Danny Tran
Brandon Hippe
Te Yan

Final Project Team 1 Report

Introduction

For this project, our goal was to design and simulate a last level cache of a new processor which can be used with up to 3 other processors.

Design Specifications

Our project was to create and simulate a last level cache. Our choice of language to simulate the cache was C. The specification of the cache is below:

- Total capacity of the cache is 16 MB
- Byte line size is 64 bytes
- 8-way associative cache
- MESI protocol for cache coherence
- Pseudo-LRU scheme for replacement policy
- Write allocate policy

Assumptions

Through our design process, there were a few assumptions that had to be made to make sure that simulation would be a success:

- **Data in the cache is irrelevant to the simulation:** The data in the cache should have no real effect on how the cache should function and operate, so we assume that we can ignore simulating data
- **Other caches are not being simulated**: We are only simulating a single L2 cache for a processor, so we assume that print statements communicating to other caches correctly simulate the response we would expect to get from then if they were physically simulated
- **Cache must maintain inclusivity:** Although we are only simulating the last level cache, communication between higher level cache and other same-level caches is important. L2 cache communicates directly with L1 cache, and monitors activity of other caches on bus through snooping.
- **L1 Cache employs write-once policy:** Part of the requirement, in L1, the first write to a line will be with a write-through policy and write after the first one will be with a write-back policy instead.

Design

Our design is written in C. All of our code is available in our Github repository, linked in the Appendix, as main.c, with function definitions and other useful definitions in main.h. We implemented functions for address parsing, PLRU updating and retrieval, searching the cache to find a tag or an empty location, each snooping operation, reading, writing, resetting the cache, and printing useful information from the cache simulation.

Example of our data structure for our cache:

```
typedef struct {
   char        MESI;
   unsigned int tag;
} Lines;

typedef struct {
   unsigned int PLRU;
   Lines lines[NUMWAYS];
} Set;
```

Part of the design was also in determining the interface for the cache. With 64 byte lines, our byte offset was the least significant 6 bits of a memory address. Given the total capacity of 16MB and the cache being 8-way set associative, our cache contains 32K sets, with 8 ways per set. It communicates with the L1 cache to signal the L1 cache with GETLINE, SENDLINE, INVALIDATELINE, and EVICTLINE operations, and snoops the operations of same level caches of other cores/processors along a shared databus.

Test Plan

In order to test our cache, we needed to create some test cases. We created tests to validate the functionality of our cache having an unoccupied miss, read conflict/evict miss, write conflict/evict miss, read hit, write hit, Pseudo LRU correctness after reading and writing, snooping an invalidation, snooping a read, snooping a write, and snooping a read with intent to modify. Total, we had 6 tests dedicated to read and write operations, 2 tests dedicated to testing LRU functionality, and 4 tests dedicated to snooping operations. The general outline for our test were:

- Ensure that the cache was empty
- Fill the cache with values in a particular set up the test case
- Execute the test case
- Verify that the output is correct

For an example:

LRU Test 1 Read

- Display cache set
- Fill cache set
- Access the first 2 lines
- Evict
- Should be the 4th way replaced

Our full test outline and test cases can be found in the Appendix.

Results

We ended up having a number of problems with our cache simulation for our demo. We had a few simple bugs, like not printing out the PLRU bits in verbose mode and doing a SENDLINE to L1 instead of a GETLINE when a RWIM occurred. We also made a mistake by updating the PLRU bits during a few of our snooping operations, which shouldn't happen, and likely was the cause of an issue with our PLRU bits getting set incorrectly at times. We also had an issue where when doing a RWIM, we didn't flush back to main memory.

Most of these issues could have been prevented with a more strict adherence to our test plan, as well as identifying the expected results of our tests.

Appendix

Test Outline: Final Project 1 Test Outline
Test Cases in Github Repository
Github Repository: Final Project Team 1