Project Report
On
Last Level Cache Design and Simulation

By,
Abhigna Bheemineni
Manikandeshwar Sasidhar
Preetam Basti
Swetha Chiliveri

1.Introduction:

With the advent of technology, the need for higher performance in computers is increasing. To achieve the higher performance the processors are made to run at a frequency as high as thousands of Megahertz to tens of Gigahertz. Even though the processor runs at such high frequencies memory systems cannot cope with them, due to which there has been search for alternative solution. The problem was solved by using the concept of "caching". Processor requests are routed through a small memory unit called cache before searching the entire memory. Cache stores the necessary information i.e., might it be data or instruction, which is frequently used by processor, so that processor requests are fulfilled as quickly as possible. If the information is not present in cache, then it is fetched from the memory.

2. Applications:

The concept of caching can be applied in various computing domains such as:

- Translation Look Aside Buffers (TLB) in Virtual memory.
- Caches in memory system as lower or higher-level caches.
- Caches in Graphics Processing Units (GPU) and Digital Signal Processors (DSP)
- Web caches, which store the websites, which are visited.

3. Technical Specification:

The last level cache (L2) design implemented in the project has the following specifications:

- Total Capacity = 16MB
- Byte Line = 64-byte lines
- Associativity = 8-way set
- Policy decisions for write miss = Write allocate
- Coherence Protocol = MESI
- Replacement Policy = pseudo-LRU scheme

L1 cache specifications:

- Byte Line = 64-byte lines
- Associativity = 4-way set

4. Assumptions:

We've made the following assumptions in designing the Last Level Cache:

- Each instruction takes only one clock cycle and all combinational elements, and the snoop signals are executed within aforementioned clock cycles.
- Since there is no message from L1 to L2, and there is modified cache line in L1 and we make a flush/eviction, we assume we get the Line from L1 and then issue a BusWrite.

5. Modes Used:

- **Silent mode:** Simulation displays only usage statistics summary and response to 9's in the trace file.

- **Normal mode:** Simulation displays everything i.e., the bus operations, reported snoop results and messages from L2 to L1 cache along with first mode

- **Debug mode:** to print the debug information during simulation

6. Multiprocessor Environment:

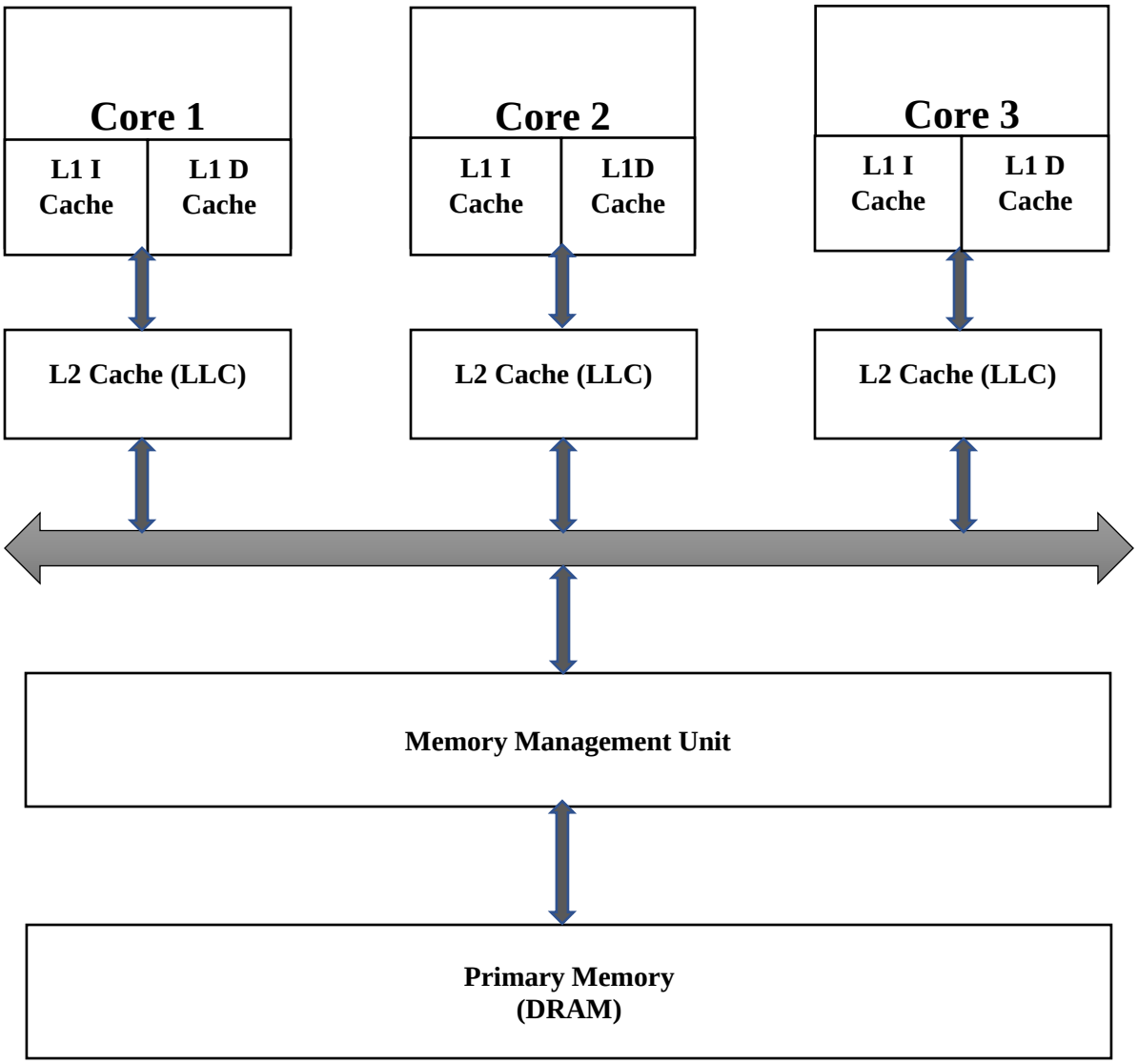
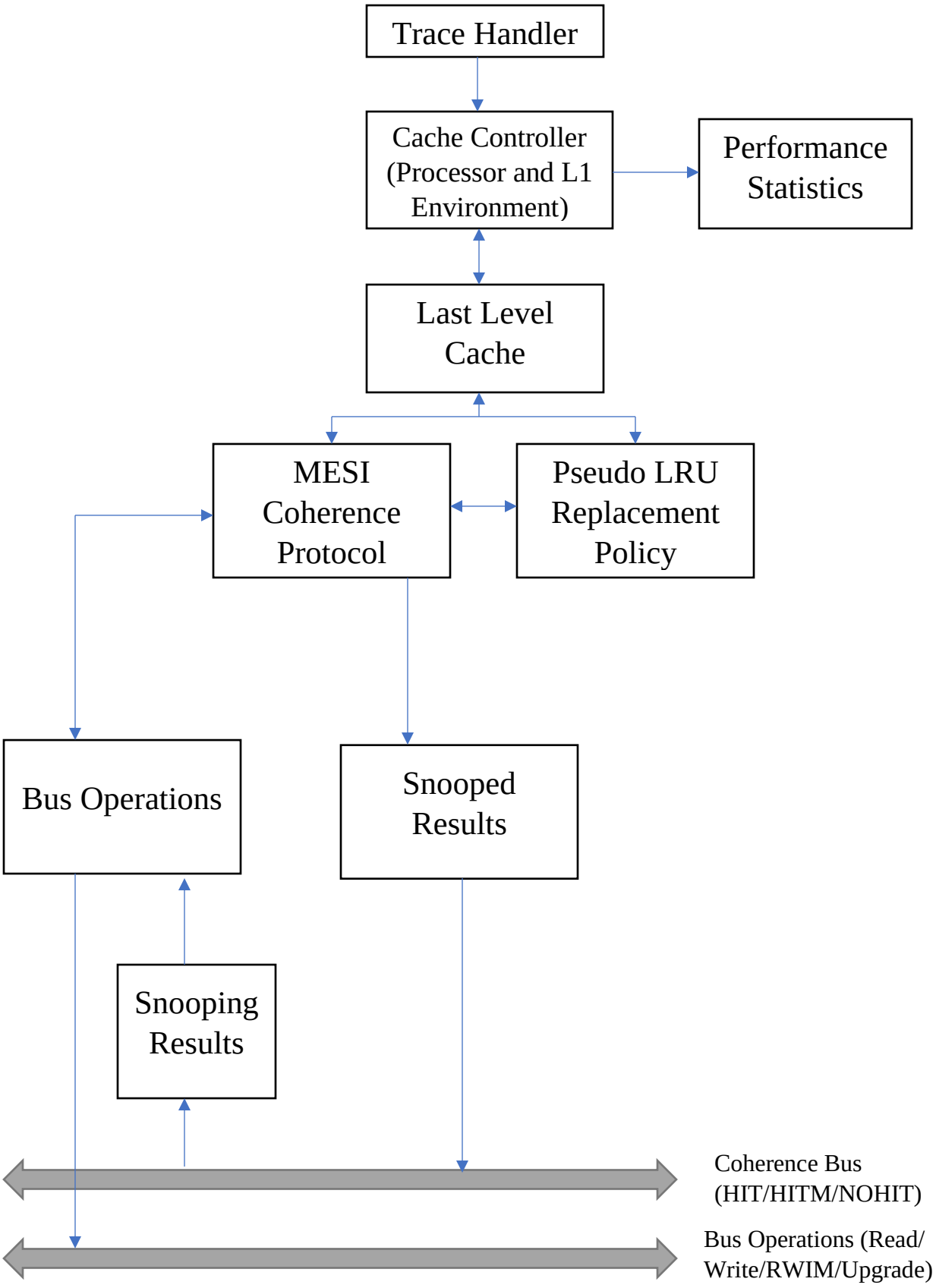


Fig 1: Shared Memory Configuration

7. Design Architecture:



8. Source Code:

a. SystemVerilog Packages:

All parameters and functions used in the design are defined in the following package files.

BusOperation.sv, CacheStructure.sv, GetSnoopResult.sv, MessageToCache.sv, ParameterDefinitions.sv, PLRU_Get.sv, PLRU_Update.sv, PutSnoopResult.sv

b. CacheDesign:

Implements the Last Level Cache as per the project specification.

CacheDesign.sv

c. TestBench:

The testbench serves as a Cache Controller and reads the commands from trace files and supplies the appropriate input signals to the Cache Design according to the function and address provided.

CacheTB.sv

9. Testing Strategies:

1. Basic Ports & Parameters test

- Tests the counters (Hit and miss), parameters and enums used and ports

2. Testing basic reads and writes to the cache

- Write to an address in the L2 cache (the entire cache set is empty) and then perform successive reads from the same address. All the reads should result in a read hit.
- Write to all 8 ways of an empty cache set and then read all the ways successively (which should result in 8 read hits)

3. Testing the PLRU

- Fill in an entire cache set (8 writes). In the next write to the same set, check which way is provided by the PLRU to be evicted.
- Fill an entire set (8 writes). Then perform reads in the set in such an order that the PLRU DOES NOT provide the least recently used way. This is to ensure that the PLRU doesn't have an accuracy of 100%.

4. Test Cases for MESI FSM

- when the cache line is in invalid state. L1 makes request to L2 and L2 reads from DRAM and send it to L1, and processor will write and L1 will change its state to modified state.
- When cache line is in invalid state, processor read (PrRd) is initiated by L2, then L2 goes to exclusive state. If the external processor snoops in same address and L2 then transition takes place from I to S, asserting cache signal. If cache signal is not asserted, then transition takes place from I to E. If external processor L2 makes BusRdX then local processor L2 moves from S to I.
- When cache line is in invalid state, when a processor L2 makes a processor write L2 goes to Modified state from invalid. External processor L2 will to a processor read from same address as local processor L2, this L2 makes transition from modified to shared resulting in flush.

5. Inclusivity test

- If there is a miss in L1 and a hit in L2, read or write request will be initiated. If it is L2 we send the required cache line. otherwise L2 should be written from where cache line should be sent to L1.

6. **Initialize / Clear cache**
    - Flood the cache with random values. Then assert a clear cache. On an immediate read after this, it should result in a cache miss followed by a DRAM read request.

7. **Write Allocate test**
    - Write to an empty cache line to acquire a miss. This will in turn produce a DRAM read request by the cache, the PLRU bits will be updated and the state will become M from I.
    - BusRdX should be asserted high.