Document Outline for LLC Simulation Project Team:17-MSD-FALL-2024

Project Objectives:

The primary objective of this project is to model and simulate the behavior of a 16MB LLC that is 16-way set associative, employs a pseudo-LRU replacement policy, and adheres to the MESI protocol for cache coherence.

Key Goals:

- Simulate cache operations such as reads, writes, and snoops.
- Maintain cache coherency in a shared memory architecture involving multiple processors.
- Gather performance statistics to evaluate cache efficiency.

Project Objectives:

Objective: Simulate a Last-Level Cache (LLC).

LLC Specifications:

- Capacity: 16 MiB
- Associativity: 16-way set associative
- Line Size: 64-byte lines
- Write Policy: Write-allocate policy
- Replacement Policy: Pseudo-LRU
- Cache Coherence Protocol: MESI

Higher Level Cache: Model communication with the next-level cache, which uses 64-byte lines and is 4-way set associative.

Bus Operations: Handle operations including Read, Write, Invalidate, and RWIM.

Snooping Responses: Implement snooping results for NOHIT, HIT, and HITM.

Performance Metrics: Track and report the number of Reads, Writes, Hits, Misses, and the Hit Ratio.

Architecture Implementation Overview:

Cache Structure:

- Design a data structure for the LLC that consists of an array of cache lines.
- Each cache line should contain essential information such as:

Tag : Bits representing the address of the data

Valid Bit: 1 bit indicating if the line contains valid data.

Dirty Bit: bit to indicate if the line has been modified.

Mesi protocol: 3 bits for MESI protocol state representation.

Associativity:

- Implement the 16-way set associative mapping, which involves dividing the cache into sets. Each set will contain multiple cache lines (16 in this case).

Replacement Policy:

- Integrate the pseudo-LRU (Least Recently Used) algorithm for cache replacement decisions.
- Implement a pseudo-LRU replacement algorithm utilizing a 15-bit representation:
- The 15 bits will encode the relationship between cache lines within a set, allowing tracking of usage history.
- Each bit represents the usage state of each line (used or not used), enabling the simulation of LRU logic while reducing resource overhead.

Cache Coherence Protocol:

- Incorporate the MESI protocol to manage cache coherence in a multi-processor environment.
- Implement state management for each cache line, allowing transitions between Modified, Exclusive, Shared, and Invalid states.

Functionality:

Cache Operations:
Implement the core operations for the cache:

- **Read Operation:** Access the cache based on the calculated set index. If not found (cache miss), fetch the data from the higher-level cache and update the tag.
- **Write Operation:** Write data to the cache line and set the dirty bit if the line is modified.
- **Invalidate Operation:** Mark specified cache lines as invalid, ensuring no stale data is returned in response to requests.
- **RWIM (Read With Intent to Modify):** Handle reads that indicate an upcoming write, which can adjust cache states based on coherence needs.

Snooping Mechanism:
Implement a snoop controller to manage incoming requests from other processors:

- **NOHIT:** Indicates the data is not present in the cache.
- **HIT:** Indicates the requested data is found in the cache.
- **HITM:** Indicates the data is found in the cache and is modified; the cache must respond appropriately.

Trace Processing Modes:

Define two modes of operation for the simulation:

- **Normal Mode:** Provides detailed logging of all operations, including bus communication and cache access details, to facilitate debugging and understanding of cache behavior.
- **Silent Mode:** Suppresses extraneous output and only logs essential statistics (e.g., total reads, writes, hits, and misses), allowing for performance evaluation without verbose output.

Performance Monitoring:

Integrate mechanisms to track performance metrics throughout the simulation.

- Counting the total number of reads and writes.
- Maintaining statistics for cache hits and misses.
- Calculating the hit ratio to evaluate the efficiency of the cache implementation.

Testing and Validation:

Create a testing framework to validate functionality:

- Develop unit tests for each operation (read, write, invalidate).
- Test for cache coherence under MESI with various scenarios.
- Verify the pseudo-LRU replacement works correctly by simulating access patterns that necessitate line replacement.

Implementation:

- **Language** : System Verilog.

a) **Cache Logic :** Implement read/write operations and account for hits, misses, and evictions.

b) Integrate **pseudo-LRU** for replacement policy.
c) **MESI protocol** : Maintain states (**Modified, Exclusive, Shared, Invalid**) and transition rules.
d) Simulate snooping results and their impact on the cache.
e) **Bus Operations** : Trigger bus operations for cache coherence (e.g., invalidation, RWIM).
f) Handle snooping by reporting and reacting to results. Simulate other **processors' behavior** for testing (can use repeatable methods for snoop results for the other 2 processors).

Cache Implementation :

Total cache capacity = 16 MB
Cache line size = 64 bytes
Total number of lines in the cache = 16MB/64Bytes
$= 2^{24}/2^{6}$
$= 2^{18}$
= 256 Ki lines
Associativity = 16 way
Total number of sets in the cache = $2^{18}/2^{4}$
$= 2^{14}$
= 16 Ki sets
Address mapping:
Byte Select bits = $\log_2$(Cache line size)
= $\log_2$(64)
= 6 bits
Index bits = $\log_2$(Total number of sets in the cache)
= $\log_2$(16 Ki)
= 14 bits
Tag bits = Address size - (Byte select + Index)
= 32 - (6 + 14)
= 12 bits

Tag array implementation in Cache:
In each line:
Tag bits = 12 bits
Valid bit = 1 bit
Dirty bit = 1 bit
MESI States = 2 bits

In each set:
PLRU bits = 15 bits (per set)

Implementation in SytemVerilog:

To implement a cache structure, we can use an array of type struct which would contain only the tag array fields. Data field isn't present inside the cache field as the data is independent of cache behavior.
We can take two different struct arrays,

1. For the implementation of sub-fields of each line. This would contain Valid bit, Dirty bit and a Tag bit.
2. For the implementation of each set of sub-fields of each line along with the PLRU bits.

Assumed Test Cases :

1.Input Validation Test Case

Test Case 1.1: Valid Input Address

Description: Validate handling of a valid memory address.

Input: address = 0x0000ABCD

Procedure:

Call cache read operation with the input address.

Expected Result: Cache processes the read without errors, returns a status code indicating success.

Test Case 1.2: Invalid Input Address

Description: Validate rejection of an invalid memory address.

Input: address = 0xFFFFFFFF

Procedure:

Call cache read operation with the input address.

Expected Result: Cache raises an error and does not process the request.

2. Cache Read Hit Test Cases

Test Case 2.1: Read Hit in Cache

Description: Confirm cache can provide data on a hit.

Input: address = 0x0000ABCD (assume it's preloaded in cache)

Procedure:

Call cache read operation.

Expected Result: Return the data corresponding to the address with a hit status.

Test Case 2.2: Read Hit Updates Valid/Dirty Bits

Description: Check if valid and dirty bits are updated correctly.

Input: address = 0x0000XYZ (preloaded, dirty)

Procedure:

Call cache read operation.

Expected Result: Dirty bit remains set; valid bit is confirmed.

3. Cache Miss and Replacement Policy Test Cases

Test Case 3.1: Read Miss in Cache

Description: Handling of a read operation that misses in cache.

Input: address = 0x0000ZZZZ (not in cache)

Procedure:

Call cache read operation.

Expected Result: Cache should fetch from the next-level cache and update the cache.

Test Case 3.2: Cache Replacement Policy Trigger

Description: Ensure pseudo-LRU logic works correctly when replacing lines.

Input: Access enough addresses to fill the cache and trigger replacement.

Procedure:

Perform multiple writes and reads to fill the cache.

Access a new address that should cause replacement.

Expected Result: The least recently used line should be replaced.

4. Write Operations Test Cases

Test Case 4.1: Write Hit Behavior

Description: Validate successful write operation to an existing line.

Input: address = 0x0000ABCD (existing in cache)

Procedure:

Call write operation on the address.

Expected Result: Data should be written and dirty bit set.

Test Case 4.2: Write Miss Behavior (Write-Allocate)

Description: Check the write-allocate behavior when miss occurs.

Input: address = 0x0000ZZZZ (not in cache)

Procedure:

Call write operation on the address.

Expected Result: Cache fetches data and allocates, dirty bit should be set.

5. Snooping Mechanism Test Cases

Test Case 5.1: Snooping NOHIT Response

Description: Validate response to a snoop indicating NOHIT.

Input: Snoop request for data not in cache.

Procedure:

Simulate snoop for nonexistent data.

Expected Result: Cache returns NOHIT.

Test Case 5.2: Snooping HIT Response

Description: Confirm handling of a snooping HIT.

Input: Snoop request for data present in cache.

Procedure:

Simulate snoop for existing data.

Expected Result: Cache returns HIT status.

Test Case 5.3: Snooping HITM Response

Description: Ensure correct state transitions on HITM during snoop.

Input: Snoop for a modified line.

Procedure:

Simulate snoop for modified data.

Expected Result: Cache correctly handles HITM response.

6. Cache Coherence under MESI Protocol Test Cases

Test Case 6.1: Transition from Exclusive to Modified

Description: Validate transition from Exclusive to Modified on write.

Input: Write to line in Exclusive state.

Procedure:

Perform a write operation.

Expected Result: Line should transition to Modified state.

Test Case 6.2: Transition from Shared to Invalid on Invalidate

Description: Ensure proper transition on invalidation.

Input: Invalidate request for a line in Shared state.

Procedure:

Perform invalidate operation.

Expected Result: Line transitions to Invalid state.

Test Case 6.3: Multiple Processor Access Validation

Description: Check coherence when multiple processors access a shared line.

Input: Access by two different processors.

Procedure:

Simulate simultaneous reads by different processors.

Expected Result: Line should maintain its Shared state unless modified.

7. Performance Monitoring Test Cases

Test Case 7.1: Hit Ratio Calculation

Description: Validate accuracy of hit ratio calculation.

Input: Mixed workload of reads and writes.

Procedure:

Perform a series of cache operations.

Expected Result: Hit ratio is correctly calculated as hits / (hits + misses)

Test Case 7.2: Performance Metrics Logging

Description: Confirm comprehensive logging of performance metrics.

Input: Execute a series of cache operations.

Procedure:

Track counts of reads, writes, hits, and misses.

Expected Result: Log correctly reflects performance metrics.

8. Final Solution Cases

Test Case 8.1: Concurrent Writes Handling

Description: Evaluate cache behavior when multiple processors write to the same address.

Input: Processor 1 writes to 0x0000ABCD, and Processor 2 simultaneously writes to the same address.

Procedure:

Simulate both writes in parallel.

Expected Result: One write should succeed (the last writer) while the other should see a conflict or invalidation.

Test Case 8.2: Snoop Response to Cache Invalidations

Description: Check snoop response when an invalidation occurs for a shared line.

Input: Processor 1 invalidates 0x0000ABCD, which is cached by Processor 2 as shared.

Procedure:

Processor 1 sends an invalidate request.

Expected Result: Processor 2's cache line should transition to Invalid state.

Test Case 8.3: Performance Under Load Testing

Description: Evaluate cache performance under sustained load from multiple processors.

Input: Simulate workload where 10 processors read and write to various addresses simultaneously.

Procedure:

Execute the multi-threaded simulation.

Expected Result: Metrics such as hit ratio, miss penalty, and response time should reflect efficient cache usage.

Test Case 8.4: Final Coherence State Verification

Description: Confirm the overall coherence state after multiple transactions.

Input: After a series of interleaved reads and writes among all processors.

Procedure:

Collect state information from all caches.

Expected Result: All caches should reflect the correct and final coherence state per MESI protocol.

Data Structure Implementation Plan :
Sets = 16Ki set, **Lines** = 256Ki used to store **tag**, **state** and the **data**.
For line : **Tag** represents the memory block in the line, **state** is for representing whether the line has been modified (**M**), shared (**S**), exclusive (**E**), invalid (**I**), [**MESI protocol**].
Memory Address : **Tag**, **Set Index**, **Block Offset**.

Replacement Policy : Within the set, pseduo-LRU decides which line to evict when the set is full.
Might have to use a binary tree to track usage, each node represents a decision point (left or right)