# Lab 7 report

PB22051022王嘉宁

实验目的与内容

> 学习高速缓存 Cache 的组织结构和工作机理

逻辑设计

任务 1：二路组相连 Cache

```
module cache (···);
    wire    [     2*LINE_WIDTH-1 : 0]    r_line;       //I DO
    wire    [       LINE_WIDTH-1 : 0]    r_line_wb;    //I DO
    wire    [      2*TAG_WIDTH-1 : 0]    r_tag;        //I DO
    wire    [                  1 : 0]    valid;        //I DO
    wire    [                  1 : 0]    dirty;        //I DO
    wire    [                  1 : 0]    hit;          //I DO
    reg    [ 1 : 0]    data_we;       //I DO
    reg    [ 1 : 0]    tag_we;        //I DO

//I DO
    reg    [7 : 0]    age;
    reg                tar;
    reg                tar_n;

    always @(posedge clk or negedge rstn) begin
        if (!rstn) begin
            age <= 0;                 //I DO
        end
        else begin
            if (addr_buf_we) begin
                tar <= tar_n;         //I DO
            end
        end
    end
    assign dirty_mem_addr = (tar == 4'd0) ? {r_tag[    TAG_WIDTH - 1 :         0],
w_index} << (LINE_OFFSET_WIDTH + SPACE_OFFSET) :
                            (tar == 4'd1) ? {r_tag[2 * TAG_WIDTH - 1 : TAG_WIDTH],
w_index} << (LINE_OFFSET_WIDTH + SPACE_OFFSET) :
                                        { {(TAG_WIDTH){1'b0}} , w_index} <<
(LINE_OFFSET_WIDTH + SPACE_OFFSET);//I DO

    bram #(
    ) tag_bram0(
        .we(tag_we[0]),                                         //I DO
        .dout({valid[0], dirty[0], r_tag[TAG_WIDTH - 1 : 0]})      //I DO
    );
    bram #(
    ) tag_bram1(
        .we(tag_we[1]),                                          //I DO
        .dout({valid[1], dirty[1], r_tag[2 * TAG_WIDTH - 1 : TAG_WIDTH]})//I DO
    );

    bram #(
    ) data_bram0(
        .we(data_we[0]),                                //I DO
        .dout(r_line[LINE_WIDTH - 1 : 0])               //I DO
    );
    bram #(
    ) data_bram1(
        .we(data_we[1]),                                //I DO
        .dout(r_line[2 * LINE_WIDTH - 1 : LINE_WIDTH])    //I DO
    );

    assign hit[0] = valid[0] && r_tag[    TAG_WIDTH - 1 :         0] == tag;//I DO
    assign hit[1] = valid[1] && r_tag[2 * TAG_WIDTH - 1 : TAG_WIDTH] == tag;//I DO
    assign r_line_wb =   hit[0] ? r_line[LINE_WIDTH - 1 : 0] :
                        hit[1] ? r_line[2 * LINE_WIDTH - 1 : LINE_WIDTH] :
{(LINE_WIDTH){1'b0}};//I DO

    always @(posedge clk) begin
        if (tar == 0) age[r_index] <= 0;
        else if (tar == 1) age[r_index] <= 1;
    end// I DO

    reg [31:0] dirty_mem_addr_buf;
    reg [127:0] dirty_mem_data_buf;
    always @(posedge clk or negedge rstn) begin
        if (!rstn) begin
        end
        else begin
            if (CS == READ || CS == WRITE) begin
                dirty_mem_data_buf <= (tar == 4'd0) ? r_line[LINE_WIDTH - 1 : 0] :
                                    (tar == 4'd1) ? r_line[2 * LINE_WIDTH - 1 :
LINE_WIDTH] : {(LINE_WIDTH){1'b0}};
            end//I DO
        end
    end

    always @(*) begin
        case (word_offset)
            0: begin
                cache_data = r_line_wb[31:0];//I DO
            end
            1: begin
                cache_data = r_line_wb[63:32];// I DO
            end
            2: begin
                cache_data = r_line_wb[95:64];// I DO
            end
            3: begin
                cache_data = r_line_wb[127:96];// I DO
            end
        endcase
    end

    always @(*) begin
        case(CS)
            READ: begin
                if (miss && !dirty[tar]) begin//I DO
                    NS = MISS;
                end else if (miss && dirty[tar]) begin //I DO
                    NS = W_DIRTY;
                end
            end
            WRITE: begin
                if (miss && !dirty[tar]) begin //I DO
                    NS = MISS;
                end else if (miss && dirty[tar]) begin //I DO
                    NS = W_DIRTY;
                end
            end
        endcase
    end

    always @(*) begin
        case(CS)
            IDLE: begin
                if(refill) begin
                    data_we = 1 << tar;//I DO
                    tag_we = 1 << tar; //I DO
                end
            end
            READ: begin
                if (hit != 2'b0) begin
                end else begin
                    if(dirty[tar])begin //I DO
                        mem_w_data = r_line[tar * LINE_WIDTH +: LINE_WIDTH]; // I
DO
                    end
                end
            end
            WRITE: begin
                if (hit!=2'b0) begin
                    if(hit[0]) begin  // I DO
                        data_we = 2'h1;
                        tag_we = 2'h1;
                    end
                    else begin       // I DO
                        data_we = 2'h2;
                        tag_we = 2'h2;
                    end
                end else begin
                    if(dirty[tar])begin  //I DO
                        mem_w_data = r_line[tar * LINE_WIDTH +: LINE_WIDTH]; // I
DO
                    end
                end
            end
        endcase
    end

always @(posedge clk) begin // I DO
    if(age[r_index]) tar_n <= 0;
    else tar_n <= 1;
end
endmodule
```

多样化的替换策略

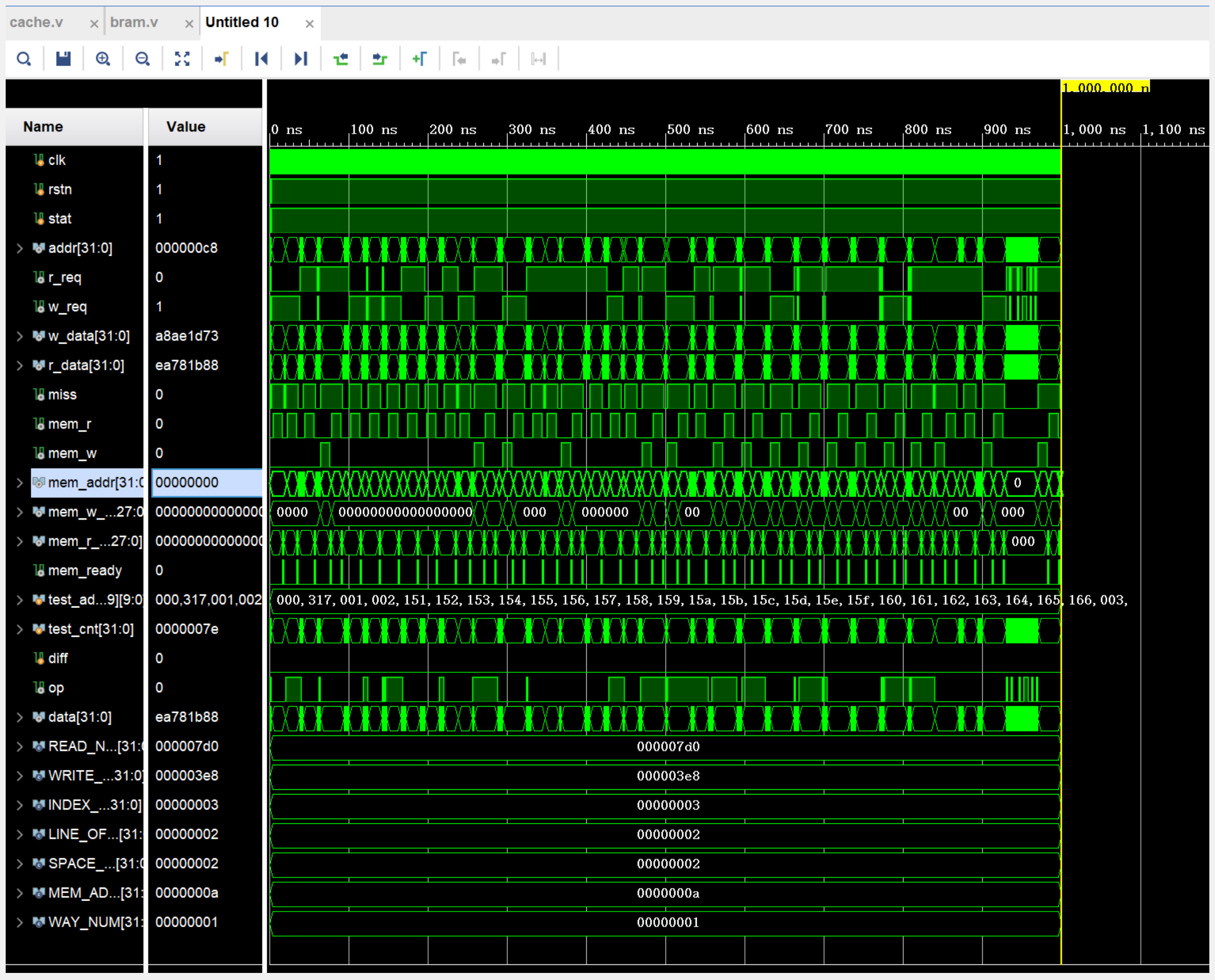
先进先出法（FIFO）

> 与LRU无异

随机替换

```
always @(posedge clk) begin
    age[r_index] <= $random%2;
end
```

测试结果与分析

> 随机替换结果
>
> 
>
>
> | Name | Value |
> |---|---|
> | clk | 1 |
> | rstn | 1 |
> | stat | 1 |
> | addr[31:0] | 000000c8 |
> | r_req | 0 |
> | w_req | 1 |
> | w_data[31:0] | a8ae1d73 |
> | r_data[31:0] | ea781b88 |
> | miss | 0 |
> | mem_r | 0 |
> | mem_w | 0 |
> | mem_addr[31:0 | 00000000 |
> | mem_w_...27:0 | 00000000000000 |
> | mem_r_...27:0] | 00000000000000 |
> | mem_ready | 0 |
> | test_ad...9][9:0 | 000,317,001,002 |
> | test_cnt[31:0] | 0000007e |
> | diff | 0 |
> | op | 0 |
> | data[31:0] | ea781b88 |
> | READ_N...[31:0 | 000007d0 |
> | WRITE_...31:0] | 000003e8 |
> | INDEX_...31:0] | 00000003 |
> | LINE_OF...[31: | 00000002 |
> | SPACE_...[31:0 | 00000002 |
> | MEM_AD...[31: | 0000000a |
> | WAY_NUM[31: | 00000001 |

总结

> 学会了二路组相连与三种替换方式。