**MICROPROCESSOR SYSTEM DESIGN**

**ECE 485/585, Fall 2024**

**LLC Simulation**

**With**

**MESI & Pseudo-LRU**

**FINAL PROJECT REPORT**

Team 17

* Thanmai Neelampalli (ID: 963346037)
* Sathwik Bandi (ID: 981066301)
* Bhanu Sai Sidhardha Tummala (ID: 987364882)
* Sai Ganesh Reddy Charian (ID: 957406061)

1. **Project Description:**

The objective of this project is to simulate the behavior of a **Last-Level Cache (LLC)** for a multi-processor system using SystemVerilog. The LLC will be shared by up to three processors in a coherent memory configuration, with a total cache capacity of 16MB, 64-byte cache lines, and 16-way set associativity. The simulation will implement the MESI protocol for cache coherence and use a pseudo-LRU replacement policy. Writing to Cache is done through write allocate policy. It will model cache operations, bus transactions, and snooping, focusing on simulating communication between the LLC and a next-level cache while maintaining inclusivity.  
  
The SystemVerilog simulation will include two modes: normal mode, which provides detailed log of bus operations, snoop results, and communication messages. The second mode is silent mode, which only reports key statistics such as cache hits, misses, and evictions. The LLC will handle bus operations like reads, writes, and invalidations, and will simulate snoop responses from other caches. The project will include comprehensive testing to validate cache behavior, coherency, eviction handling, and interactions with the higher-level cache to ensure the correctness of the simulation.

1. **Cache Design Specification:**Describe the interface and behavior of your LLC module. **2.1. Cache Parameters**

* **Capacity:** 16 MB
* **Line Size:** 64 bytes
* **Associativity:** 16-way set associative
* **Coherence Protocol:** MESI
* **Replacement Policy:** Pseudo-LRU
* **Write Policy:** Write allocate

**2.2. Supported Operations**

List and explain the bus operations:

* **READ:** Bus Read
* **WRITE:** Bus Write
* **INVALIDATE:** Bus Invalidate
* **RWIM:** Read with Intent to Modify

**2.3. Snoop Results**

Explain the snoop result types:

* **NOHIT:** No hit
* **HIT:** Hit
* **HITM:** Hit to modified line

**2.4. Interface to Higher-Level Cache**

Describe the communication messages:

* **GETLINE:** Request data for modified lines in L1
* **SENDLINE:** Send cache line to L1
* **INVALIDATELINE:** Invalidate an L1 line
* **EVICTLINE:** Evict a line from L1

1. **Cache Implementation:**Total cache capacity = 16 MB Cache line size = 64 bytes

Total number of lines in the cache = 16MB/64Bytes

= 224/26

= 218

= 256 Ki lines

Associativity = 16 way

Total number of sets in the cache = 256 Ki/16

= 218/24

= 214   
 = 16 Ki sets.

Address mapping:

Byte Select bits = log2(Cache line size)

= log2(64)

= 6 bits

Index bits = log2(Total number of sets in the cache)

= log2(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)

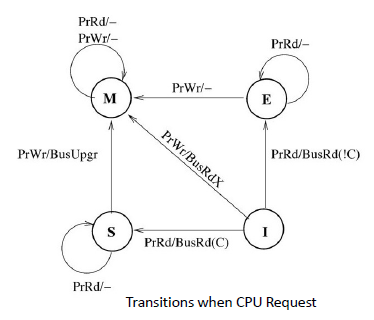
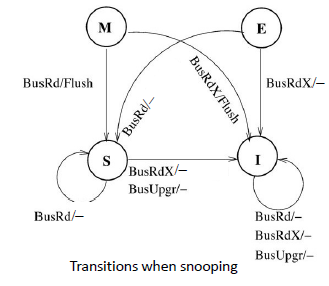
|  |  |  |
| --- | --- | --- |
| **Parameters** | **Calculations** | **Values** |
| No of lines in Cache | 16MB/64Bytes = 224/26 | 218= 256 Ki lines |
| No of sets in Cache | 256 Ki/16 = 218/24 | 214= 16 Ki sets |
| Byte select bits | log2(Cache line size) = log2(64) | 6 bits |
| Index bits | log2(No of sets in Cache) = log2(16 Ki) | 14 bits |
| Tag bits | Address size – (Byte select + Index)  = 32 – (6 + 14) | 12 bits |

1. A table with numbers and text

   Description automatically generated with medium confidence **Cache Design:**
2. **PLRU:**

**A diagram of a diagram

Description automatically generated**

1. **MESI:**
2. **Internal Design:**Detail the architecture and implementation of your LLC simulation. **7.1. Data Structures**

* Describe the cache structure (e.g., sets, ways, tags, states).
* Pseudo-LRU implementation

**7.2. Functions**

Explain key functions:

* **BusOperation()** - Simulates bus operations and captures snoop results.
* **GetSnoopResult() -** Returns snoop results.
* **PutSnoopResult() -** Reports snoop results to other caches.
* **MessageToCache() -** Simulates communication to the higher-level cache.

**7.3. Modes of Operation**

* **Normal Mode:** Displays detailed operations, bus transactions, and messages.
* **Silent Mode:** Displays only summary statistics and responses to 9 commands.

1. **Cache Functional Design:**

The cache functional design for this project focuses on implementing a 16MB Last-Level Cache (LLC) with 16-way set associativity and 64-byte cache lines. It uses a write-allocate policy to optimize write operations and ensure spatial locality. Cache coherence is maintained through the MESI protocol, enabling consistent state management across multiple processors in a shared memory configuration. The replacement policy employs a pseudo-LRU scheme, balancing efficiency and complexity in eviction decisions. To support inclusivity, the LLC design ensures that it maintains coherence with upper-level caches, modeling realistic hierarchical cache behavior.

The design incorporates robust communication mechanisms, including interfaces for simulating bus operations (READ, WRITE, RWIM, INVALIDATE) and reporting snoop results to coordinate with other processors’ caches. Two operational modes are supported: Normal Mode, which provides detailed logging of bus operations, snoop results, and cache communication; and Silent Mode, which focuses on reporting summary statistics, such as cache reads, writes, hits, misses, and hit ratio. The design collects these statistics to provide performance insights and validate the model's effectiveness.

The cache simulation is designed to handle variable input trace files without recompilation, utilizing $value$plusargs for runtime configuration. Simplified data handling focuses on state transitions and coherence rather than storing actual data, which streamlines the simulation while preserving accuracy. The modular structure ensures extensibility, allowing future scalability for changes in cache size, associativity, or replacement policy. This design ensures a comprehensive, flexible, and efficient simulation of LLC behavior in a multi-processor environment.

1. **Processor and Upper Level Cache Assumptions:**

List and justify any assumptions made during the project:

* **Processor Interface:** Assumed simplified interactions for trace-based simulation.
* **L1 Cache:** Assumed inclusivity policy with write-once mechanism.
* **Address Alignment:** No alignment or boundary crossing in addresses.

1. **Testing and Verification:**

**10.1. Test Plan**

Outline your test plan:

1. **Functionality Tests:** Verify basic operations (READ, WRITE, INVALIDATE).
2. **Coherence Tests:** Validate MESI transitions and cache coherence.
3. **Replacement Policy Tests:** Ensure pseudo-LRU works correctly.
4. **Boundary Cases:** Edge cases (e.g., cache full, conflicting addresses).

**10.2. Sample Test //Cases:** A screenshot of a computer

Description automatically generated

1. **Simulation Results:**

We got the expected outputs and verified them with the theoretical calculations.

1. **Conclusion:**

The simulation of the Last-Level Cache (LLC) for a multi-processor system successfully demonstrates the essential principles of modern cache design, coherence, and communication in shared memory environments. By implementing a 16MB, 16-way set-associative cache with a write-allocate policy and a pseudo-LRU replacement scheme, the project models an efficient and realistic cache system. The use of the MESI protocol ensures robust coherence management across multiple processors, highlighting the importance of maintaining consistency in shared data accesses.

The modular design allows for the simulation of critical operations such as cache hits, misses, evictions, and snoop responses, ensuring that the LLC behaves correctly under various workloads and configurations. Furthermore, the inclusion of Normal and Silent operational modes enables both detailed analysis and concise reporting of cache behavior, supporting thorough testing and evaluation. The system also incorporates extensibility, allowing for scalability in cache parameters and replacement policies, making it adaptable to future architectural changes.

Overall, the project demonstrates a comprehensive understanding of cache architecture and coherence protocols while providing a flexible and efficient simulation environment. This work serves as a foundation for further exploration of advanced caching techniques and their integration into modern multi-core processors. The insights gained from this simulation can contribute to the design of more efficient memory hierarchies in real-world computing systems.

1. **References:**

ECE 485/585 Course Materials

SystemVerilog IEEE Standard 1800-2017

Project Description Document

1. **Appendices:**
2. Full Code Listing:

Include your complete source code or link to your GitHub repository.

1. Trace Files:

Provide sample trace files used during testing.

1. Make file:

Include the makefile for compiling and running the simulation.