

M-way Set-Associative and Sector Mapped Cache in Verilog- Design and Testing

EE 275 Advanced Computer Architecture

Final Project Submission

By:

Kamlesh Kumar

Student ID- 017047212

December 12, 2024

Abstract

This project focuses on developing an advanced cache simulator to analyze the performance of both m-way set-associative caches and sector-mapped caches across various configurations. These configurations range from simple direct-mapped designs to fully associative and sector-mapped approaches. The simulator evaluates performance metrics such as hit ratio and miss ratio using Verilog, employing the Least Recently Used (LRU) replacement algorithm for set-associative caches and sector-specific strategies for sector-mapped caches. By experimenting with different cache sizes, line sizes, associativity, and sector-mapping techniques, we aim to understand their impact on cache performance and identify key factors that enhance efficiency.

Through detailed simulations and analyses, this study optimizes cache design by balancing hit rate improvement with minimal storage costs. For m-way caches, line size and associativity significantly influence performance, while sector mapping introduces a new dimension of block segmentation and mapping efficiency. Ultimately, the objective is to provide insights into cache operation and configuration, equipping system designers with the knowledge to make informed decisions and enhance overall system performance. GitHub Repository: github.com/kamlesh364/EE275.git

1. Introduction

The primary goal of this project is to develop a highly accurate cache simulator that evaluates the performance of m-way set-associative caches and sector-mapped caches. Cache memory, a critical component of modern computer architecture, significantly reduces memory access time compared to traditional hard drives. By enabling faster access to frequently used data, cache memory plays a vital role in enhancing system performance.

In this project, we present a Verilog-based cache simulator capable of analyzing two prominent cache memory designs:

- (1) **M-way Set-Associative Cache:** Configurable from direct-mapped ($m = 1$) to fully associative ($m = \text{cache size/line size}$), supporting associativity values such as 2, 4, 8, 16, and 32.
- (2) **Sector-Mapped Cache:** A design that divides cache lines into smaller sectors, optimizing storage efficiency and access speed for specific workloads.

The simulator supports cache sizes up to 128 KB and incorporates the Least Recently Used (LRU) algorithm for block replacement in set-associative caches. For sector-mapped caches, a sector-based management strategy ensures efficient handling of data blocks. To evaluate cache performance, the simulator uses an address trace file where each entry logs a byte address, and differences between consecutive addresses are recorded to optimize storage.

This dual-design simulator focuses on key parameters such as block (or line) size, cache size, associativity, and sector configuration. By analyzing the sequence of memory load instructions executed by programs, the simulator provides comprehensive insights into how these parameters influence performance.

The accompanying report presents a detailed analysis of simulation results, exploring the technical intricacies of m-way and sector-mapped cache designs. This documentation aims to aid developers and researchers in optimizing cache memory systems, advancing the understanding of cache behaviour and fostering innovation in computer architecture.

2. M-way Set Associative Cache

2.1.Design Methodology

Understanding cache design and operation is fundamental to mastering efficient memory management techniques. Cache memory is a specialized, high-speed memory that bridges the performance gap between the CPU and slower memory components, such as RAM and disk storage. While less costly than CPU registers, cache memory is more expensive than main memory but offers significantly faster access times. Positioned as a buffer between the CPU and RAM, it stores frequently accessed data and instructions, enabling the CPU to retrieve information quickly and thereby reducing the average time required to access main memory.

Cache memory operates by maintaining copies of frequently accessed data from main memory in a smaller, faster memory space. Modern CPUs typically include separate instruction and data caches to streamline processing. When the CPU requires data, it first checks the cache. If the requested data is found—a cache hit—the data is retrieved directly from the cache. Conversely, a cache miss occurs when the data is absent, prompting the cache to fetch it from main memory, create a new cache entry, and subsequently serve the request.

For instance, consider a system with a main memory comprising 128 words, where each block contains four words. This setup results in 32 memory blocks, each holding four words. If the cache size is limited to 16 words, the cache accommodates four lines ($16/4 = 4$), with each line corresponding to a block size of four words. The mapping of main memory blocks to cache lines is determined by the modulo operation ($k \bmod n$), where k is the block number, and n is the number of cache lines. In a direct-mapped cache, block 4 maps exclusively to line 0, as $4 \bmod 4 = 0$. Similarly, block 13 maps to line 1 ($13 \bmod 4 = 1$), irrespective of the occupancy of other lines.

This mapping principle, illustrated in Figure 1, demonstrates how main memory blocks are allocated to cache lines in a direct-mapped configuration. By efficiently managing these mappings and leveraging cache memory, systems achieve enhanced performance through reduced memory access times.

2.2.Design Description

Blocks of different sizes are used to segment caches. The number of blocks in a cache is typically a power of two. The simplest method is the Direct-Mapped Cache. In Direct Mapped Cache, every memory address correlates to a single cache block. One method for figuring out which cache block belongs at a given memory location is the mod (remainder) operation. The data at the memory address would be delivered to the cache block index if the cache contained 2^k blocks, giving $I \bmod 2^k$. At any one time, the CPU only needs a specific quantity of data. The cache memory duplicates the contents of the memory locations ahead of time to identify the range of memory locations that the processor will need shortly. The CPU loads data from the main memory into the cache first when it needs it, and if it cannot find it there, it must wait until the main memory is filled with the necessary data. At this point, data from nearby sites that match the requested address are also transferred to the cache. The basic idea behind cache operation is the locality of reference, sometimes referred to as the concept of locality. According to this principle, every program or application running on a processor ought to only use a tiny portion of the total address space at any one time. Reference locations come in two varieties.

- (1) Temporal locality, or time locality: There is a good chance that a certain data item will be needed again shortly if it is utilized at one point in time.
- (2) Spatial locality: There is a high probability that data items from nearby addresses will be needed shortly if a data item from a particular memory location is used at a given moment.

A cache controller receives the address when the CPU tries to read from memory. A block in the cache will be indexed using the lowest k bits of the address. If the block is valid and the tag matches the top $(m-k)$ bits of the m -bit address, the data will be sent to the CPU. What we've been expecting to be memory delays are cache hits. If the CPU implementations had gone straight to the main memory, the cycle times would have been much longer. However, a much slower main memory access is necessary upon a cache miss. A total of 2^n words and n -bit address lines make up the main memory. It is arranged into several blocks, with k words in each block. Consequently, its total number of main memory blocks is $\frac{2^n}{k} MB$. Similarly, a few lines, each containing k words, make up the cache memory.

There are far fewer cache lines (CL) than there are main memory blocks (MB) in total. Because of this, only a small number of main memory blocks are mapped to the cache at any given moment. The full block containing the requested data item is mapped to one of the cache lines when the processor starts a read request for data that is not in the cache. A cache line cannot be unique to a single main memory block due to the vast number of blocks compared to cache lines. Consequently, a tag designating the physical location linked to every cache line is assigned. How soon it returns the requested data is what defines the cache's performance. It is computed using the access time and hit/miss ratio. A cache hit occurs when the requested data is found in the cache; a cache miss occurs when it is not. The number of memory accesses that are discovered in the cache relative to the total number of memory accesses requested is known as the hit ratio. One may describe the miss ratio as $(1 - \text{hit ratio})$. Hit time is the amount of time it takes the cache to provide the requested data in the case of a hit. The necessary data is acquired from the main memory and mapped to the cache in the event of a cache miss. The time it takes to get and map the data is called a miss penalty. Conventional methods of enhancing cache performance (by raising the hit or lowering the miss ratio) are generally categorized as

- (1) Increasing block and cache size,
- (2) Increasing associativity,
- (3) Cache probing,
- (4) Supplementing the regular cache with victim cache,
- (5) Hardware data prefetching, and
- (6) Including additional cache hierarchy.

In the direct mapped cache, or one-way-set associative cache, every block from the main memory is allocated to a particular cache line. There are two benefits to the direct mapped cache: a straightforward design and extremely easy hardware implementation. But this architecture's mapping of every block from main memory to a single cache line is its lone flaw. The Direct Mapped Cache architecture is shown in Figure 1.

In set-associative cache design, the cache memory is divided into several tiny, direct-mapped modules, each of which is referred to as a set. A cache composed of M sets is referred to as M -way set associative. The performance of the set-associative cache lies in the center of these two cache architectures as it is essentially a compromise between completely associative and direct-mapped

caches. Two-way and four-way set-associative caches offer the best performance in terms of hit ratio and access time for embedded systems. A 4-way set associative cache's design is shown in Figure 2.

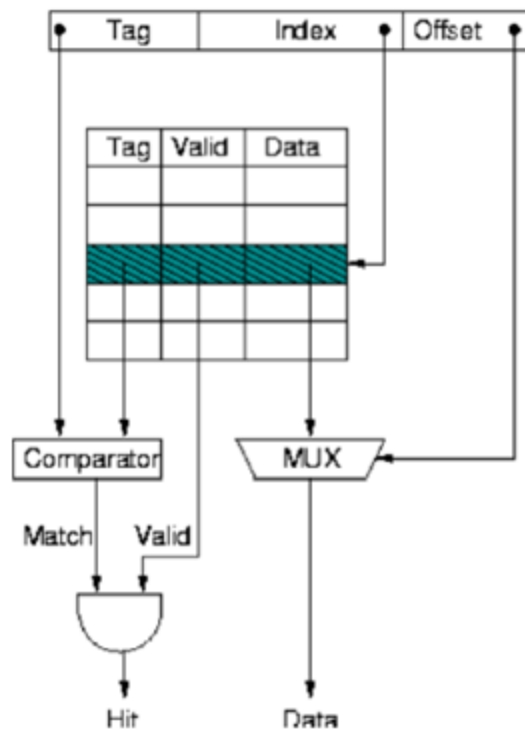


Figure 1: Architecture of Direct Mapped Cache.

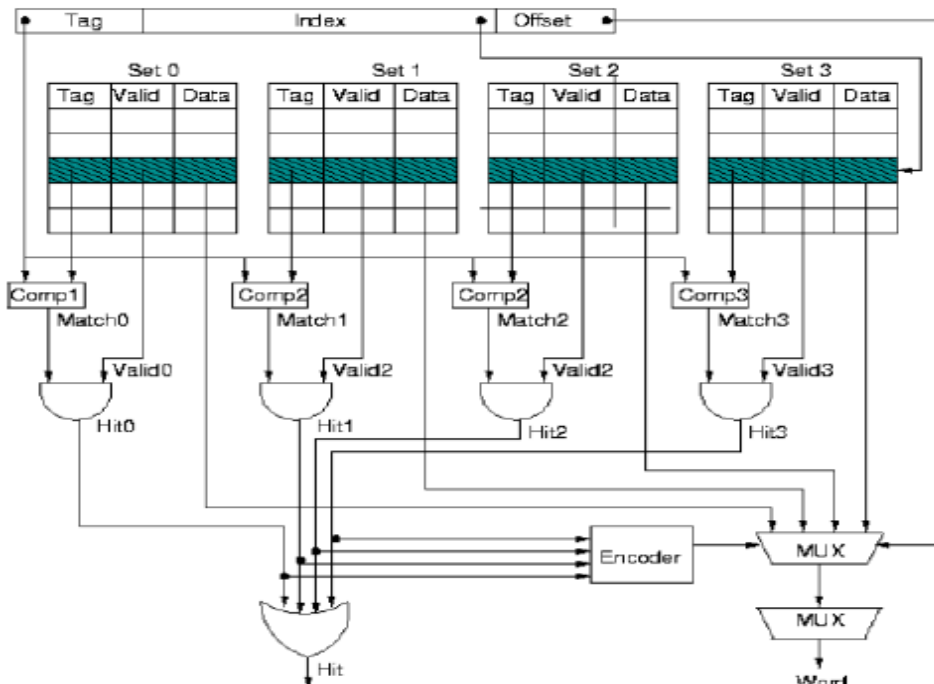


Figure 2: Architecture of 4-way set-associative cache.

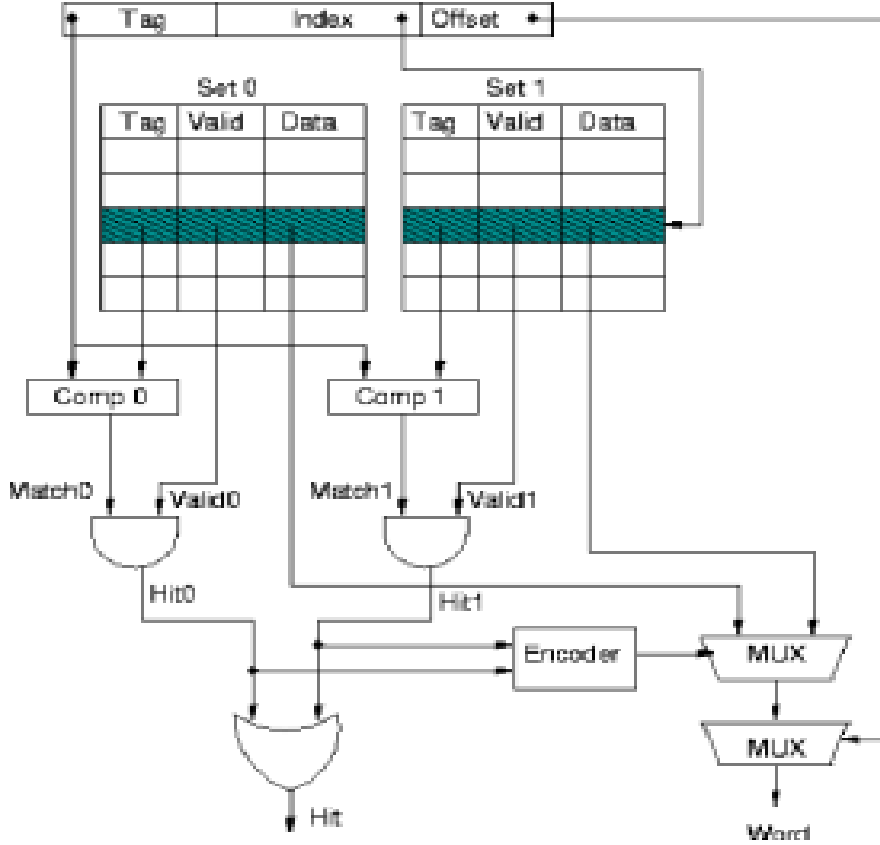


Figure 3: Architecture of fully associative cache.

In a completely associative cache, every block from the main memory can be allocated to any of the cache lines, giving the best possible hit ratio. It also involves the expense, intricacy, hardware, and access time related to looking up the requested location in the cache. Every cache block is compared simultaneously to determine if a requested memory location is present in the cache. The building of a completely associative cache is shown in Figure 3.

2.3.Simulations

First, we compare the miss ratio to the variable cache size. In the first example, with the fixed block size 32 and fixed associativity 4, a line will be formed with the miss rate as the Y-axis and the cache size as the X-axis for the graph.

Data collection and visualization of results for cache sizes of 16KB, 32KB, 64KB, and 128KB:

Table 1: Miss rate variation with 32 bytes fixed block size, 4-way set associativity, and varying cache size.

S.No.	Cache Size (in KB)	Block Size	Set Associativity	Total Access Count	Cache Hits	Hit Ratio	Miss Ratio
1	16	32	4	1500000	1488766	99.25	0.75
2	32	32	4	1500000	1493426	99.56	0.44
3	64	32	4	1500000	1495663	99.71	0.29
4	128	32	4	1500000	1497219	99.81	0.19

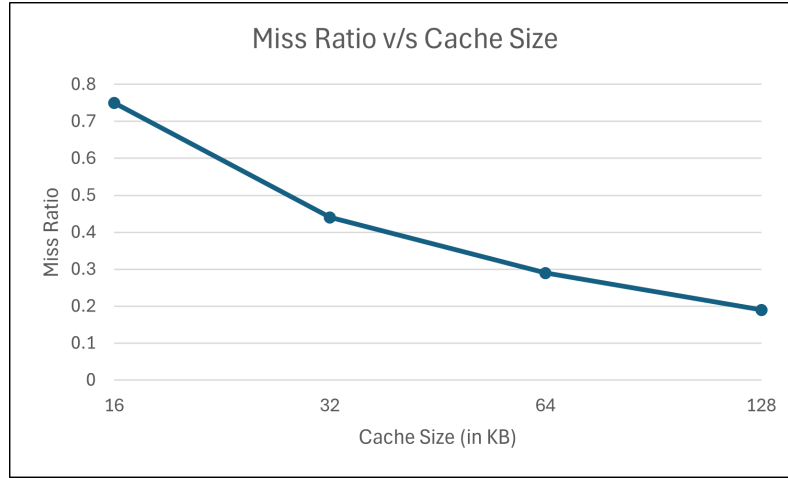


Figure 6: Miss Ratio Vs Cache Size Chart.

Furthermore, it is important to consider the changeable line size of the Miss Ratio. In the second example, given the fixed cache size of 8192 and fixed associativity of 4, a line will be formed with the block size as the X- axis and the miss rate as the Y- axis for the graph. The results are obtained and a graph based on the results for block sizes 16, 32, 64, and 128 is produced, as shown in Fig. 7.

Table 2: Miss rate variation with varying block size, 4-way set associativity, and 8KB cache size.

S.No.	Cache Size (in KB)	Block size	Set Associativity	Total Access Count	Cache Hits	Hit Ratio	Miss Ratio
1	8	16	4	1500000	1474170	98.28	1.72
2	8	32	4	1500000	1471126	98.08	1.92
3	8	64	4	1500000	1459207	97.28	2.78
4	8	128	4	1500000	1430504	95.37	4.63

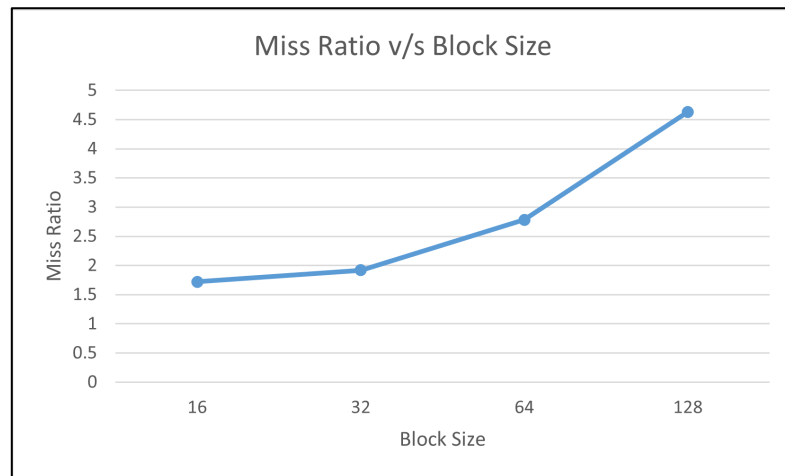


Figure 7: Miss Ratio Vs Block Size Chart.

Third, contrast the Miss Ratio with Variable Associativity. In the third example, given the fixed cache size of 8 KB and the fixed block size of 32 bytes, a line will be drawn with the associativity

acting as the X-axis and the miss rate acting as the Y-axis for the graphs. The results for associativity of 1, 2, 4, 8, 16, and completely associative are generated and displayed, as seen in Fig. 8.

Table 3: Miss rate variation with 32 bytes fixed block size, 8KB fixed cache size, and varying associativity.

S.No.	Cache Size (in KB)	Block Size	Set Associativity	Total Access Count	Cache Hits	Hit Ratio	Miss Ratio
1	8	32	1	1500000	1417382	94.49	5.51
2	8	32	2	1500000	1456971	97.13	2.87
3	8	32	4	1500000	1471126	98.08	1.92
4	8	32	8	1500000	1476941	98.48	1.52
5	8	32	16	1500000	1479609	98.64	1.36
6	8	32	32	1500000	1480144	98.68	1.32

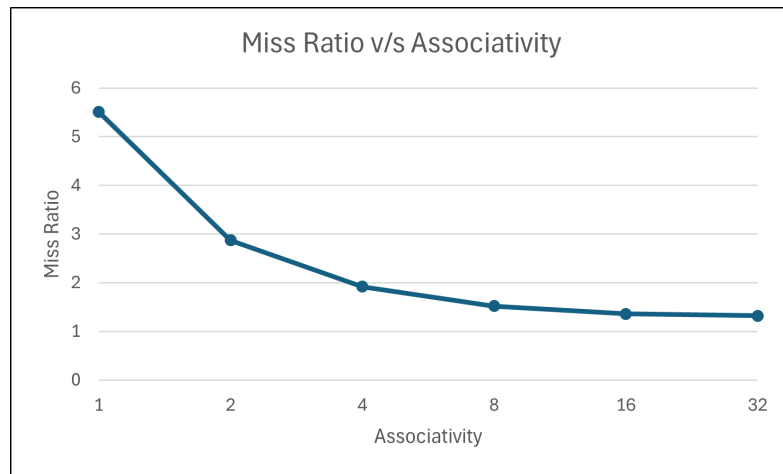


Figure 8: Miss Ratio Vs Associativity Chart.

3. Sector-Mapped Cache

3.1.Design Methodology

Sector-mapped cache design introduces an innovative approach to cache memory management by segmenting cache lines into smaller, independently addressable units called sectors. This technique optimizes storage utilization and reduces the overhead associated with caching large blocks, particularly for workloads where only portions of a block are frequently accessed. Understanding the sector-mapped cache architecture is essential to exploring its benefits and implications for system performance.

In a sector-mapped cache, each cache line is divided into multiple sectors, and each sector can hold a smaller subset of data from a memory block. This segmentation enables fine-grained data storage, allowing only the relevant parts of a memory block to occupy cache space. The key components of sector-mapped cache design include the **sector tag**, which identifies the memory address mapped to the sector, and a **valid bit**, which indicates whether the sector contains valid data.

Cache operations in a sector-mapped cache follow a unique flow:

- (1) When the CPU requests data, the cache controller extracts the memory block address and checks the

sector tags within the corresponding cache line.

- (2) If the requested data is found in a valid sector (sector hit), it is retrieved directly from the cache.
- (3) If the requested data is not found (sector miss), the cache fetches the specific sector from the main memory rather than the entire block, minimizing memory traffic.
- (4) Replacement policies, such as Least Recently Used (LRU) or variations adapted for sectors, manage sector replacement within cache lines.

To illustrate, consider a main memory with 128 words divided into 32 blocks, each containing four words. A sector-mapped cache with a line size of four sectors (each holding one word) and a total size of 16 words would have four cache lines. Each line accommodates a block of four sectors, with each sector independently mapped and validated. In this architecture, the block number determines the cache line, while the sector index specifies the target sector within the line.

3.2. Design Description

The sector-mapped cache architecture offers several advantages over traditional set-associative caches:

- (1) **Fine-Grained Storage:** By caching only the accessed sectors of a block, it reduces the overhead of storing unused data.
- (2) **Efficient Miss Handling:** Fetching individual sectors instead of entire blocks decreases memory bandwidth usage and improves system responsiveness.
- (3) **Enhanced Flexibility:** The sector-based approach adapts well to workloads with irregular access patterns, where only small portions of a block are frequently used.

Sector-mapped caches operate under the principle of **locality of reference**:

- (1) **Temporal Locality:** Recently accessed sectors are likely to be accessed again soon.
- (2) **Spatial Locality:** Sectors close to the currently accessed sector have a high probability of being accessed soon.

The cache controller plays a crucial role in managing sector mapping and replacement. Upon receiving a memory address, the controller decodes it into a block number and a sector offset. The block number determines the cache line, while the offset identifies the target sector. If the sector is valid and its tag matches the memory address, the controller serves the data directly to the CPU, achieving a sector hit.

In the event of a sector miss, the controller fetches the specific sector from main memory. This operation involves updating the sector tag and valid bit, ensuring efficient cache utilization. Replacement strategies such as sector-LRU manage the eviction of sectors when cache space is fully utilized.

3.3. Simulations

To compare the cache performance before and after the addition of sector mapping, we observe the changes in miss rate by comparing it while varying Cache Size, Block Size, Set Associativity, and Sector Size, one-by-one keeping other parameters constant and note the performance difference with the results we got in first part.

As first experiment, we perform performance evaluation of sector mapped cache by comparing miss rate variation with respect to variation in cache memory size. Table 5 presents the results from this experiment and Figure 9 illustrates how miss ratio varies with increment in cache size.

Table 4: Performance of Sector-mapped cache with variation in Cache Size keeping Block Size, Associativity, and Sector Size, constant.

S.No.	Cache Size (in KB)	Block Size	Set Associativity	Sector Size	Total Access Count	Cache Hits	Hit Ratio	Miss Ratio
1	8	32	4	4	1500000	1477927	98.53	1.47
2	16	32	4	4	1500000	1490650	99.38	0.62
3	32	32	4	4	1500000	1495824	99.72	0.28
4	64	32	4	4	1500000	1497872	99.86	0.14
5	128	32	4	4	1500000	1497880	99.86	0.14

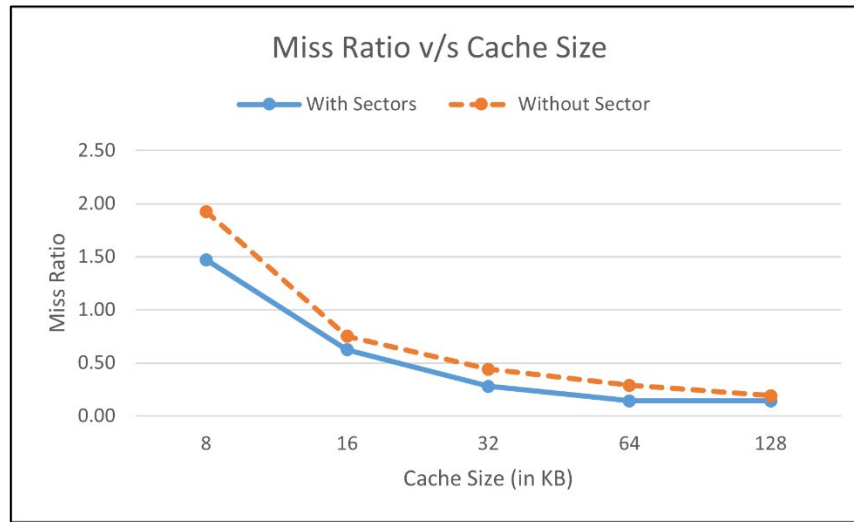


Figure 9: Miss Ratio variation with respect to Cache Size keeping other parameters constant.

Further, we perform performance evaluation of sector mapped cache by comparing miss rate variation with respect to variations in Block Size and Associativity as well, for which results are noted in Table 5 and 6, and Figure 10 and 11. These results demonstrate that miss ratio increases only with increasing Block Size and shows downward trend in all other cases.

Table 5: Cache Performance variation with variation in Block Size.

S.No.	Cache Size (in KB)	Block Size	Set Associativity	Sector Size	Total Access Count	Cache Hits	Hit Ratio	Miss Ratio
1	8	8	4	4	1500000	1476092	98.41	1.59
2	8	16	4	4	1500000	1479373	98.62	1.38
3	8	32	4	4	1500000	1477927	98.53	1.47
4	8	64	4	4	1500000	1497872	97.88	2.12
5	8	128	4	4	1500000	1497880	96.24	3.76

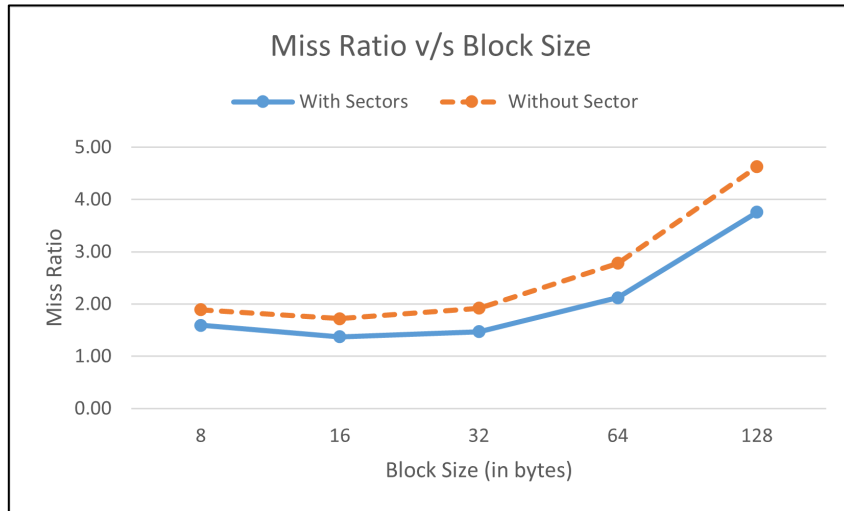


Figure 10: Cache Miss Ratio visualization with increment in Block Size, keeping all other parameters constant.

Table 6: Cache Performance comparison with variation in associativity (from 1 to 32).

S.No.	Cache Size (in KB)	Block Size	Set Associativity	Sector Size	Total Access Count	Cache Hits	Hit Ratio	Miss Ratio
1	8	32	1	4	1500000	1419474	94.63	5.37
2	8	32	2	4	1500000	1458418	97.23	2.77
3	8	32	4	4	1500000	1477927	98.53	1.47
4	8	32	8	4	1500000	1488655	99.24	0.76
5	8	32	16	4	1500000	1490635	99.38	0.62
6	8	32	32	4	1500000	1492758	99.52	0.48

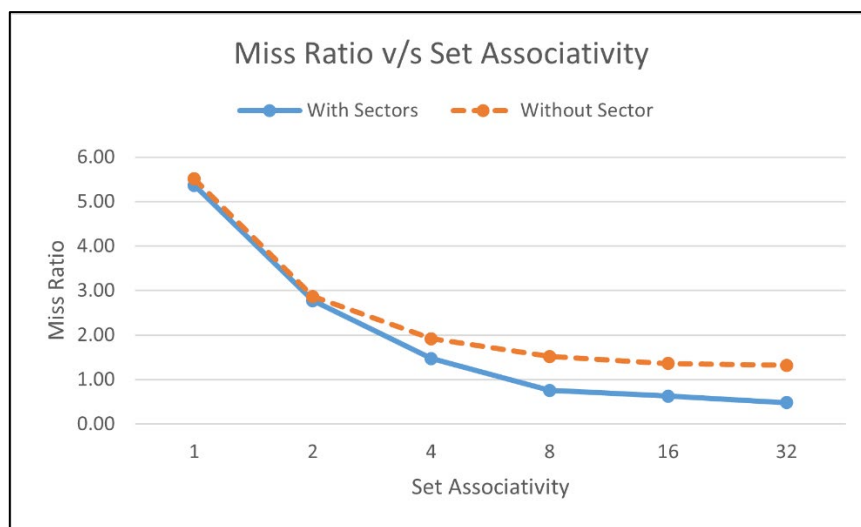


Figure 11: Miss Ratio v/s Associativity curve keeping other conditions constant.

We also performed experiments with variable sector size (2, 4, 8, 16, 32, and 64) to observe the performance of

sector-mapped cache memory with respect to change in sector size. Table 7 and Figure 12, illustrate comparison between Miss Ratio and Sector Size, keeping Block Size (32 bytes), Cache Size (8 KB), and Associativity (4-way) as constants. One may observe that with increment in Sector Size, Miss Ratio is showing downward trend which implies that increasing Sector Size, may improve cache memory performance by increasing the Cache Hits.

Table 7: Cache performance with variation in Sector size.

S. No.	Cache Size (in KB)	Block Size	Set Associativity	Sector Size	Total Access Count	Cache Hits	Hit Ratio	Miss Ratio
1	8	32	4	1	1500000	1471177	98.08	1.92
2	8	32	4	2	1500000	1475487	98.37	1.63
3	8	32	4	4	1500000	1477927	98.53	1.47
4	8	32	4	8	1500000	1480129	98.68	1.32
5	8	32	4	16	1500000	1481955	98.80	1.20

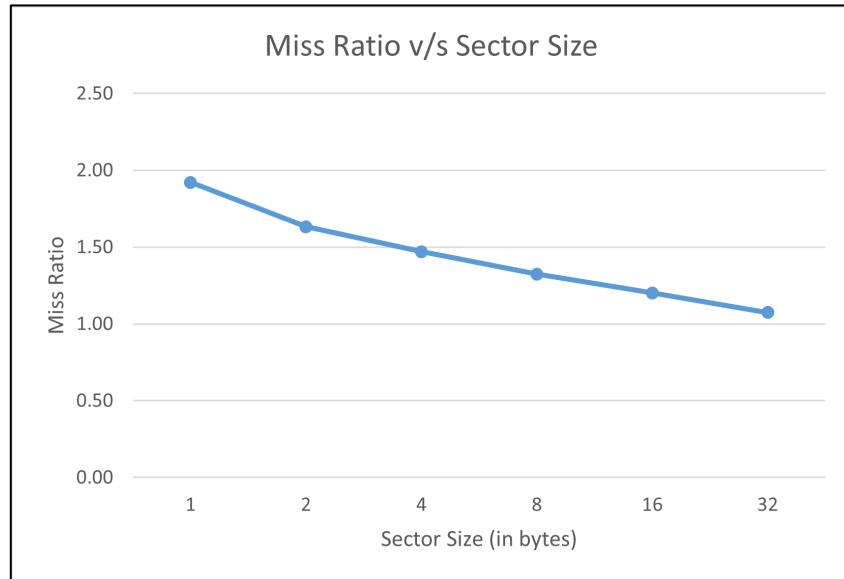


Figure 12: Miss Ratio variation with change in Sector size, keeping other parameters constant.

Best Parameters

By performing all above experiments we noted that the following parameters result in best performance or least miss ratio. Cache Size 64KB, 16-byte Block Size, and Fully Mapped set associativity, seem to be the optimal parameters for best performance. In this section we observe the changes in miss rate with respect to change in sector size while keeping other parameters to their optimal values. Table 8 and Figure 13 below shows the results and illustrates that miss rate further reduces when we use all optimal values simultaneously and achieve up to 99.89% Hit Rate.

Table 8: Cache performance with respect to variable sector size and optimal constant parameters.

S. No.	Cache Size (in KB)	Block Size	Set Associativity	Sector Size	Total Access Count	Cache Hits	Hit Ratio	Miss Ratio
1	64	16	32	1	1500000	1496306	99.75	0.25
2	64	16	32	2	1500000	1496858	99.79	0.21
3	64	16	32	4	1500000	1497304	99.82	0.18
4	64	16	32	8	1500000	1497805	99.85	0.15
5	64	16	32	16	1500000	1498290	99.89	0.11

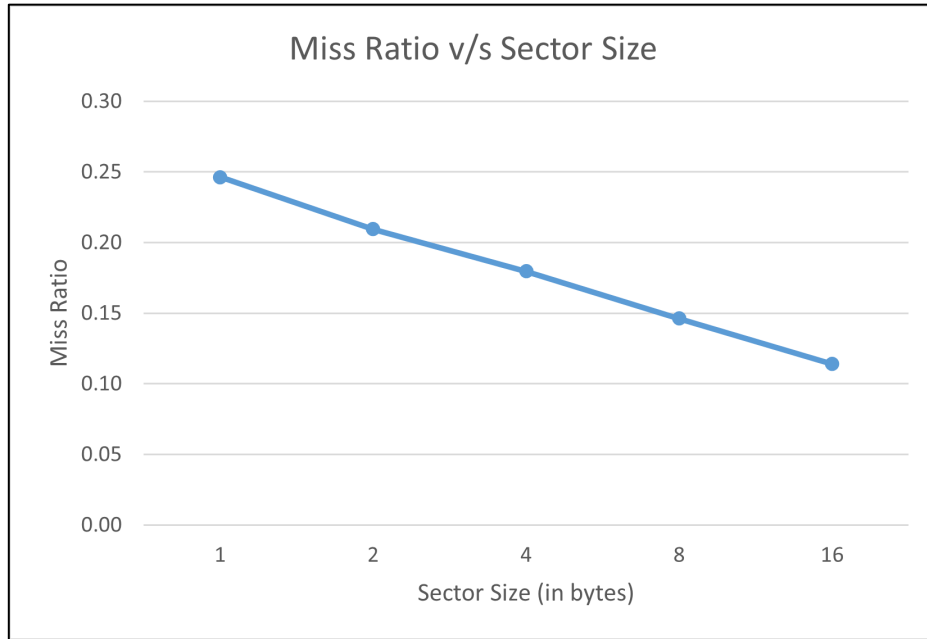


Figure 13: Cache performance with respect to sector size.

Conclusions

The research focused on developing a data cache simulator capable of evaluating m-way set-associative caches and sector-mapped caches across various configurations. Using Verilog and the Least Recently Used (LRU) algorithm, the simulator was implemented to handle complex cache architectures, requiring 2 GB of RAM for the given address space. The study explored three primary cache parameters—associativity, block (line) size, and cache size—and analyzed their influence on hit and miss rates for both m-way set-associative and sector-mapped caches.

M-Way Set-Associative Cache:

The results for m-way caches demonstrated that **cache size** significantly impacts performance. Increasing the cache size reduced capacity misses, thereby enhancing the hit ratio. However, larger cache sizes increased storage costs, emphasizing the trade-off between performance improvement and implementation expense.

- **Block size** played a critical role by affecting spatial locality. Larger blocks improved the hit ratio by fetching more adjacent data, but excessively large blocks reduced efficiency, as unused data displaced frequently accessed blocks. This finding highlights the importance of selecting an optimal block size for balancing locality and cache utilization.
- **Associativity** also influenced performance, with higher associativity reducing conflict misses and improving the hit ratio. Fully associative caches achieved the lowest miss rates but involved higher hardware complexity and cost. These results underscore the trade-offs in designing efficient, cost-effective cache systems.

Sector-Mapped Cache:

The sector-mapped cache experiments added new dimensions to the findings by introducing finer granularity through sector division. Key observations from the results include:

- **Cache size:** Increasing cache size from 8 KB to 64 KB with a fixed block size of 32 bytes and associativity of 4-way yielded a consistent improvement in the hit ratio, reaching as high as **99.86%**. This trend mirrors the behaviour of m-way caches but demonstrates sector-mapped caches' efficiency in reducing memory traffic due to fine-grained sector fetching.
- **Block size:** Smaller blocks generally resulted in higher hit ratios due to better utilization of spatial locality within sectors. However, extremely small blocks increased sector management overhead, indicating the need for balanced block size selection.
- **Associativity:** Higher associativity reduced conflict misses, as seen in configurations ranging from direct-mapped (94.63% hit ratio) to fully associative (99.52% hit ratio) caches. Sector-mapped caches further benefit from associativity by leveraging their finer granularity to minimize unnecessary block evictions.
- **Sector size:** Smaller sector sizes provided better cache utilization by reducing the storage of unused data. For example, reducing the sector size to 4 bytes increased the hit ratio to **98.53%**, while larger sector sizes, such as 32 bytes, achieved **99.75%**, balancing spatial locality with fewer cache misses. Furthermore, by using optimization we get **99.89%** hit rate.

General Insights:

The experimental results revealed how cache size, associativity, block size, and sector size interact to influence hit and miss rates. Larger cache sizes and higher associativity improved performance by reducing misses. Sector-mapped caches added efficiency by enabling fine-grained data handling, which was particularly beneficial for workloads with irregular access patterns. Optimal block and sector sizes further enhanced the hit ratio by balancing locality and utilization.

By integrating both m-way and sector-mapped cache designs, this research provides comprehensive insights into cache memory optimization. These findings equip system designers with a deeper understanding of the trade-offs and opportunities in selecting cache configurations for modern computing applications, contributing to the development of high-performance memory systems.

REFERENCES:

1. Lecture Notes EE275-Fall 2024: Dr. Harish Hiriyannaiah.
2. Computer Organization and Design: The Hardware/Software Interface, Fourth Edition by John L. Hennessy & David A. Patterson
3. Computer system architecture textbook by M Morris Mano ISBN: 81-317-0070-4
4. [Cache Memory in Computer Organization - GeeksforGeeks](#)

Supplementary Material

A. M-Way Associative Cache Simulation

A.1. Design File Verilog

```
`define cache_size (1024*8)
`define line_size 128
`define Associativity 4

`define Index_bit (`Associativity==0)? 0: $clog2(`cache_size/(`line_size*`Associativity))
`define Offset_bit $clog2(`line_size)
`define Tag_bit 31-(`Offset_bit+`Index_bit)

module test(adder 41,clk 41,rst 41,misses 41,hits 41);
    input clk_41,rst_41;
    input [30:0] adder 41;
    output [30:0] misses 41,hits 41;

    reg [30:0] Num Blocks 41,Num Sets 41,misses 41,hits 41,cache block 41,cache set 41,set index 41,Curr Block 41,Curr Count 41;
    reg data_present 41;

    integer i,j;
    integer k;

    parameter CS = `cache_size;
    parameter LS = `line_size;
    parameter Assoc = (`Associativity==0)? (CS/LS):`Associativity;
    parameter Tbit = `Tag_bit;
    parameter Ob = `Offset_bit;
    parameter Ib = `Index_bit;

    reg [(LS*8)-1:0] cache [0:(CS/LS) - 1];
    reg [Tbit-1:0] tag_array [0:(CS/LS) - 1]; // for all tags in cache
    reg valid_array [0:(CS/LS) - 1]; //0 - there is no data 1 - there is data

    reg [Tbit-1:0] tag; // for current tag

    reg [Assoc-1:0] counter [0:(CS/LS) - 1];
```



```

initial begin
    hits_41 = 0;
    misses_41 = 0;
    Num_Blocks_41 = CS/LS;
    Num_Sets_41 = Num_Blocks_41/Assoc;

    for (k = 0; k < Num_Blocks_41 ; k = k + 1)
        begin
            valid_array[k] = 0;
            tag_array[k] = 0;
        end
    for (j = 0; j < Num_Sets_41 ; j = j + 1)
        for (k = 0; k < Assoc ; k = k + 1)
            counter[(j*Assoc)+k] = k;

end

always@(posedge clk_41) begin

    cache_block_41 = (adders_41/LS)% Num_Blocks_41;
    cache_set_41 = ((adders_41/LS)% (Num_Blocks_41/Assoc));
    set_index_41 = cache_set_41*Assoc; //pointing at first 41 block of current set
    data_present_41 = 0;

    tag = adders_41[30:(Ob+Ib)];

    for(i=0; i<Assoc; i=i+1) begin
        if ((valid_array [set_index_41+i] == 1) && (tag == tag_array[set_index_41+i])) begin
            hits_41 = hits_41+1;
            data_present_41 = 1;
            Curr_Block_41 = set_index_41+i;
            Curr_Count_41 = counter[Curr_Block_41];
            // $display("hits_41=%d",hits_41);
        end
    end

    if (data_present_41 == 0) begin
        misses_41 = misses_41+1;
        // $display("misses_41=%d",misses_41);

        for(i=0; i<Assoc; i=i+1) begin
            if (counter[set_index_41+i]==0)begin
                tag_array[set_index_41+i] = tag;
                valid_array [set_index_41+i] = 1;
                Curr_Block_41 = set_index_41+i;
                Curr_Count_41 = 0;
            end
        end
    end
end

```

```

    end
  end
end

for(i=0; i<Assoc; i=i+1) begin // counter

  if (counter[set_index_41+i]>Curr_Count_41) begin
    counter[set_index_41+i] = counter[set_index_41+i] - 1;
  end

  counter[Curr_Block_41] = Assoc - 1 ;

end

end

endmodule

```

A.2. Test-bench Verilog

```

module tb();
  reg clk_41, rst_41;
  reg [30:0] adder_41;
  reg [8:0] LS_41;
  reg [16:0] CS_41;
  reg [5:0] Assoc_41;

  wire [30:0] misses_41,hits_41;
  real hit_ratio_41,per = 100.00;
  real m,h;

  `define Length 1500000
  integer p,r,c;
  integer file,stat,out,i,j;
  reg signed [30:0] face[0:`Length-1];
  reg [30:0] actual[0:`Length-1];

  test t1(adder_41,clk_41, rst_41,misses_41,hits_41);

  initial begin
    file=$fopen("addr_trace.txt","r");
    i=0;
    while (! $feof(file))
      begin

```

```

        stat=$fscanf(file,"%d\n",face[i]);
        i=i+1;
    end
    $fclose(file);

    for(i=0;i<`Length`;i=i+1)
        begin
            if (i==0) actual[0] = face[0];
            else actual[i]=actual[i-1]+face[i];

        end
    end

    always #5 clk_41 = ~clk_41;

    initial begin

        clk_41 = 0;

        for (j = 0 ; j<`Length` ; j=j+1) begin

            @(posedge clk_41) adder_41 = actual[j];
            //$display("Address=%d",adder_41);
        end

        @(posedge clk_41)
        m=misses_41;
        h=hits_41;
        hit_ratio_41 = (h/(m+h))*per;
        $display("Hit Ratio=%7.2f Percentage, Cache Hits = %d, Total Simulation Addresses =
%d",hit_ratio_41,hits_41,(misses_41 + hits_41));
        $finish;
    end

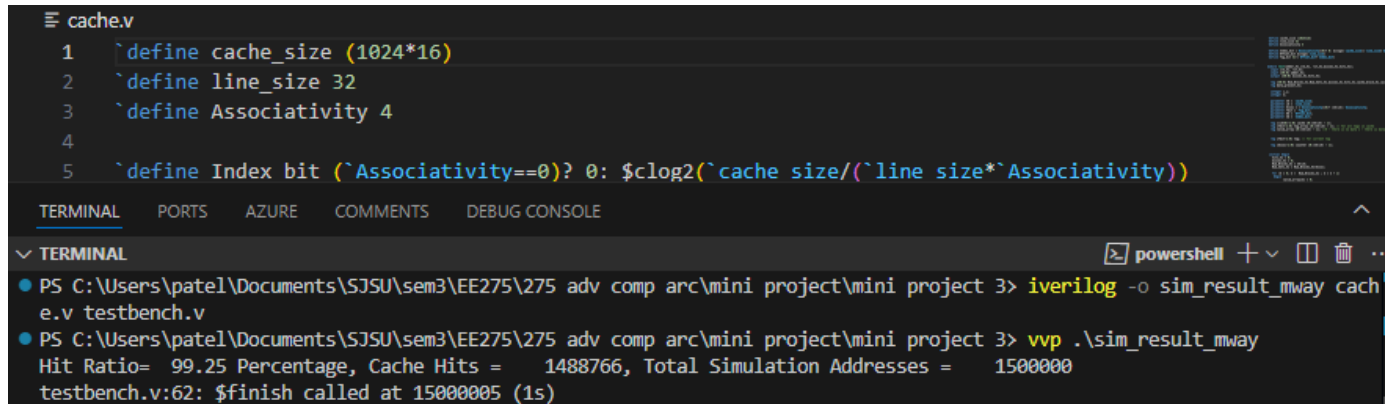
endmodule

```

A.3. Simulation Results

First case: Varying Cache size (Fixed block size= 32 Bytes, Fixed Associativity=4)

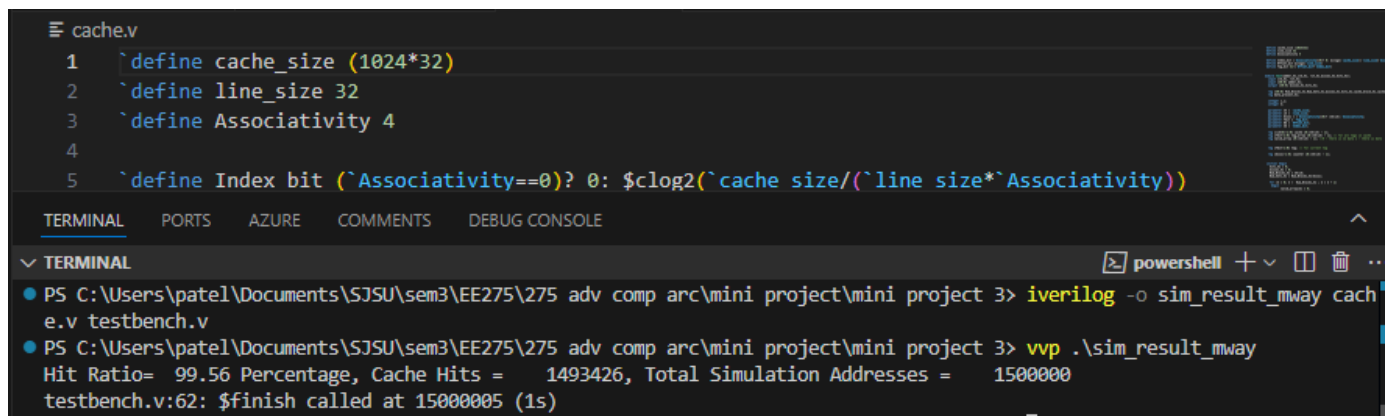
Cache size = 16KB:



```
cache.v
1  `define cache_size (1024*16)
2  `define line_size 32
3  `define Associativity 4
4
5  `define Index bit (`Associativity==0)? 0: $clog2(`cache_size/(`line_size*`Associativity))

TERMINAL
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_mway cach
e.v testbench.v
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_mway
Hit Ratio= 99.25 Percentage, Cache Hits = 1488766, Total Simulation Addresses = 1500000
testbench.v:62: $finish called at 15000005 (1s)
```

Cache Size = 32KB:



```
cache.v
1  `define cache_size (1024*32)
2  `define line_size 32
3  `define Associativity 4
4
5  `define Index bit (`Associativity==0)? 0: $clog2(`cache_size/(`line_size*`Associativity))

TERMINAL
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_mway cach
e.v testbench.v
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_mway
Hit Ratio= 99.56 Percentage, Cache Hits = 1493426, Total Simulation Addresses = 1500000
testbench.v:62: $finish called at 15000005 (1s)
```

Cache size = 64KB:



```
cache.v
1  `define cache_size (1024*64)
2  `define line_size 32
3  `define Associativity 4
4
5  `define Index bit (`Associativity==0)? 0: $clog2(`cache_size/(`line_size*`Associativity))

TERMINAL
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_mway cach
e.v testbench.v
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_mway
Hit Ratio= 99.71 Percentage, Cache Hits = 1495663, Total Simulation Addresses = 1500000
testbench.v:62: $finish called at 15000005 (1s)
```

Cache size = 128KB:

```
1  `define cache_size (1024*128)
2  `define line_size 32
3  `define Associativity 4
4
5  `define Index bit (`Associativity==0)? 0: $clog2(`cache_size/(`line_size*`Associativity))
```

TERMINAL PORTS AZURE COMMENTS DEBUG CONSOLE

▼ TERMINAL

```
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_mway cach
e.v testbench.v
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_mway
Hit Ratio= 99.81 Percentage, Cache Hits = 1497219, Total Simulation Addresses = 1500000
testbench.v:62: $finish called at 15000005 (1s)
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3>
```

Second case: Varying Block size (fixed cache size = 8192 KB, fixed associativity =4)

Block size = 16:

```
cache.v
1 `define cache_size (1024*8)
2 `define line_size 16
3 `define Associativity 4
4
5 `define Index bit (`Associativity==0)? 0: $clog2(`cache_size/(`line_size*`Associativity))
```

TERMINAL PORTS AZURE COMMENTS DEBUG CONSOLE

▼ TERMINAL powershell + -

- PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_mway cache.v testbench.v
- PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_mway

Hit Ratio= 98.28 Percentage, Cache Hits = 1474170, Total Simulation Addresses = 1500000

testbench.v:62: \$finish called at 15000005 (1s)

Block size = 32:

```
cache.v
1  `define cache_size (1024*8)
2  `define line_size 32
3  `define Associativity 4
4
5  `define Index bit (`Associativity==0)? 0: $clog2(`cache_size/(`line_size*`Associativity))
```

TERMINAL PORTS AZURE COMMENTS DEBUG CONSOLE

▼ TERMINAL

- PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_mway cache.v testbench.v
- PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_mway

Hit Ratio= 98.08 Percentage, Cache Hits = 1471126, Total Simulation Addresses = 1500000

testbench.v:62: \$finish called at 15000005 (1s)

Block size = 64:

```
cache.v
1  `define cache_size (1024*8)
2  `define line_size 64
3  `define Associativity 4
4
5  `define Index bit (`Associativity==0)? 0: $clog2(`cache_size/(`line_size*`Associativity))

TERMINAL  PORTS  AZURE  COMMENTS  DEBUG CONSOLE
▼ TERMINAL
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_mway cache.v testbench.v
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_mway
Hit Ratio= 97.28 Percentage, Cache Hits = 1459207, Total Simulation Addresses = 1500000
testbench.v:62: $finish called at 15000005 (1s)
```

Block size = 128:

```
cache.v
1  `define cache_size (1024*16)
2  `define line_size 128
3  `define Associativity 4
4
5  `define Index bit (`Associativity==0)? 0: $clog2(`cache_size/(`line_size*`Associativity))

TERMINAL  PORTS  AZURE  COMMENTS  DEBUG CONSOLE
▼ TERMINAL
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_mway cache.v testbench.v
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_mway
Hit Ratio= 99.02 Percentage, Cache Hits = 1485315, Total Simulation Addresses = 1500000
testbench.v:62: $finish called at 15000005 (1s)
```

Third case: Varying Associativity (fixed cache size = 8192 KB, block size = 32KB)

1-way Set Associativity (Direct mapped cache):

```
cache.v
1  `define cache_size (1024*8)
2  `define line_size 32
3  `define Associativity 1
4
5  `define Index bit (`Associativity==0)? 0: $clog2(`cache_size/(`line_size*`Associativity))

TERMINAL  PORTS  AZURE  COMMENTS  DEBUG CONSOLE
▼ TERMINAL
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_mway cache.v testbench.v
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_mway
Hit Ratio= 94.49 Percentage, Cache Hits = 1417382, Total Simulation Addresses = 1500000
testbench.v:62: $finish called at 15000005 (1s)
```

2-way set associativity:

```
cache.v
1 `define cache_size (1024*8)
2 `define line_size 32
3 `define Associativity 2
4
5 `define Index bit (`Associativity==0)? 0: $clog2(`cache_size/(`line_size*`Associativity))

TERMINAL PORTS AZURE COMMENTS DEBUG CONSOLE
▼ TERMINAL
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_mway cache.v testbench.v
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_mway
Hit Ratio= 97.13 Percentage, Cache Hits = 1456971, Total Simulation Addresses = 1500000
testbench.v:62: $finish called at 15000005 (1s)
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> |
```

4-way set associativity:

```
cache.v
1 `define cache_size (1024*8)
2 `define line_size 32
3 `define Associativity 4
4
5 `define Index bit (`Associativity==0)? 0: $clog2(`cache_size/(`line_size*`Associativity))

TERMINAL PORTS AZURE COMMENTS DEBUG CONSOLE
▼ TERMINAL
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_mway cache.v testbench.v
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_mway
Hit Ratio= 98.08 Percentage, Cache Hits = 1471126, Total Simulation Addresses = 1500000
testbench.v:62: $finish called at 15000005 (1s)
```

8-way set associativity:

```
cache.v
1 `define cache_size (1024*8)
2 `define line_size 32
3 `define Associativity 8
4
5 `define Index bit (`Associativity==0)? 0: $clog2(`cache_size/(`line_size*`Associativity))

TERMINAL PORTS AZURE COMMENTS DEBUG CONSOLE
▼ TERMINAL
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_mway cache.v testbench.v
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_mway
Hit Ratio= 98.46 Percentage, Cache Hits = 1476941, Total Simulation Addresses = 1500000
testbench.v:62: $finish called at 15000005 (1s)
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> |
```

16-way set associativity:

```
cache.v
1 `define cache_size (1024*8)
2 `define line_size 32
3 `define Associativity 16
4
5 `define Index bit (`Associativity==0)? 0: $clog2(`cache_size/(`line_size*`Associativity))

TERMINAL PORTS AZURE COMMENTS DEBUG CONSOLE
▼ TERMINAL
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_mway
Hit Ratio= 98.64 Percentage, Cache Hits = 1479609, Total Simulation Addresses = 1500000
testbench.v:62: $finish called at 15000005 (1s)
```

32-way set associativity (Fully Associative):

```
cache.v
1  `define cache_size (1024*8)
2  `define line_size 32
3  `define Associativity 32
4
5  `define Index_bit (`Associativity==0)? 0: $clog2(`cache_size/(`line_size*`Associativity))

TERMINAL  PORTS  AZURE  COMMENTS  DEBUG CONSOLE
▼ TERMINAL
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_mway cache.v testbench.v
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_mway
Hit Ratio= 98.68 Percentage, Cache Hits = 1480144, Total Simulation Addresses = 1500000
testbench.v:62: $finish called at 15000005 (1s)
```

B. Sector Mapped Cache Simulation

B.1. Design File Verilog

```
`define cache_size (1024*8)
`define line_size 32
`define sector_size 64 // new definition for sector size
`define Associativity 4

`define Index_bit (`Associativity==0)? 0: $clog2(`cache_size/(`line_size*`Associativity))
`define Offset_bit $clog2(`line_size)
`define Sector_Offset_Bit $clog2(`sector_size)
`define Tag_bit 31-(`Offset_bit+`Index_bit+`Sector_Offset_Bit)

module test(adder_41,clk_41, rst_41,misses_41,hits_41);
    input clk_41, rst_41;
    input [30:0] adder_41;
    output [30:0] misses_41,hits_41;

    reg [30:0] Num_Blocks_41,Num_Sets_41,misses_41,hits_41,cache_block_41,cache_set_41,set_index_41,Curr
    _Block_41,Curr_Count_41;
    reg data_present_41;

    integer i,j;
    integer k;

    parameter CS = `cache_size;
    parameter LS = `line_size;
    parameter SS = `sector_size; // sector size
    parameter Assoc = (`Associativity==0)? (CS/LS):`Associativity;
    parameter Tbit = `Tag_bit;
    parameter Ob = `Offset_bit;
```



```

parameter Ib = `Index_bit;
parameter SOB = `Sector_Offset_Bit; // sector offset bit

reg [(LS*8)-1:0] cache [0:(CS/LS) - 1];
reg [Tbit-1:0] tag_array [0:(CS/LS) - 1]; // for all tags in cache
reg valid_array [0:(CS/LS) - 1]; //0 - there is no data 1 - there is data

// sector information
reg [SOB-1:0] sector_offset [0:(CS/LS) - 1];

reg [Tbit-1:0] tag; // for current tag

reg [Assoc-1:0] counter [0:(CS/LS) - 1];

initial begin
    hits_41 = 0;
    misses_41 = 0;
    Num_Blocks_41 = CS/LS;
    Num_Sets_41 = Num_Blocks_41/Assoc;

    for (k = 0; k < Num_Blocks_41 ; k = k + 1)
        begin
            valid_array[k] = 0;
            tag_array[k] = 0;
            sector_offset[k] = 0; // initialize sector offset
        end
    for (j = 0; j < Num_Sets_41 ; j = j + 1)
        for (k = 0; k < Assoc ; k = k + 1)
            counter[(j*Assoc)+k] = k;

end

always@(posedge clk_41)begin

    cache_block_41 = (adder_41/LS)% Num_Blocks_41;
    cache_set_41 = ((adder_41/LS)% (Num_Blocks_41/Assoc));
    set_index_41 = cache_set_41*Assoc; //pointing at first block of current set
    data_present_41 = 0;

    tag = adder_41[30:(0b+Ib+SOB)];

    for(i=0; i<Assoc; i=i+1) begin

```

```

        if ((valid_array [set_index_41+i] == 1) && (tag == tag_array[set_index_41+i]))
begin
    hits_41 = hits_41+1;
    data_present_41 = 1;
    Curr_Block_41 = set_index_41+i;
    Curr_Count_41 = counter[Curr_Block_41];
    // $display("hits_41=%d",hits_41);
    end
end

if (data_present_41 == 0) begin
    misses_41 = misses_41+1;

    for(i=0; i<Assoc; i=i+1) begin
        if (counter[set_index_41+i]==0)begin
            tag_array[set_index_41+i] = tag;
            valid_array [set_index_41+i] = 1;
            sector_offset[set_index_41+i] = adder_41[(0b+Ib+S0B-1):(0b+Ib)]; // store
sector offset
            Curr_Block_41 = set_index_41+i;
            Curr_Count_41 = 0;
        end
    end
end

for(i=0; i<Assoc; i=i+1) begin // counter

    if (counter[set_index_41+i]>Curr_Count_41)begin
        counter[set_index_41+i] = counter[set_index_41+i] - 1;
    end

    counter[Curr_Block_41] = Assoc - 1 ;

end

end

endmodule

```

B.2. Test-bench Verilog

```
module tb();
    reg clk_41, rst_41;
    reg [30:0] adder_41;
    reg [8:0] LS_41;
    reg [16:0] CS_41;
    reg [5:0] Assoc_41;

    wire [30:0] misses_41,hits_41;
    real hit_ratio_41,per = 100.00;
    real m,h;

    `define Length 1500000
    integer p,r,c;
    integer file,stat,out,i,j;
    reg signed [30:0] face[0:`Length-1];
    reg [30:0] actual[0:`Length-1];

    test t1(adder_41,clk_41, rst_41,misses_41,hits_41);

    initial begin
        file=$fopen("addr_trace.txt","r");
        i=0;
        while (! $feof(file))
            begin
                stat=$fscanf(file,"%d\n",face[i]);
                i=i+1;
            end
        $fclose(file);

        for(i=0;i<`Length;i=i+1)
            begin
                if (i==0) actual[0] = face[0];
                else actual[i]=actual[i-1]+face[i];
            end
    end

    always #5 clk_41 = ~clk_41;
```

```

initial begin

    clk_41 = 0;
    rst_41 = 1;
    @(posedge clk_41) rst_41 = 0;

    for (j = 0 ; j<`Length ; j=j+1) begin

        @(posedge clk_41) adder_41 = actual[j];
        //$display("Address=%d",adder_41);
    end

    @(posedge clk_41)
    m=misses_41;
    h=hits_41;
    hit_ratio_41 = (h/(m+h))*per;
    $display("Hit Ratio=%7.2f Percentage, Cache Hits = %d, Total Simulation Addresses = %d",hit_ratio_41,hits_41,(misses_41 + hits_41));
    $finish;
end

endmodule

```

B.3. Simulation Results

First case: Varying Cache size (Fixed block size= 32 Bytes, Fixed Associativity=4, Fixed sector size=4)

Cache size = 8KB:

The screenshot shows a Verilog source code editor with the following code for `cache_sector.v`:

```

1  `define cache_size (1024*8)
2  `define line_size 32
3  `define sector_size 4 // new definition for sector size
4  `define Associativity 4
5

```

Below the code editor is a terminal window showing the command prompt and the simulation results:

```

PS C:\Users\patel\Documents\JSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector .\cache_sector.v .\testbench_sector.v
PSvvp .\sim_result_sector                                ject\mini project 3>
Hit Ratio= 98.53 Percentage, Cache Hits = 1477927, Total Simulation Addresses = 1500002
.\testbench_sector.v:63: $finish called at 15000015 (1s)

```

Cache size = 16KB:

```
cache_sector.v
1  `define cache_size (1024*16)
2  `define line_size 32
3  `define sector_size 4 // new definition for sector size
4  `define Associativity 4
5

TERMINAL PORTS AZURE COMMENTS DEBUG CONSOLE
▼ TERMINAL powershell + - [ ] [ ]
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector
  \cache_sector.v .\testbench_sector.v
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_sector
  Hit Ratio= 99.38 Percentage, Cache Hits = 1490650, Total Simulation Addresses = 1500002
  .\testbench_sector.v:63: $finish called at 15000015 (1s)
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> |
```

Cache size = 32KB:

```
cache_sector.v
1  `define cache_size (1024*32)
2  `define line_size 32
3  `define sector_size 4 // new definition for sector size
4  `define Associativity 4
5

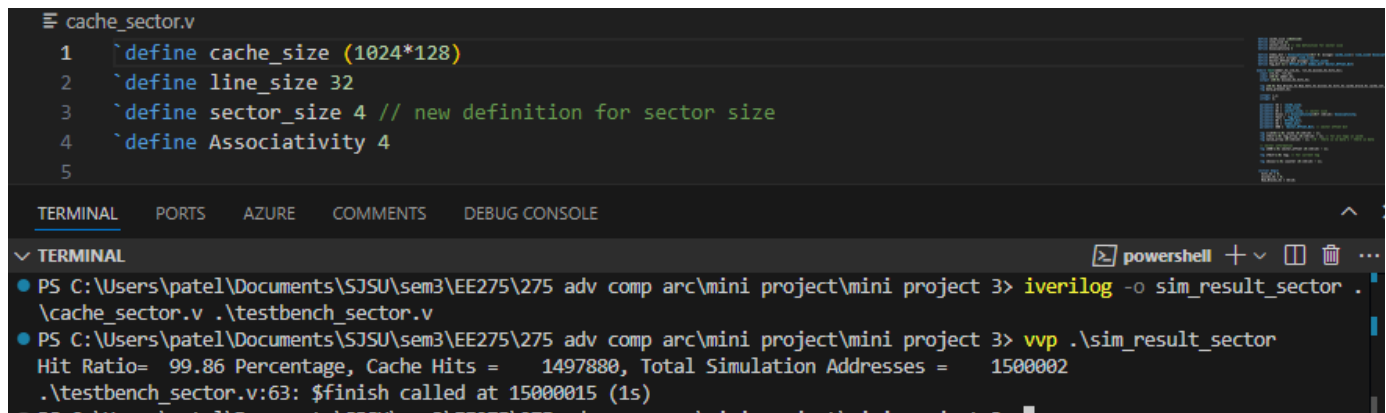
TERMINAL PORTS AZURE COMMENTS DEBUG CONSOLE
▼ TERMINAL powershell + - [ ] [ ]
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector
  \cache_sector.v .\testbench_sector.v
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_sector
  Hit Ratio= 99.86 Percentage, Cache Hits = 1497872, Total Simulation Addresses = 1500002
  .\testbench_sector.v:63: $finish called at 15000015 (1s)
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> |
```

Cache size = 64KB:

```
cache_sector.v
1  `define cache_size (1024*32)
2  `define line_size 32
3  `define sector_size 4 // new definition for sector size
4  `define Associativity 4
5

TERMINAL PORTS AZURE COMMENTS DEBUG CONSOLE
▼ TERMINAL powershell + - [ ] [ ]
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector
  \cache_sector.v .\testbench_sector.v
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_sector
  Hit Ratio= 99.72 Percentage, Cache Hits = 1495824, Total Simulation Addresses = 1500002
  .\testbench_sector.v:63: $finish called at 15000015 (1s)
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> |
```

Cache size = 128KB:

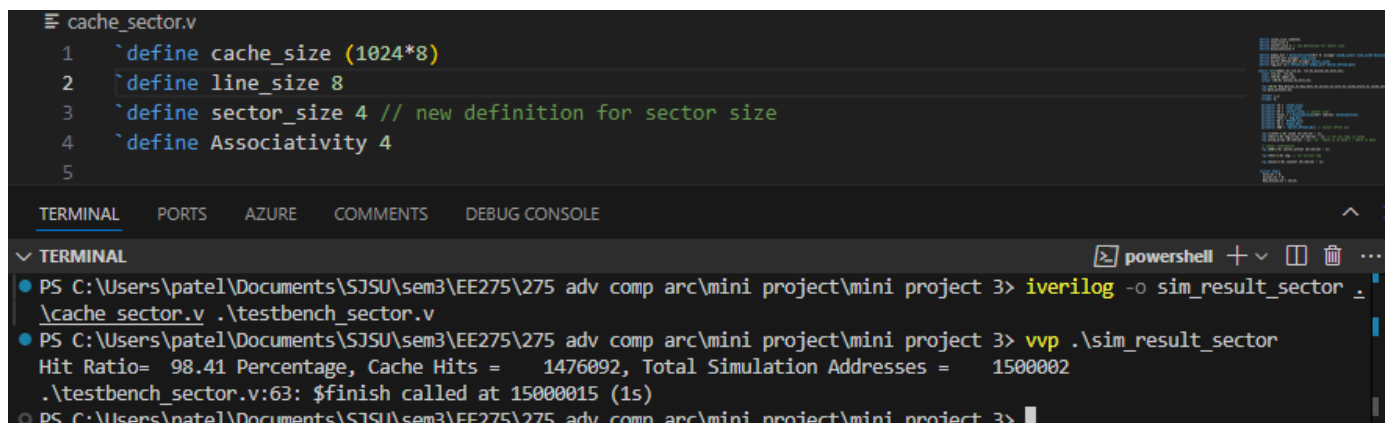


```
cache_sector.v
1 `define cache_size (1024*128)
2 `define line_size 32
3 `define sector_size 4 // new definition for sector size
4 `define Associativity 4
5

TERMINAL
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector .\cache_sector.v .\testbench_sector.v
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_sector
Hit Ratio= 99.86 Percentage, Cache Hits = 1497880, Total Simulation Addresses = 1500002
.\testbench_sector.v:63: $finish called at 15000015 (1s)
```

Second case: Varying Block Size (Fixed cache size= 8KB, Fixed Associativity=4, Fixed sector size=4)

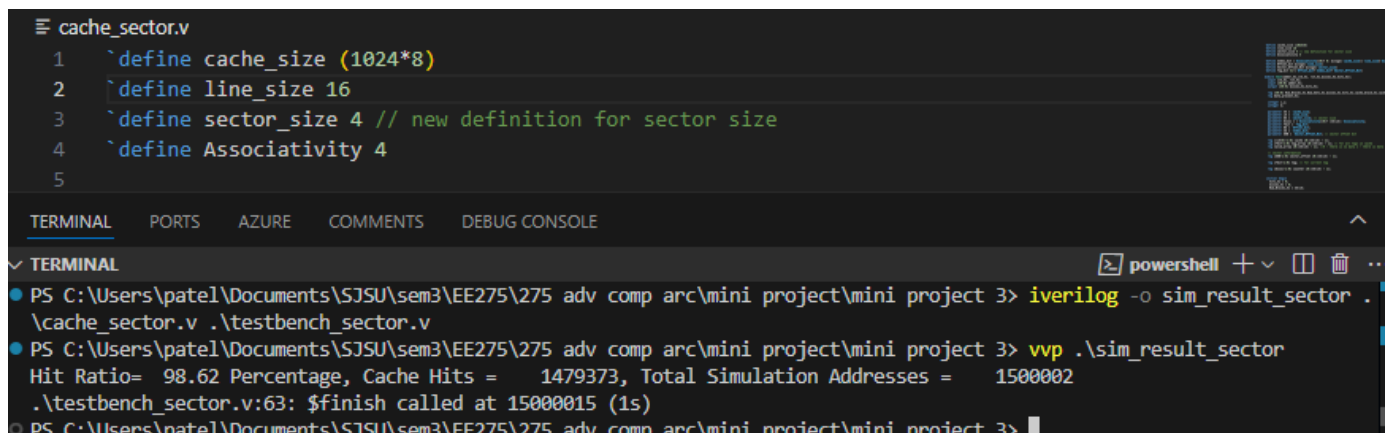
Block size = 8 bytes:



```
cache_sector.v
1 `define cache_size (1024*8)
2 `define line_size 8
3 `define sector_size 4 // new definition for sector size
4 `define Associativity 4
5

TERMINAL
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector .\cache_sector.v .\testbench_sector.v
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_sector
Hit Ratio= 98.41 Percentage, Cache Hits = 1476092, Total Simulation Addresses = 1500002
.\testbench_sector.v:63: $finish called at 15000015 (1s)
```

Block size = 16 bytes:



```
cache_sector.v
1 `define cache_size (1024*8)
2 `define line_size 16
3 `define sector_size 4 // new definition for sector size
4 `define Associativity 4
5

TERMINAL
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector .\cache_sector.v .\testbench_sector.v
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_sector
Hit Ratio= 98.62 Percentage, Cache Hits = 1479373, Total Simulation Addresses = 1500002
.\testbench_sector.v:63: $finish called at 15000015 (1s)
```

Block size = 32 bytes:

```
cache_sector.v
1  `define cache_size (1024*8)
2  `define line_size 32
3  `define sector_size 4 // new definition for sector size
4  `define Associativity 4
5

TERMINAL PORTS AZURE COMMENTS DEBUG CONSOLE
▼ TERMINAL powershell + - [ ] [X]
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector .\cache_sector.v .\testbench_sector.v
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_sector
Hit Ratio= 98.53 Percentage, Cache Hits = 1477927, Total Simulation Addresses = 1500002
.\testbench_sector.v:63: $finish called at 15000015 (1s)
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> |
```

Block size = 64 bytes:

```
cache_sector.v
1  `define cache_size (1024*8)
2  `define line_size 64
3  `define sector_size 4 // new definition for sector size
4  `define Associativity 4
5

TERMINAL PORTS AZURE COMMENTS DEBUG CONSOLE
▼ TERMINAL powershell + - [ ] [X]
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector .\cache_sector.v .\testbench_sector.v
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_sector
Hit Ratio= 97.88 Percentage, Cache Hits = 1468141, Total Simulation Addresses = 1500002
.\testbench_sector.v:63: $finish called at 15000015 (1s)
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> |
```

Block size = 128 bytes:

```
cache_sector.v
1  `define cache_size (1024*8)
2  `define line_size 128
3  `define sector_size 4 // new definition for sector size
4  `define Associativity 4
5

TERMINAL PORTS AZURE COMMENTS DEBUG CONSOLE
▼ TERMINAL powershell + - [ ] [X]
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector .\cache_sector.v .\testbench_sector.v
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_sector
Hit Ratio= 96.24 Percentage, Cache Hits = 1443644, Total Simulation Addresses = 1500002
.\testbench_sector.v:63: $finish called at 15000015 (1s)
```

Third case: Varying Associativity (Fixed cache size= 8KB, Fixed Block Size=32 bytes, Fixed sector size=4)

Associativity=1 (Direct Mapped):

```
cache_sector.v
1  `define cache_size (1024*8)
2  `define line_size 32
3  `define sector_size 4 // new definition for sector size
4  `define Associativity 1
5

TERMINAL
PORTS
AZURE
COMMENTS
DEBUG CONSOLE
✓ TERMINAL
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector .\cache_sector.v .\testbench_sector.v
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_sector
Hit Ratio= 94.63 Percentage, Cache Hits = 1419474, Total Simulation Addresses = 1500002
.\testbench_sector.v:63: $finish called at 15000015 (1s)
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3>
```

Associativity = 2:

```
cache_sector.v
1  `define cache_size (1024*8)
2  `define line_size 32
3  `define sector_size 4 // new definition for sector size
4  `define Associativity 2
5

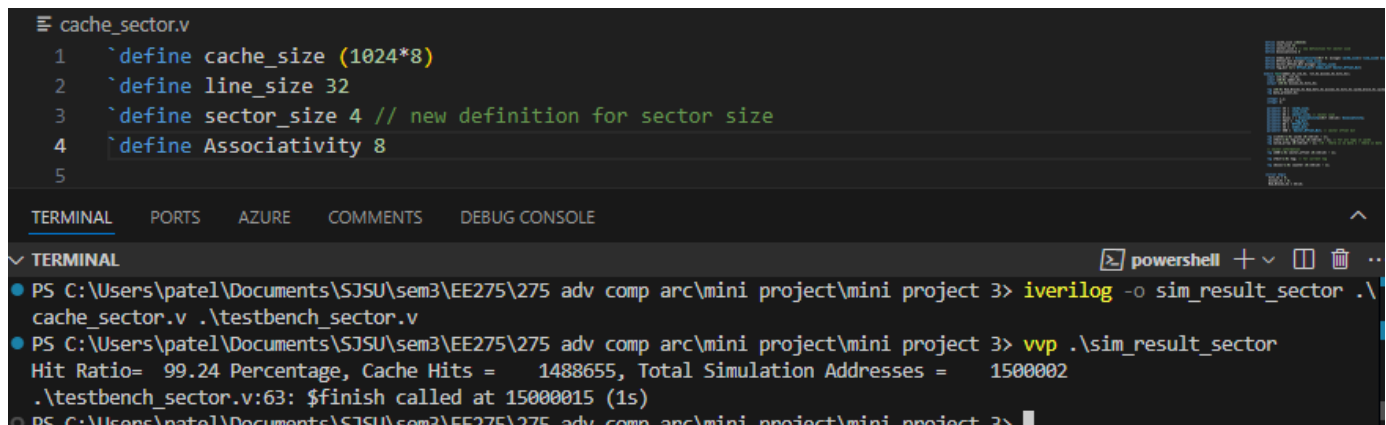
TERMINAL
PORTS
AZURE
COMMENTS
DEBUG CONSOLE
✓ TERMINAL
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector .\cache_sector.v .\testbench_sector.v
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_sector
Hit Ratio= 97.23 Percentage, Cache Hits = 1458418, Total Simulation Addresses = 1500002
.\testbench_sector.v:63: $finish called at 15000015 (1s)
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3>
```

Associativity = 4:

```
cache_sector.v
1  `define cache_size (1024*8)
2  `define line_size 32
3  `define sector_size 4 // new definition for sector size
4  `define Associativity 4
5

TERMINAL
PORTS
AZURE
COMMENTS
DEBUG CONSOLE
✓ TERMINAL
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector .\cache_sector.v .\testbench_sector.v
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_sector
Hit Ratio= 98.53 Percentage, Cache Hits = 1477927, Total Simulation Addresses = 1500002
.\testbench_sector.v:63: $finish called at 15000015 (1s)
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3>
```

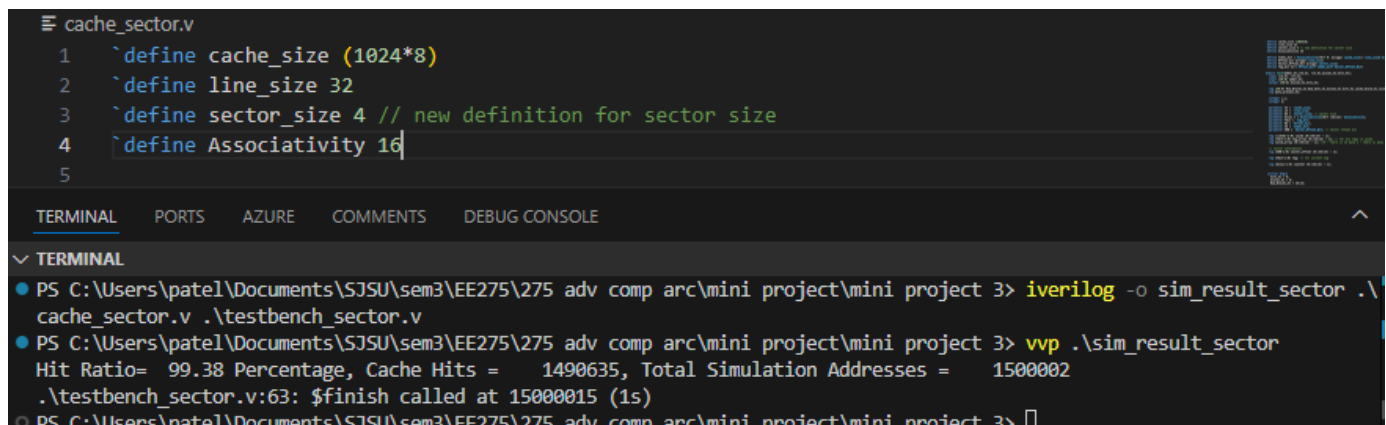

Associativity = 8:



```
cache_sector.v
1  `define cache_size (1024*8)
2  `define line_size 32
3  `define sector_size 4 // new definition for sector size
4  `define Associativity 8
5

TERMINAL
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector .\
cache_sector.v .\testbench_sector.v
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_sector
Hit Ratio= 99.24 Percentage, Cache Hits = 1488655, Total Simulation Addresses = 1500002
.\testbench_sector.v:63: $finish called at 15000015 (1s)
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3>
```

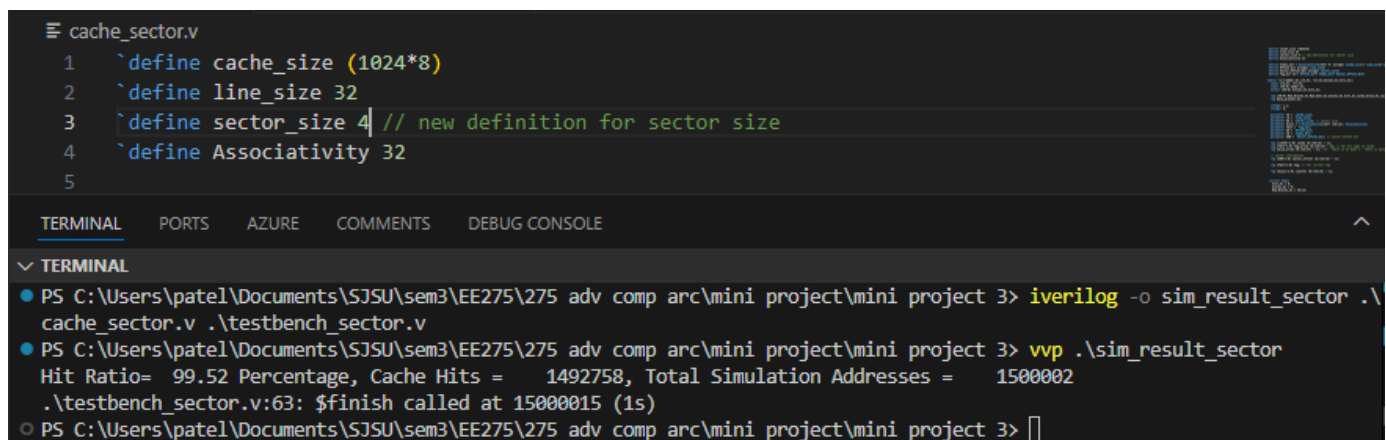
Associativity = 16:



```
cache_sector.v
1  `define cache_size (1024*8)
2  `define line_size 32
3  `define sector_size 4 // new definition for sector size
4  `define Associativity 16
5

TERMINAL
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector .\
cache_sector.v .\testbench_sector.v
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_sector
Hit Ratio= 99.38 Percentage, Cache Hits = 1490635, Total Simulation Addresses = 1500002
.\testbench_sector.v:63: $finish called at 15000015 (1s)
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3>
```

Associativity = 32 (Fully mapped):



```
cache_sector.v
1  `define cache_size (1024*8)
2  `define line_size 32
3  `define sector_size 4 // new definition for sector size
4  `define Associativity 32
5

TERMINAL
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector .\
cache_sector.v .\testbench_sector.v
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_sector
Hit Ratio= 99.52 Percentage, Cache Hits = 1492758, Total Simulation Addresses = 1500002
.\testbench_sector.v:63: $finish called at 15000015 (1s)
PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3>
```

Fourth case: Varying Sector Size (Fixed cache size= 8KB, Fixed Block Size=32 bytes, Fixed associativity = 4)

Sector Size=2:

```
cache_sector.v
1  `define cache_size (1024*8)
2  `define line_size 32
3  `define sector_size 2 // new definition for sector size
4  `define Associativity 4
5

TERMINAL PORTS AZURE COMMENTS DEBUG CONSOLE
▼ TERMINAL powershell + - [ ] [X]
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector .\
cache_sector.v .\testbench_sector.v
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_sector
Hit Ratio= 98.37 Percentage, Cache Hits = 1475487, Total Simulation Addresses = 1500002
.\testbench_sector.v:63: $finish called at 15000015 (1s)
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> [ ]
```

Sector Size=4:

```
cache_sector.v
1  `define cache_size (1024*8)
2  `define line_size 32
3  `define sector_size 4 // new definition for sector size
4  `define Associativity 4
5

TERMINAL PORTS AZURE COMMENTS DEBUG CONSOLE
▼ TERMINAL powershell + - [ ] [X]
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector .\
cache_sector.v .\testbench_sector.v
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_sector
Hit Ratio= 98.53 Percentage, Cache Hits = 1477927, Total Simulation Addresses = 1500002
.\testbench_sector.v:63: $finish called at 15000015 (1s)
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> [ ]
```

Sector Size=8:

```
cache_sector.v
1  `define cache_size (1024*8)
2  `define line_size 32
3  `define sector_size 8 // new definition for sector size
4  `define Associativity 4
5

TERMINAL PORTS AZURE COMMENTS DEBUG CONSOLE
▼ TERMINAL powershell + - [ ] [X]
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector .\
cache_sector.v .\testbench_sector.v
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_sector
Hit Ratio= 98.68 Percentage, Cache Hits = 1480129, Total Simulation Addresses = 1500002
.\testbench_sector.v:63: $finish called at 15000015 (1s)
• PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> [ ]
```

Sector Size=16:

```
cache_sector.v
1  `define cache_size (1024*8)
2  `define line_size 32
3  `define sector_size 16 // new definition for sector size
4  `define Associativity 4
5

TERMINAL  PORTS  AZURE  COMMENTS  DEBUG CONSOLE
✓ TERMINAL
● PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector .\
cache_sector.v .\testbench_sector.v
● PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_sector
Hit Ratio= 98.80 Percentage, Cache Hits = 1481955, Total Simulation Addresses = 1500002
.\testbench_sector.v:63: $finish called at 15000015 (1s)
```

Sector Size=32:

```
cache_sector.v
1  `define cache_size (1024*8)
2  `define line_size 32
3  `define sector_size 32 // new definition for sector size
4  `define Associativity 4
5

TERMINAL  PORTS  AZURE  COMMENTS  DEBUG CONSOLE
✓ TERMINAL
● PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> iverilog -o sim_result_sector .\
cache_sector.v .\testbench_sector.v
● PS C:\Users\patel\Documents\SJSU\sem3\EE275\275 adv comp arc\mini project\mini project 3> vvp .\sim_result_sector
Hit Ratio= 98.93 Percentage, Cache Hits = 1483879, Total Simulation Addresses = 1500002
.\testbench_sector.v:63: $finish called at 15000015 (1s)
```