**CMPE 315: Principles of VLSI Design**

**Project Cover Page**

**Project Title : Cache Design [ Submission Part 1]**

**Name : Marinna Ricketts-Uy & Ben Thompson**

**Section : 01**

**Date Submitted : April 22, 2018**

**TA / Grader Use Only:**

**Late Submission Deduction (20% per day late):**

**Other Deductions:**

**Final Project Grade:**

**Comments to student:**

**CONTENTS Page#**

**----------------------------------------------------------------------------------------------------------**

**No table of contents entries found.**

**Introduction**

Implemented the two-way associative cache design.

**Design Process**

**Chip Design**

Figure x shows the block diagram for the overall design of the chip.

[Insert Block Diagram]

|  |  |
| --- | --- |
| Inputs | Outputs |
| CPU Address | Busy |
| CPU Data | Memory Enable |
| CPU RD/WR Signal | Memory Address |
| Start |  |
| CLK |  |
| Reset |  |
| Memory Data |  |
| Vdd |  |
| Gnd |  |

**Cache Design**

The cache is designed using the two-way associative cache design. The cache was designed using a hierarchical design. There is a 1-bit cache, 8-bit cache, cache block, and a full cache. There is a left and right cache.

Table x details the input and output signals for each cache.

|  |  |
| --- | --- |
| Inputs | Outputs |
| Reset | Valid Bit Data (vb\_data) |
| Cache Enable Signal (Cache\_en) | Tag Data (tag\_data) |
| Hit/Miss Enable Signal (HM\_en) |  |
| Byte Select (CS) |  |
| Block Select (RS) |  |
| ValidBit/Tag RD/WR Signal (vb\_tag\_rd\_wr) |  |
| RD/WR signal (rd\_wr) |  |
| Valid Bit Data (vb\_data) |  |
| Tag Data (tag\_data) |  |

On a reset, the contents of each cache are zeroed out. The cache enable signal is received from the state machine. The left cache is enabled when this signal is low and the right cache is enabled when the signal goes high. The hit/miss enable signal is used to signify whether the state machine is currently determining whether there is a hit or a miss. When this signal is high, this signifies that both caches must be enabled to check for a hit or miss in both caches. The byte select value is received from the byte offset decoder and is used to select which byte to read or write from. This signal is active low. The block select value is received from the block offset decoder and is used to select which block to read or write from. This signal is active low. The valid and tag rd/wr signal is used to signify whether to read or write to the valid and tag. This signal is being received from the state machine. The rd/wr signal is used to signify whether to read or write to the appropriate byte. The valid bit data and tag data are inputs from the state machine and outputs to the hit/miss entity. The state machine will write this data to the cache when it is performing a read miss operation.

**State Machine Design**

Figure x shows the diagram for the state machine.

[Insert State Machine Diagram]

Table x details the inputs and outputs for the state machine

|  |  |
| --- | --- |
| Inputs | Outputs |
| CLK | Busy |
| Reset | CPU Data Output Enable |
| Start | Memory Address Output Enable |
| Hit/Miss Left | Data Mux Select |
| Hit/Miss Right | Valid Bit / Tag RD/WR |
| RD/WR | Cache RD/WR |
| CPU Address | Hit/Miss Enable Signal |
| LRU | Cache Enable |
|  | LRU RD/WR |
|  | Address |
|  | VB Data Left |
|  | VB Data Right |
|  | Tag Data Left |
|  | Tag Data Right |

When the reset input signal goes high, the state machine starts in the initial state 0. For any of the states, if the reset input signal from the CPU goes high, the state machine will move to state 0. The state machine will remain in this state until the start signal from the CPU goes high. From state 0, if the RD/WR input signal signifies a read operation, the state machine will move to state 1, else the state machine will move to state 7. For a read operation, the state machine will wait to receive the outcome of the hit/miss operation. If the read operation results in a hit, the state machine will move to state 6. If the read operation results in a miss, the state machine will move to state 2. For a write operation, the state machine will wait to receive the outcome of the hit/miss operation. If the write operation results in a hit, the state machine will move to state 8. If the write operation results in a miss, the state machine will move to state 9.

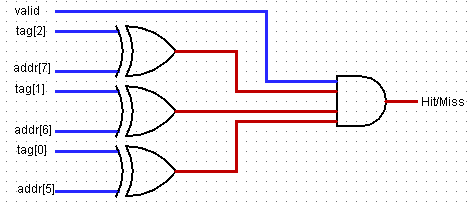
The busy output signal is set during states

**Hit/Miss Design**

Table x details the inputs and outputs for the hit/miss entity.

|  |  |
| --- | --- |
| Inputs | Outputs |
| Valid Data (valid) | Hit\_Miss (h\_m) |
| Tag Data [2:0] (tag) |  |
| CPU Address [7:5] (addr) |  |

Figure x shows the schematic for the hit/miss entity



The valid and tag data input for the hit/miss entity is an output from the cache. The CPU address bits and the tag data bits are compared to see whether they match. The output of each xor gate is then an input for an and gate with the valid bit. The hit/miss entity results in a hit when the valid bit is set to 1 and each CPU address bit matches the tag bits.

**Decoder Design**

There is a 2 to 4 decoder for the byte offset and a 3 to 8 decoder for the block offset. The decoders are both active low.

**LRU Design**

For the purpose of block replacement, there is a LRU entity that is used to determine the least recently used block in either of the caches. The LRU value for a block is written to when the state machine is at the end of a read hit/miss operation or a write hit operation. The state machine will read from the LRU entity during a read miss operation to see which cache block to write the new data to.

**Data Mux Design**

The data mux is used to select whether to be using memory data or cpu data. The data mux will be writing out data when

**Entity Architecture Pairs**

|  |  |
| --- | --- |
| **Entity** | **Using the Entity** |
| D\_latch |  |
| reg8 |  |
| oe8 |  |
| mux2to1\_8 |  |
| addr\_sel |  |
|  |  |
|  |  |

**Test Bench & Simulations**

Sample Inputs and Outputs

**Enscripted code**

**Work Break Up Sheet**

|  |  |
| --- | --- |
| **Ben** | **Marinna** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |