ECE485 Cash Controller Design Project

Jinho Park

Design Log

1. Objective
   1. Design a 4-way set associative L2 Cache for a 32-bit processor.
   2. Simulate the following three replacement policy: Random, LRU and Pseudo-LRU
2. Requirement Specification
   1. Cache & Cache Controller
      1. The Cache Line Size MUST BE greater than 4 bytes (32 bits)
      2. MUST implement **‘write-back’** and **‘write allocate’** policies
      3. MUST implement all of the following replacement policies
         1. Random
         2. LRU
         3. Pseudo-LRU
      4. The total Cache Size (including Tag, Valid, Dirty and LRU bits) SHOULD BE smaller than 2 MB for extra credit B
   2. Testbench
      1. MUST read memory access from a text file.
      2. MUST keep track of following key parameters
         1. Total number of memory references
         2. Number of reads
         3. Number of writes
         4. Number of cache hits
         5. Number of cache misses
         6. Hit ratio
      3. Should emulate a burst mode in memory read
      4. Might assume addressing by word and world-aligned memory
   3. Memory reference format in trace file



* + 1. Command: 0-read request/1-write request/2-instruction fetch(=read request)
    2. Address: – 32bit hex value

1. Design Decisions
   1. **Cache Size** **= 2MB** (to aim extra credit)
   2. **Cache Line Size = 64 bytes** – because it is the standard cache line size and the performance seems to be optimal between 16-64. (64 byte > 4 byte)
   3. **Number of Cache Lines**
      1. Random Replacement
         1. Cache line size calculation

(\*if the index bits are set to 13 bits, there should be 213 lines, which gives ((WordSize + 1 + 1 + NumTagBits)\* NumLines\*NumWays)/(8\*2^20) = **2.0508 MB > 2 MB**. Therefore, there should be 12 index bits)

* + - 1. Address bit configuration

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Configuration | Tag bits | Index | World Select | 00 |
| Size (32 total) | 14 | 12 | 4 | 2 |

* + - 1. Cache Line Configuration

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Configuration | Tag bits | Valid Bit | Dirty Bit | Data |
| Size | 14 bits | 1 bit | 1 bit | 64 bytes |

* + 1. LRU – Finite State Machine
       1. Cache line size calculation
       2. Address bit configuration

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Configuration | Tag bits | Index | World Select | 00 |
| Size (32 total) | 14 | 12 | 4 | 2 |

* + - 1. Cache Line Configuration

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Configuration | LRU Bits | Tag bits | Valid Bit | Dirty Bit | Data |
| Size | 5 bits | 14 bits | 1 bit | 1 bit | 64 bytes |

|  |
| --- |
| LRU Bits / line |
| 5 bits |

* + 1. LRU – Counter
       1. Cache line size calculation

* + - 1. Address bit configuration

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Configuration | Tag bits | Index | World Select | 00 |
| Size (32 total) | 14 | 12 | 4 | 2 |

* + - 1. Cache Line Configuration

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Configuration | LRU Bits | Tag bits | Valid Bit | Dirty Bit | Data |
| Size | 2 bits | 14 bits | 1 bit | 1 bit | 64 bytes |

* + 1. Pseudo-LRU
       1. Cache line size calculation

* + - 1. Address bit configuration

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Configuration | Tag bits | Index | World Select | 00 |
| Size (32 total) | 14 | 12 | 4 | 2 |

* + - 1. Cache Line Configuration

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Configuration | | Tag bits | | Valid Bit | Dirty Bit | Data |
| Size | | 14 bits | | 1 bit | 1 bit | 64 bytes |
| LRU Bits / line | |
| 3 bits | |

1. Block Diagram
   1. Level-0

Address(A31-A0)

Data(D63-D0)

Address(A0-A31)

Data(D63-D0)

Cache

Clock

* 1. Level-1

Addr

Data

Addr

Data

Clock

L2 Cache array

Cache

Controller

Addr

Data

Addr

Data

Addr

Data

1. Module Definitaion.
   1. **L1 Cache Module**
      1. Purpose: Read in memory reference from the trace file and make cache read/write request to L2 cache.
      2. Port Requirements
         1. Input : Data (64) – For the purpose of debugging.
         2. Output: WE(1) , Address (32), STB
      3. Algorithm

Open trace file

REPEAT

Read in memory reference

Output command and memory address

Until the last memory reference is read

End Algorithm

* + 1. Verilog Module Definition (prototype)

Module L1Cache(addr\_out, we, stb, data\_inout);

* 1. **Memory Module**
     1. Purpose: Send data upon L2 cache request using burst mode.
     2. Port Requirements
        1. Input: Command(1), Address(32)
        2. Output: Data(64), Strobe(1)
     3. Requirements
        1. Implement burst mode with burst length of 8.
        2. Send out strobe signal along with each chunk of data (source synchronous)
     4. Algorithm Memory

Read in command and data line

IF the command is read

WHILE loop control variable < burst length DO

Output address to the data port

Increment loop control variable

Else if the command is write

Output the address for the write

End Algorithm

* + 1. Verilog Module Definition

module Main\_Memory(addr\_in, RAS, CAS, CS, WE, clk, data\_inout, stb\_inout)

* 1. **L2 Cache Module**
     1. Purpose: Manage L2 cache according to the L1 cache request and the replacement policy.
     2. Requirements
        1. Read Miss
           1. Replacement
           2. Cache line fill
        2. Write Miss
           1. Replacement
           2. Cache line fill
     3. Algorithm L2 cache

Initialize L2 cache ( = create L2 cache array)

While (L1 Cache request Cache read/write)

Read in command and address line

Decode address( TAG? INDEX? WORD\_SELECT?)

IF command is read

IF CACHE **HIT**

OUTPUT DATA

ELSE IF CACHE **MISS**

IF EMPTY LINE PRESENT

READ IN A CACHE LINE

ELSE IF EMPTY LINE DOESN’T PRESENT

CHECK LRU

IF LINE DIRTY

WRITE BACK!

READ IN A CACHE LINE

OUTPUT DATA  
 UPDATE LRU

IF command is write

IF CACHE **HIT**

WRITE DATA

ELSE IF CACHE **MISS**

IF EMPTY LINE PRESENT

READ IN A CACHE LINE

WRITE DATA

ELSE IF EMPTY LINE DOESN’T PRESENT

CHECK LRU

WRITE DATA

UPDATE LRU

PROCEDURE Check\_Addr

Repeat for way N

Check if the line is valid

If the line is valid

Check if the TAG bits match

If the Tag bits match

Cache hit!

Output data to data line.

Else

Do nothing

UNTIL all 4 ways are checked

IF no match found

Procedure Replacement\_Manager

Output data to L1 cache

ELSE IF command is write

PROCEDURE Check\_Addr

IF no match found

Procedure Replacement\_Manager

(Write data to cache)

* + 1. Algorithms for procedures
       1. Check Addr
       2. Replacement Manager
    2. Verilog Module Definition

module L2CacheTest(addr\_out, RAS, CAS, CS, WE, we\_in, stb, addr\_in, data\_IO, data\_IO\_mem, stb\_IO);

1. Protocol
   1. L1-L2 Protocol
      1. Address Follower: When valid address present, process the request.

L1 Cache Module

L2 Cache Module

Memory Module

* + 1. Minimum Requirement
       1. Read
          1. Read Command (WE)→
          2. Address →
          3. (Address Ready(STB) →)
          4. ←Data
          5. ←Data Ready (STB)
          6. (Data Read Complete (ACK)→)
       2. Write
          1. Write Command (WE)→
          2. Data →
          3. Address →
          4. (Address Ready (STB)→)
          5. ←Data Write Complete (STB)
  1. L2-Memory Protocol
     1. Synchronous
     2. strobe In/Out
     3. Assuming Following Parameters for the SDRAM
        1. 4 GB
        2. 1KB Page Size/8 banks/tRCD=CL=3no ECC
        3. Using 4Gb SDRAM x 8 = 8 chips
        4. 32/8 = 4 data lines/chip
        5. 1024 M x 4 bits / chip
        6. 30 address bits for 1024 M – 32bit words
        7. **10 bits** of column address bits = 13 addr lines
        8. 30-10-3 = **17 row** address bits = 20 addr lines

|  |  |  |  |
| --- | --- | --- | --- |
| ROW | BANK | COLUMN | Byte |
| 17 | 3 | 10 | 2 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | CS# | RAS# | CAS# | WE# |
| ACTIVE | L | L | H | H |
| READ | L | H | L | H |
| WRITE | L | H | L | L |

* + 1. Read Protocol
       1. Clock0: CS#/RAS#/ROW ADDR →
       2. Clock 3:CS#/CAS#/COLUMN ADDR→
       3. Clock 6-9: ←Data x 8 (Burst)
    2. Write Protocol
       1. Clock0: CS#/RAS#/ROW ADDR →
       2. Clock 3:CS#/CAS#/WE#/COLUMN ADDR→
       3. Clock 6-9: Data x 8→ (Burst)
  1. Milestones
     1. Module definitions
     2. Write to memory and display
     3. Read from memory and display

1. To do 11/19/2009 6:09 PM
   1. Freak out for 5 min
   2. Quickly learn how to use revision control system
   3. Setup coding environment.
   4. Finish documentation.
   5. Progress. 12/3/20098:01 PM
      1. L1 Cache stub
         1. inout port

inout [63:0] data\_inout;

assign data\_inout=(data\_dir)?64'bz:write\_data;

* + - 1. handshaking (we & stb)
  1. To do
     1. L2 – Memory protocol
     2. L2 Memory Module stub

1. L2 Cache
   1. Cache Initialization
      1. Data only Cache
      2. Test Plan
         1. Fill All cache lines
         2. Read a word from a cache line.
2. MESI Protocol Support
   1. Assumptions
      1. HIT/HITM always not asserted
   2. Normal behavior
      1. Read Miss -> Cache Line Fill -> Exclusive
      2. Read Hit: -> No transition
      3. Write Miss/Write Hit: Modified (assert Invalidate All)
   3. Snooped Read
      1. Invalid : Do Nothing
      2. Shared: Do Nothing
      3. Modified: stall -> write back -> continue -> shared
      4. Exclusive: -> Shared
   4. Snooped Write
      1. Invalid: Do Nothing
      2. Shared: -> Invalid
      3. Modified: stall -> write back -> continue -> Invalid
      4. Exclusive: -> Invalid
   5. Bit Assignment
      1. 00 – Invalid
      2. 01 – Exclusive
      3. 10 – Shared
      4. 11 – Modified
   6. Added Ports
      1. Output: INV (active high)
      2. Input: snoop, (snoop result) – 2 bit
         1. 0X – No Snoop
         2. 10 – Snoop Read
         3. 11 – Snoop Write
3. Extra Credit Competition – Fully Associative Cache
   1. Cache Line Size
      1. # Tag Bits = 32-4-2 = 26
   2. Concept
      1. Look for match by comparing tags (index does not matter)
      2. Cache Hit – Do Nothing
      3. Cache Miss
         1. If Empty Slot Present –Cache Line Fill
         2. If Not, replace the line next to the line that was replaced
4. Testing
   1. Cache Initiation
      1. Parameters: 2 Ways, 2 Lines, 4 Words, Burst Length = 2

parameter BURST\_LENGTH = 2;

parameter CACHE\_WAY\_SIZE = 2;

parameter CACHE\_INDEX\_SIZE = 2;

parameter CACHE\_WORD\_WIDTH = 2;

parameter CACHE\_WORD\_MSB = 1;

* + 1. Initiate cache\_data so that the 4 MSBs represent the line # and the 4 LSB’s represent the word number in the line.
    2. Result:

SIM: Way: 0

SIM: Index: 0

SIM: Word: 0: Content: 00000000

SIM: Word: 1: Content: 00000001

SIM: Word: 2: Content: 00000002

SIM: Word: 3: Content: 00000003

SIM: Index: 1

SIM: Word: 0: Content: 10000000

SIM: Word: 1: Content: 10000001

SIM: Word: 2: Content: 10000002

SIM: Word: 3: Content: 10000003

SIM: Way: 1

SIM: Index: 0

SIM: Word: 0: Content: 00000000

SIM: Word: 1: Content: 00000001

SIM: Word: 2: Content: 00000002

SIM: Word: 3: Content: 00000003

SIM: Index: 1

SIM: Word: 0: Content: 10000000

SIM: Word: 1: Content: 10000001

SIM: Word: 2: Content: 10000002

SIM: Word: 3: Content: 10000003

* 1. Cache Line Fill
     1. Parameters: 2 Ways, 2 Lines, 4 Words, Burst Length = 2

parameter BURST\_LENGTH = 2;

parameter CACHE\_WAY\_SIZE = 2;

parameter CACHE\_INDEX\_SIZE = 2;

parameter CACHE\_WORD\_WIDTH = 2;

parameter CACHE\_WORD\_MSB = 1;

* + 1. Fill cache line 0 by causing a cache miss on line 0 with address 0xFFFc0000.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tag[31:18] | | | | | | | | | | | | | | Index[17:6] | | | | | | | | | | | | Word  [5:4] | | | | | 00 | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  | |  | |  |
| F | | | | F | | | | F | | | | C | | | | 0 | | | | 0 | | | | 0 | | | | | 0 | | | | | |
| 16383 | | | | | | | | | | | | | | 0 | | | | | | | | | | | | | 0 | | | | | | 0 | |

Dram output increment the address by 4(0x00000004) after each burst., so that each item stored represent the starting address of the 32 bit word.

* + 1. Expected Result of cache line fill:

|  |  |  |  |
| --- | --- | --- | --- |
| FFFF0000 | FFFF0004 | FFFF0008 | FFFF000C |

* + 1. Result:

SIM: command:2 address:fffc0**000**([17:6])

SIM: Tag: 16383 Index: 0 Word 0

SIM: L2 MISS

SIM: Task : Cache\_Line\_Fill

SIM: Way: 0, Tag: 16383, Line: 0

SIM: Way: 0

SIM: Index: 0

SIM: Word: 0: Content: fffc0000

SIM: Word: 1: Content: fffc0004

SIM: Word: 2: Content: fffc0008

SIM: Word: 3: Content: fffc000c

SIM: Index: 1

SIM: Word: 0: Content: 10000000

SIM: Word: 1: Content: 10000001

SIM: Word: 2: Content: 10000002

SIM: Word: 3: Content: 10000003

SIM: Way: 1

SIM: Index: 0

SIM: Word: 0: Content: 00000000

SIM: Word: 1: Content: 00000001

SIM: Word: 2: Content: 00000002

SIM: Word: 3: Content: 00000003

SIM: Index: 1

SIM: Word: 0: Content: 10000000

SIM: Word: 1: Content: 10000001

SIM: Word: 2: Content: 10000002

SIM: Word: 3: Content: 10000003

* 1. Cache Look Up (Hit)
     1. Parameters: 2 Ways, 2 Lines, 4 Words, Burst Length = 2

parameter BURST\_LENGTH = 2;

parameter CACHE\_WAY\_SIZE = 2;

parameter CACHE\_INDEX\_SIZE = 2;

parameter CACHE\_WORD\_WIDTH = 2;

parameter CACHE\_WORD\_MSB = 1;

* + 1. After fill the cache line as above with address FFFC0000 (Tag = 16383, Index = 0, Word = 0). Access FFFC0004, FFFC0008, FFFC000C (Tag = 16383, Index =1, Word = 1~3) and see if each addresses that match the tag and index it gives hit.

* + 1. Result

SIM: command:2 address:fffc0004

SIM: Tag: 16383 Index: 0 Word 1

SIM: **L2 HIT**

SIM: L2 outputs

SIM: Way: 0, Index: 0 Word 1

SIM: **Content: fffc0004**

SIM: ======================================================

SIM: command:2 address:fffc0008

SIM: Tag: 16383 Index: 0 Word 2

SIM: **L2 HIT**

SIM: L2 outputs

SIM: Way: 0, Index: 0 Word 2

SIM: **Content: fffc0008**

SIM: ======================================================

SIM: command:2 address:fffc000c

SIM: Tag: 16383 Index: 0 Word 3

SIM: **L2 HIT**

SIM: L2 outputs

SIM: Way: 0, Index: 0 Word 3

SIM: **Content: fffc000c**

======================================================

* 1. Cache Look Up (Miss)
     1. After fill the cache line as above with address FFFC0000 (Tag = 16383, Index = 0, Word = 0). Access FFFC0040 (Tag = 16383, Index =1, Word = 0) and see if it gives miss and result in cache line fill in line 1.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tag[31:18] | | | | | | | | | | | | | | Index[17:6] | | | | | | | | | | | | Word  [5:4] | | | | | 00 | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  | |  | |  |
| F | | | | F | | | | F | | | | C | | | | 0 | | | | 0 | | | | 4 | | | | | 0 | | | | | |
| 16383 | | | | | | | | | | | | | | **1** | | | | | | | | | | | | | 0 | | | | | | 0 | |

* + 1. Result:

SIM: ======================================================

SIM: command:2 address:fffc0040

SIM: Tag: 16383 **Index: 1** Word 0

SIM: **L2 MISS**

SIM: Task : **Cache\_Line\_Fill**

SIM: Way: 0

SIM: Index: 0

SIM: Word: 0: Content: fffc0000

SIM: Word: 1: Content: fffc0004

SIM: Word: 2: Content: fffc0008

SIM: Word: 3: Content: fffc000c

**SIM: Index: 1**

**SIM: Word: 0: Content: fffc0040**

**SIM: Word: 1: Content: fffc0044**

**SIM: Word: 2: Content: fffc0048**

**SIM: Word: 3: Content: fffc004c**

SIM: Way: 1

SIM: Index: 0

SIM: Word: 0: Content: 00000000

SIM: Word: 1: Content: 00000001

SIM: Word: 2: Content: 00000002

SIM: Word: 3: Content: 00000003

SIM: Index: 1

SIM: Word: 0: Content: 10000000

SIM: Word: 1: Content: 10000001

SIM: Word: 2: Content: 10000002

SIM: Word: 3: Content: 10000003

SIM:

* 1. Replacement
     1. Random
     2. LRU
     3. PLRU
  2. MESI
  3. Fully Associative