课程名称： 计算机体系结构 实验类型： 综合
实验项目名称：Topic 3. Cache Design
学生姓名：郑奕佳 专业： 计算机科学与技术 学号： 3200105861
同组学生姓名：李真泉 指导老师： 陈文智
实验地点： 曹西301 实验日期： 2022 年 11 月 23 日

一、 实验目的和要求

Understand Cache Line.
Understand the principle of Cache Management Unit (CMU) and State Machine of CMU.
Master the design methods of CMU.
Master the design methods of Cache Line.
master verification methods of Cache Line.

二、 实验内容和原理

Experiment Tasks

1. Design of Cache Line and CMU.
2. Verify the Cache Line and CMU.
3. Observe the Waveform of Simulation.

Cache Line Design

本次实验设计的cache为2路组关联结构，cache hit时采用回写法策略，cache miss时采用写分配法策略。cache结构如下图所示：

LRU	V	D	Tag	Data

使用recent位标记新放入的block，valid和dirty分别为有效位和脏位。

三、 实验过程和数据记录

主要实验任务是完成`cache.v`中的填空，具体功能与设计说明放在注释部分。

`cache.v`

```
    // the bits in an input address:
    wire [TAG_BITS-1:0] addr_tag;
    wire [SET_INDEX_WIDTH-1:0] addr_index;     // idx of set
```

```
    wire [ELEMENT_INDEX_WIDTH-1:0] addr_element1;
    wire [ELEMENT_INDEX_WIDTH-1:0] addr_element2;     // idx of element
    wire [ELEMENT_INDEX_WIDTH+ELEMENT_WORDS_WIDTH-1:0] addr_word1;
    wire [ELEMENT_INDEX_WIDTH+ELEMENT_WORDS_WIDTH-1:0] addr_word2; // element
index + word index

    assign addr_tag = addr[ADDR_BITS-1:ADDR_BITS-TAG_BITS]; //依据图示分配tag位宽
    assign addr_index = addr[ADDR_BITS-TAG_BITS-
1:ELEMENT_WORDS_WIDTH+WORD_BYTES_WIDTH];////依据图示分配index位宽
    assign addr_element1 = {addr_index, 1'b0};
    assign addr_element2 = {addr_index, 1'b1};      //2路组关联
    assign addr_word1 = {addr_element1,
addr[ELEMENT_WORDS_WIDTH+WORD_BYTES_WIDTH-1:WORD_BYTES_WIDTH]};
    assign addr_word2 = {addr_element2,
addr[ELEMENT_WORDS_WIDTH+WORD_BYTES_WIDTH-1:WORD_BYTES_WIDTH]};           //need
to fill in

    assign word1 = inner_data[addr_word1];
    assign word2 = inner_data[addr_word2];                //need to fill in
    assign half_word1 = addr[1] ? word1[31:16] : word1[15:0];
    assign half_word2 = addr[1] ? word2[31:16] : word2[15:0];           //need
to fill in
    assign byte1 = addr[1] ?
                   addr[0] ? word1[31:24] : word1[23:16] :
                   addr[0] ? word1[15:8] :  word1[7:0]   ;
    assign byte2 = addr[1] ?
                   addr[0] ? word2[31:24] : word2[23:16] :
                   addr[0] ? word2[15:8] :  word2[7:0]   ;//need to fill in

    assign recent1 = inner_recent[addr_element1];
    assign recent2 = inner_recent[addr_element2];  //设置recent，valid，dirty位
    assign valid1 = inner_valid[addr_element1];
    assign valid2 = inner_valid[addr_element2];  //need to fill in
    assign dirty1 = inner_dirty[addr_element1];
    assign dirty2 = inner_dirty[addr_element2]; //need to fill in
    assign tag1 = inner_tag[addr_element1];
    assign tag2 = inner_tag[addr_element2]; //取tag的值

    assign hit1 = valid1 & (tag1 == addr_tag);
    assign hit2 = valid2 & (tag2 == addr_tag); //有效位为1，且tag与访问地址tag相同，则
命中

    always @ (posedge clk) begin
        valid <= recent1 ? valid2 : valid1;//如果最新放入的是1，则取2的valid，dirty，
tag，准备替换2
        dirty <= recent1 ? dirty2 : dirty1;                  //need to fill in
        tag <= recent1 ? tag2 : tag1;                     //need to fill in
        hit <= hit1 | hit2; //如果1和2有任一hit，则cache hit

        // read $ with load==0 means moving data from $ to mem
        // no need to update recent bit
        // otherwise the refresh process will be affected
        if (load) begin
            if (hit1) begin
                dout <=
```

```
                    u_b_h_w[1] ? word1 :
                    u_b_h_w[0] ? {u_b_h_w[2] ? 16'b0 : {16{half_word1[15]}},
half_word1} :
                    {u_b_h_w[2] ? 24'b0 : {24{byte1[7]}}, byte1};

                // inner_recent will be refreshed only on r/w hit
                // (including the r/w hit after miss and replacement)
                inner_recent[addr_element1] <= 1'b1;
                inner_recent[addr_element2] <= 1'b0;
            end
            else if (hit2) begin
                    //与hit1时镜像填入hit2的取值与更新recent处理
                dout <=
                    u_b_h_w[1] ? word2 :
                    u_b_h_w[0] ? {u_b_h_w[2] ? 16'b0 : {16{half_word2[15]}},
half_word2} :
                    {u_b_h_w[2] ? 24'b0 : {24{byte2[7]}}, byte2};

                // inner_recent will be refreshed only on r/w hit
                // (including the r/w hit after miss and replacement)
                inner_recent[addr_element1] <= 1'b0;
                inner_recent[addr_element2] <= 1'b1;
            end
        end
        else dout <= inner_data[ recent1 ? addr_word2 : addr_word1 ];

        if (edit) begin
            if (hit1) begin
                inner_data[addr_word1] <=
                    u_b_h_w[1] ?        // word?
                        din
                    :
                        u_b_h_w[0] ?    // half word?
                            addr[1] ?       // upper / lower?
                                {din[15:0], word1[15:0]}
                            :
                                {word1[31:16], din[15:0]}
                        :   // byte
                            addr[1] ?
                                addr[0] ?
                                    {din[7:0], word1[23:0]}   // 11
                                :
                                    {word1[31:24], din[7:0], word1[15:0]} // 10
                            :
                                addr[0] ?
                                    {word1[31:16], din[7:0], word1[7:0]}   // 01
                                :
                                    {word1[31:8], din[7:0]} // 00
                ;
                inner_dirty[addr_element1] <= 1'b1;
                inner_recent[addr_element1] <= 1'b1;
                inner_recent[addr_element2] <= 1'b0;
            end
            else if (hit2) begin
                    //与hit1时镜像填入hit2的写入数据，更新dirty和recent
```

```
                inner_data[addr_word2] <=
                      u_b_h_w[1] ?        // word?
                      din
                      :
                      u_b_h_w[0] ?    // half word?
                      addr[1] ?       // upper / lower?
                      {din[15:0], word2[15:0]}
                      :
                      {word2[31:16], din[15:0]}
          :   // byte
                      addr[1] ?
                      addr[0] ?
                      {din[7:0], word2[23:0]}   // 11
                      :
                      {word2[31:24], din[7:0], word2[15:0]} // 10
          :
                      addr[0] ?
                      {word2[31:16], din[7:0], word2[7:0]}   // 01
                      :
                      {word2[31:8], din[7:0]} // 00
                      ;
            inner_dirty[addr_element2] <= 1'b1;
            inner_recent[addr_element1] <= 1'b0;
            inner_recent[addr_element2] <= 1'b1;
            end
        end

        if (store) begin
            if (recent1) begin  // replace 2
                inner_data[addr_word2] <= din;
                inner_valid[addr_element2] <= 1'b1;
                inner_dirty[addr_element2] <= 1'b0;
                inner_tag[addr_element2] <= addr_tag;
            end else begin

                //如果recent1为0，recent2为1时就替换1，recent2位0时就说明这一组都未填入
数据，直接放入1即可
                inner_data[addr_word1] <= din;
                inner_valid[addr_element1] <= 1'b1;
                inner_dirty[addr_element1] <= 1'b0;
                inner_tag[addr_element1] <= addr_tag;
            end
        end

        // not used currently, can be used to reset the cache.
        if (invalid) begin
            inner_recent[addr_element1] <= 1'b0;
            inner_recent[addr_element2] <= 1'b0;
            inner_valid[addr_element1] <= 1'b0;
            inner_valid[addr_element2] <= 1'b0;
            inner_dirty[addr_element1] <= 1'b0;
            inner_dirty[addr_element2] <= 1'b0;
        end
    end
```

```
endmodule
```

四、 实验结果分析

本实验通过仿真波形验证正确性：

仿真测试代码设计为先初始化，read miss与hit；write miss与hit；读0组0行后该行recent被应该设为1，执行store时应该被存入0组1行；读0组1行hit，设置recent和dirty；读0组0行hit，设置recent；read miss，替换0组1行。

对比仿真波形可知，cache实现了预期功能。