Cache Coherence Protocol for a multi-core processor with Shared Cache Memory

Shubham Pandit
Dept. of Electrical Engineering
IIT Gandhinagar

Sai Shubham
Dept. of Electrical Engineering
IIT Gandhinagar

***Abstract*—Cache coherence is a fundamental aspect of computer architecture, particularly in modern multi-core systems where operations are massively parallel. It ensures that all processors or cores have a consistent view of memory at any given time, regardless of which processor accesses the data, avoiding inconsistencies that may lead to incorrect or unpredictable behaviour. The MESI protocol is a widely used cache coherence protocol that helps maintain cache coherence in shared-memory multiprocessor systems. Cache coherence also helps improve performance by minimizing memory traffic, as data is only transferred between caches and main memory when necessary**

***Index Terms*—Parallel Computing, multi-core systems, MESI protocol, Cache coherence**

I. Introduction

Parallel computing has become increasingly important in recent years due to the growing demand for high-performance computing in a wide range of fields, including scientific research, machine learning, and data analysis. One of the main advantages of parallel computing is that it can significantly reduce the time required to complete complex tasks. By breaking a large problem down into smaller, more manageable pieces and assigning these pieces to multiple processors, parallel computing can dramatically speed up computations.

Multicore systems have played a major role in the rise of parallel computing. A multicore system is a computer system that contains multiple processors or cores on a single chip. This allows multiple computations to be performed simultaneously on a single machine, which can significantly improve performance. Multicore systems have become increasingly prevalent in recent years, as the demand for high-performance computing has grown. They are now commonly found in desktop computers, laptops, and servers, as well as in specialized computing systems such as supercomputers and data centers.

Data inconsistency is a common issue that can arise in multicore systems when multiple processors or cores access and modify the same data simultaneously. In these systems, each processor typically has its own cache, which is a small, fast memory that stores frequently accessed data from main memory.

When multiple processors access the same data, each processor may have its own copy of the data in its cache. If one processor modifies its copy of the data, the other processors may not be aware of the modification, and their copies of the data may become inconsistent with the modified copy. This can lead to errors, race conditions, and other issues that can be difficult to diagnose and fix.

To address data inconsistency in multicore systems, cache coherence protocols are used. These protocols ensure that all processors have a consistent view of the data by tracking which processor has the most up-to-date copy of each cache line. If one processor modifies a cache line, the cache coherence protocol ensures that all other processors that have a copy of that line are notified and update their copies accordingly.

The MESI protocol is a widely used cache coherence protocol that helps maintain cache coherence in shared-memory multiprocessor systems such as 1. The cache controller resides inside each core, and snooping is done on System Bus between L2 cache and main memory. In our project, we have implemented MESI protocol on two core systems with a shared bus architecture, with each core having only one level of cache.

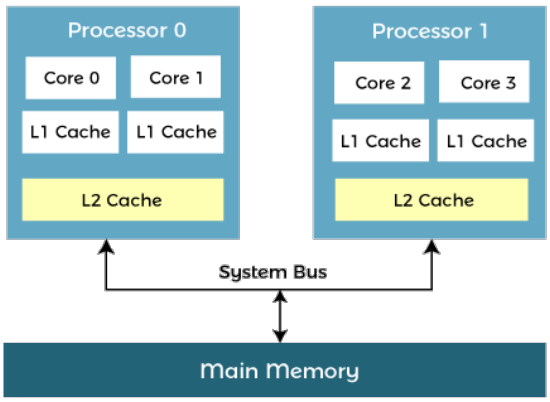

Fig. 1. Multi-core system Architecture

II. Working Principles

MESI stands for Modified, Exclusive, Shared, and Invalid, which are the four states that a cache line can be in under this protocol.

The working principle of the MESI protocol is based on the idea that each cache line can be in one of four states, depending on whether it has been modified or accessed by the processor and whether other processors share it. MESI protocol allows the use of a ”Write Back” system, i.e., a Cache line is written back to shared memory only when it is evicted

from the Cache minimizing the bus traffic and improving the performance, unlike the ”Write Through” system.

1) Modified (M): If a processor has modified a cache line, the line is marked as ”Modified.” This means that the cache line contains a valid copy of the data, and the data is not currently present in any other cache.

2) Exclusive (E): If a processor has accessed a cache line but has not modified it, the line is marked as ”Exclusive.” This means that the cache line contains a valid copy of the data, and the data is not currently present in any other cache.

3) Shared (S) : If a cache line is present in multiple caches and has not been modified, the line is marked as ”Shared.” This means that the cache line contains a valid copy of the data, and the data is also present in other caches.

4) Invalid (I): If a cache line is not valid, it is marked as ”Invalid.” This means that the cache line does not contain valid data and must be fetched from the main memory if needed.

When a processor accesses a cache line, the MESI protocol determines the current state of the line and updates it accordingly. For example, if a processor accesses a cache line in the ”Shared” state and wants to modify it, it will first put a request on Bus, notifying others that a particular cache line is going to be modified, so if they have the same cache line, they need to update the cache status to ”Invalid”. Then only, the protocol changes the state to ”Modified” in the local processor.

The MESI protocol also includes a mechanism for resolving conflicts between processors that attempt to access the same cache line simultaneously. When conflicts occur, the protocol uses a priority mechanism to determine which processor has priority in accessing the cache line by sending requests on the shared bus. Also, there are other methods to determine which processor gets the priority depending on which processor has the access to the shared bus, specifically in cases where there is a circular rotation of each processor's access to the shared bus.

III. MESI PROTOCOL IMPLEMENTATION

As shown in Fig 2, each cache line can receive a request to change its state from the local processor and the neighboring processors via bus, which we will denote as a bus request. The processor can issue two kinds of requests -

1) PrRd: Processor wants to read the cache line.
2) PrWr: Processor wants to write into the cache line.

Moreover, there are five types of Bus Requests that neighboring processors can issue -

1) BusRd: read request to a Cache line requested by another processor.
2) BusRdX: write request to a Cache line requested by another processor that doesn't already have the line.
3) BusUpgr: write request to a Cache block requested by another processor that already has that cache block residing in its own cache so that other snooping processors can mark their copy of the cache line as ”Invalid”.
4) FlushOpt: An entire cache block is put on the bus.
5) Fill: Shared memory to Cache transfer.

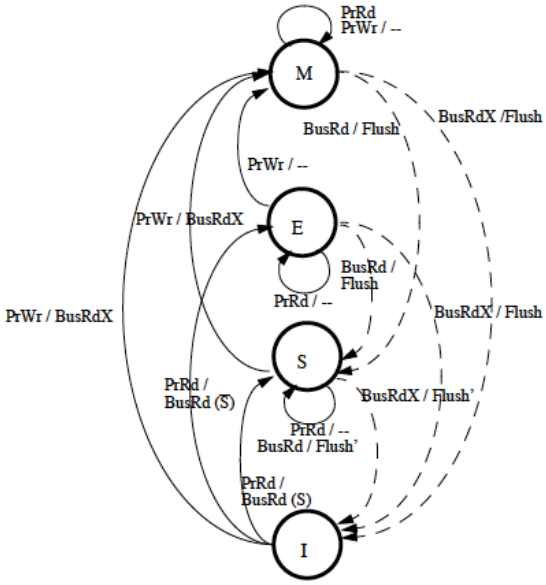

Fig. 2. MESI State-flow diagram

When Processor issues PrRd or PrWr, the following case can occur:

A. Read hit

When a processor issues a read request (PrRd) for a cache line that is already present in its cache, and the cache line is in the ”Modified,” ”Exclusive,” or ”Shared” state, the processor can directly read the data from its cache without accessing main memory. In this case, there is no state change. The cache line would not be in an ”Invalid” state as it would be a cache miss.

B. Read miss

When a processor issues a read request for a cache line that is not present in its cache and the cache line is in the ”Invalid” state in all other caches, the processor must fetch the data from the main memory and store it in ”Exclusive” state. In this case, the processor updates the state of the cache line to ”Exclusive” and stores the data in its cache. If the cache line is present in other cores in the ”Exclusive” or ”Shared” state, it will fetch the cache line from the main memory and store it in the ”Shared” state, and issue a request on the bus to all other cores to change the state of the corresponding state to ”Shared” state. Moreover, if the cache line is present in the ”Modified” state in any other cache, then memory access will be abandoned. That cache with the ”Modified” copy will put the data on the bus for the requesting processor to read. This is important as even the main memory will not have the latest copy of that Cache line as it's in a ”Modified” state, and we're using the ”Write Back” system here.

C. Write hit

When a processor issues a write request for a cache line that is already present in its cache, and the cache line is in

the ”Modified” or ”Exclusive” state, the processor can directly write the data to its cache without accessing the main memory or notifying other processors. In the case of ”Modified”, there is no state change. In the case of ”Exclusive”, the state is changed to ”Modified”. In the case of a ”Shared” state, the core must place a request on the bus for all other cores possessing the same cache line to invalidate its corresponding cache line. Once the request has been placed and received affirmatively by other cores, the local core changes its cache state to ”Modified”.

D. Write miss

When a processor issues a write request for a cache line that is not present in its cache, and the cache line is in the ”Invalid” state in all other caches, the processor must fetch the data from the main memory and update its cache. In this case, the processor updates the state of the cache line to ”Modified” and stores the data in its cache. In the case, it is present in other cores in any form other than the ”Invalid” state, it will issue a request on the bus to read the cache line from the core and invalidate its own copy of cache lines. Then the local processor writes to the newly-fetched cache and changes its state to ”Modified”.

It’s important to note that the state transitions in the MESI protocol are dependent on the access patterns of multiple processors, and conflicts can arise if multiple processors attempt to access the same cache line simultaneously. The MESI protocol includes mechanisms for resolving these conflicts and ensuring consistency between caches and main memory.

IV. Vivado Simulations

We have synthesised the design in Xilinx Vivado EDA tool, and have received zero warnings, zero critical warnings and zero errors. Figure 3 shows results of behavioral simulation implying that the design is working correctly.

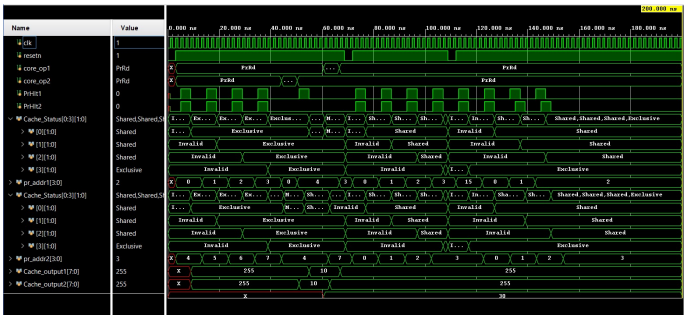

Fig. 3. MESI protocol design Vivado simulation

V. Genus Implementation

We have synthesised the design in Cadence Genus EDA tool and obtained the timing, power and area report. We have synthesised with clock frequency being 1.6ns. Figure 4 shows the timing report with slack observed to be just 4ps. Figure 6 shows the area report with total minimum area being 1426 μm^2. Figure 5 shows the power report with total power consumption being 908 μW. We have modelled L1 Cache using registers. So, it can be understood as 60% of power consumption is due to registers. MESI protocol, at its core is a Finite State Machine and hence 35% of power consumption is in combinational logic. Rest 5% accounts for the clock.

```
  Generated by:           Genus(TM) Synthesis Solution 21.10-p002_1
  Generated on:           Apr 15 2023  05:22:42 pm
  Module:                 MESI_FSM
  Technology library:     uk65lscllmvbbr_120c25_tc
  Operating conditions:   uk65lscllmvbbr_120c25_tc (balanced_tree)
  Wireload mode:          top
  Area mode:              timing library

          Pin                    Type       Fanout Load Slew Delay Arrival
                                                   (fF) (ps)  (ps)   (ps)

(clock clk)                  capture                                 1600 R
                             uncertainty                       -10   1590 R

Cost Group   : 'clk' (path_group 'clk')
Timing slack :       4ps
Start-point  : tag_directory_reg[1][0]/CK
End-point    : L1_Cache_reg[1][0]/D
```

Fig. 4. Genus Timing Report

```
Instance: /MESI_FSM
Power Unit: W
PDB Frames: /stim#0/frame#0
  Category         Leakage     Internal    Switching        Total    Row%
     memory     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
   register     3.94806e-08  4.85784e-04  4.44737e-05  5.30297e-04  58.43%
      latch     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
      logic     5.52895e-08  1.62644e-04  1.64395e-04  3.27095e-04  36.04%
       bbox     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
      clock     0.00000e+00  0.00000e+00  5.02200e-05  5.02200e-05   5.53%
        pad     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
         pm     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
  Subtotal     9.47700e-08  6.48429e-04  2.59089e-04  9.07612e-04 100.00%
Percentage          0.01%       71.44%       28.55%      100.00% 100.00%
```

Fig. 5. Genus Power Report

```
  Generated by:           Genus(TM) Synthesis Solution 21.10-p002_1
  Generated on:           Apr 15 2023  05:22:41 pm
  Module:                 MESI_FSM
  Technology library:     uk65lscllmvbbr_120c25_tc
  Operating conditions:   uk65lscllmvbbr_120c25_tc (balanced_tree)
  Wireload mode:          top
  Area mode:              timing library

Instance Module  Cell Count  Cell Area  Net Area   Total Area  Wireload
MESI_FSM               482   1426.320     0.000     1426.320   wl0 (D)
  (D) = wireload is default in technology library
  (T) = wireload mode is 'top'
```

Fig. 6. Genus Area Report

VI. NCSim Implementation

We have done post-synthesis simulation using Cadence NCSim EDA tool. The Figure 8 shows the console of the NCSim. There are zero setup and hold violations when the testbench clock frequency provided is 4ns. The Figure 7 shows the post-synthesis simulation of the design. The output delay observed is 116ps

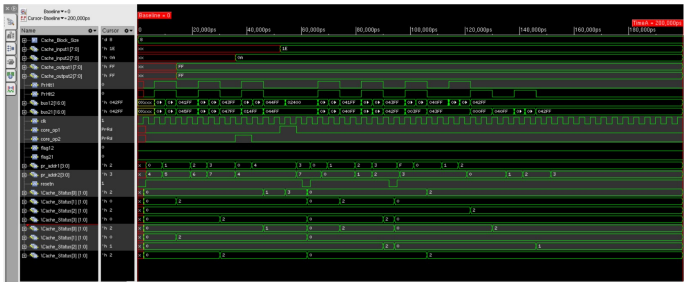

Fig. 7. NCSim Post-synthesis Report

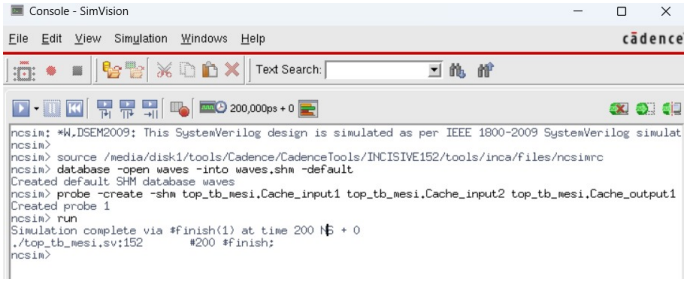

Fig. 8. NCSim Console Report: No Setup-Hold Violations

VII. Comparison with MSI protocol

Even though MESI protocol, when compared to MSI protocol, has one additional state called ”Exclusive,” it has the same extra area overhead as implementing 3-state (MSI protocol) or 4-state (MESI protocol), both will require an extra overhead of 2 bits in the tag directory. However, adding an ”Exclusive” state saves additional bus requests when the processor wants to read/write into an ”Exclusive” Cache line, as it’s clear that no other processor cache shares that same cache line, and thus no need to notify them. Saving bus requests can significantly reduce the operation latency, improving the performance, as to put the bus request, the processor needs to wait in line first to gain access to the bus.

Thus, MESI protocol is superior compared to the MSI protocol. More complex protocols like MOESI, MESIF, etc are also available out there.

VIII. Conclusion

In conclusion, the MESI protocol is a cache coherence protocol used in multicore systems to maintain consistency between caches and main memory. The protocol uses four states - Modified, Exclusive, Shared, and Invalid - to track the status of cache lines and updates them based on processor access patterns.

The MESI protocol provides efficient and effective cache coherence by allowing processors to directly read and write to cache lines that are in the ”Modified” or ”Exclusive” state, and by minimizing access to main memory. The protocol also includes mechanisms for resolving conflicts between processors that attempt to access the same cache line simultaneously.

Overall, the MESI protocol is a crucial component of modern multicore systems, as it enables multiple processors to share data efficiently and maintain consistency across multiple caches.

Acknowledgment

The authors acknowledge the help and guidance of Prof. Joycee Mekie, Tom Glint and Kailash Prasad in understanding the concepts better and implementing them.

References

[1] https://users.cs.utah.edu/ rajeev/cs6810/
[2] Culler, David (1997). Parallel Computer Architecture. Morgan Kaufmann Publishers. pp. Figure 5–15 State transition diagram for the Illinois MESI protocol. Pg 286