based low-power last-level cache for mobile hardware rendering systems. ACM Transactions on Architecture and Code Optimization (TACO). 2019;**16**(3):1-26

[25] Rossi D et al. Exploiting aging benefits for the design of reliable drowsy cache memories. 2017;**37**(7):1345-1357