

**本科实验报告**

|  |  |
| --- | --- |
| 课程名称： | 计算机体系结构 |
| 姓 名： | 汪辉 |
| 学 院： | 计算机科学与技术学院 |
| 系： |  |
| 专 业： | 计算机科学与技术 |
| 学 号： | 3190105609 |
| 指导教师： | 陈文智 |

2021年11月2日

**浙江大学实验报告**

课程名称： 计算机体系结构 实验类型： 综合

实验项目名称： Cache Design

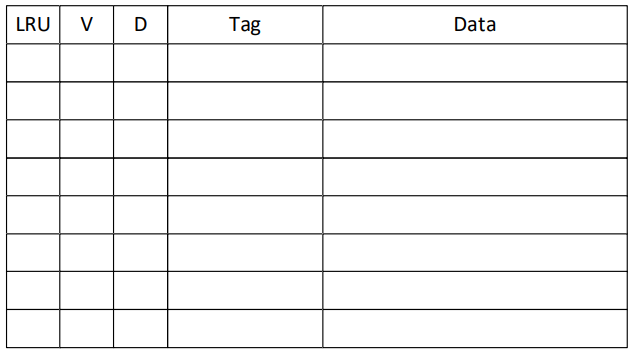
学生姓名： 汪辉 专业： 计算机科学与技术 学号： 3190105609

同组学生姓名： 王嘉豪 指导老师： 陈文智

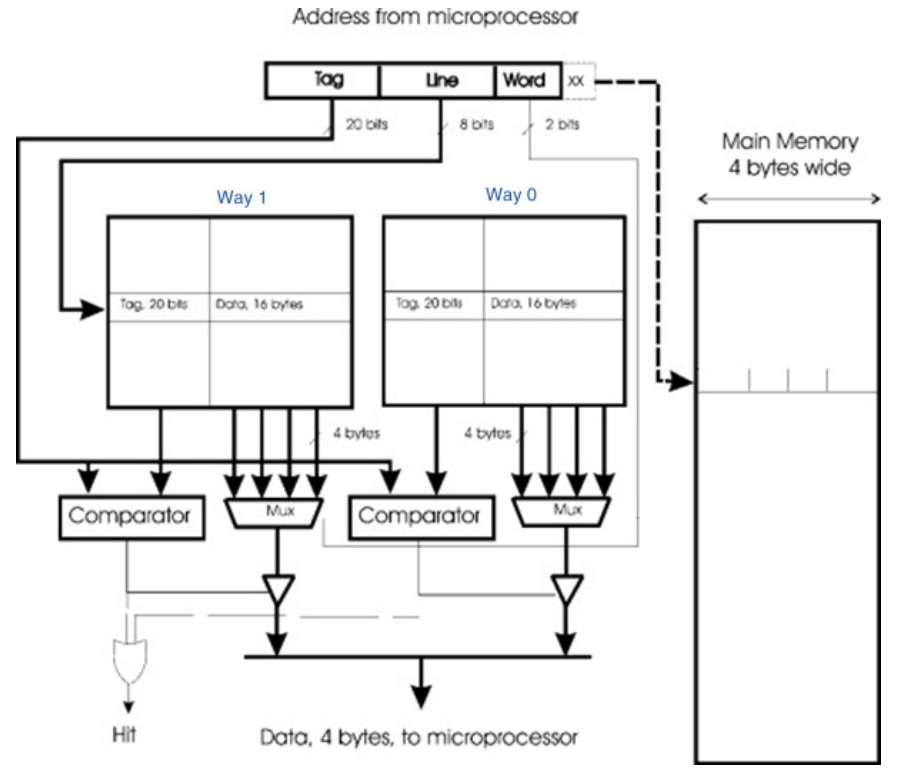
实验地点： 曹西301 实验日期： 2021 年 11 月 2 日

1. 实验目的和要求
2. 理解Cache Line。
3. 理解缓存管理单元 (CMU)和CMU状态机的原理。
4. 掌握CMU的设计方法。
5. 掌握Cache Line的设计方法。
6. 掌握检验Cache Line的方法。
7. 实验内容和原理
8. 实验内容
   1. 设计Cache Line和CMU
   2. 合适Cache Line和CMU
   3. 观察仿真波形
9. 实验原理

2.1. Cache Line



2.2. Cache Mode



1. 实验过程和数据记录

本实验的主要工作是编写cache.v，以下对相关代码进行分析。代码开头给出的地址结构比较重要。

// | ----------- address 32 ----------- |

// | 31   9 | 8     4 | 3    2 | 1    0 |

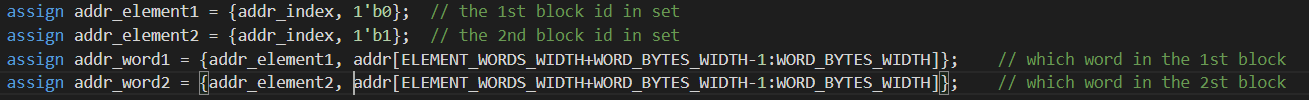
// | tag 23 | index 5 | word 2 | byte 2 |

1. 从地址中截取部分作为tag和index的值

    assign addr\_tag = addr[ADDR\_BITS-1:ADDR\_BITS-TAG\_BITS];

    assign addr\_index = addr[ADDR\_BITS-TAG\_BITS-1:BLOCK\_WIDTH];

2. 补全地址信号



element地址是set地址加上一位way选择，word地址是element地址加上word的偏移量。

3. 补全数据信号

    assign word1 = inner\_data[addr\_word1];

    assign word2 = inner\_data[addr\_word2];

    assign half\_word1 = addr[1] ? word1[31:16] : word1[15:0];

    assign half\_word2 = addr[1] ? word2[31:16] : word2[15:0];

    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]   ;

word的数据是根据地址从数据区直接获取。half\_word的数据是在word中选取高位或者低位，由addr[1]决定。byte是在half\_word的基础上由addr[0]决定是高位还是低位。  
 4. 补充element状态和tag

    assign recent1 = inner\_recent[addr\_element1];

    assign recent2 = inner\_recent[addr\_element2];

    assign valid1 = inner\_valid[addr\_element1];

    assign valid2 = inner\_valid[addr\_element2];

    assign dirty1 = inner\_dirty[addr\_element1];

    assign dirty2 = inner\_dirty[addr\_element2];

    assign tag1 = inner\_tag[addr\_element1];

    assign tag2 = inner\_tag[addr\_element2];

根据element地址，从状态数组中找到对应的状态值，从tag数组中找到对应的tag值。

5. 命中

    assign hit1 = valid1 & (tag1 == addr\_tag);  // if 1st block hit

    assign hit2 = valid2 & (tag2 == addr\_tag);  // if 2nd block hit

当tag一致且为有效状态时说明命中。

1. 部分状态

valid <= (!recent1 & valid1) | (!recent2 & valid2); // recent is the

    dirty <= (!recent1 & dirty1) | (!recent2 & dirty2);

    tag <= ({23{~recent1}} & tag1) | ({23{~recent2}} & tag2);

    hit <= hit1 | hit2;

当一个element近期没有使用过且状态有效时，则状态有效。当一个element近期没有使用过且dirty为1时，dirty赋为1。最近未使用过的element的tag即为tag。任意一个element命中时说明命中。

7. load时命中

else if (hit2) begin

    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[addr\_element2] <= 1'b1;

    inner\_recent[addr\_element1] <= 1'b0;

end

若u\_b\_h\_w[1]为1，则数据为字；若其为0，且u\_b\_h\_w[0]=1&& u\_b\_h\_w[2]=1，则数据为half\_word且无符号拓展，若u\_b\_h\_w[0]=1&& u\_b\_h\_w[2]=0，则数据为half\_word且有符号拓展，若u\_b\_h\_w[0]=0&& u\_b\_h\_w[2]=1，数据为byte且无符号拓展，若u\_b\_h\_w[0]=0&& u\_b\_h\_w[2]=0，数据为byte且有符号拓展。

8. edit时命中

    else if (hit2) begin

        //need to fill in

    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\_element2] <= 1'b1;

    inner\_recent[addr\_element1] <= 1'b0;

end

命中时，写入数据由u\_b\_h\_w[]、addr[]、din决定。

9. store时发现最近使用过

else begin

    inner\_data[addr\_word1] <= din;

    inner\_valid[addr\_element1] <= 1'b1;

    inner\_dirty[addr\_element1] <= 1'b0;

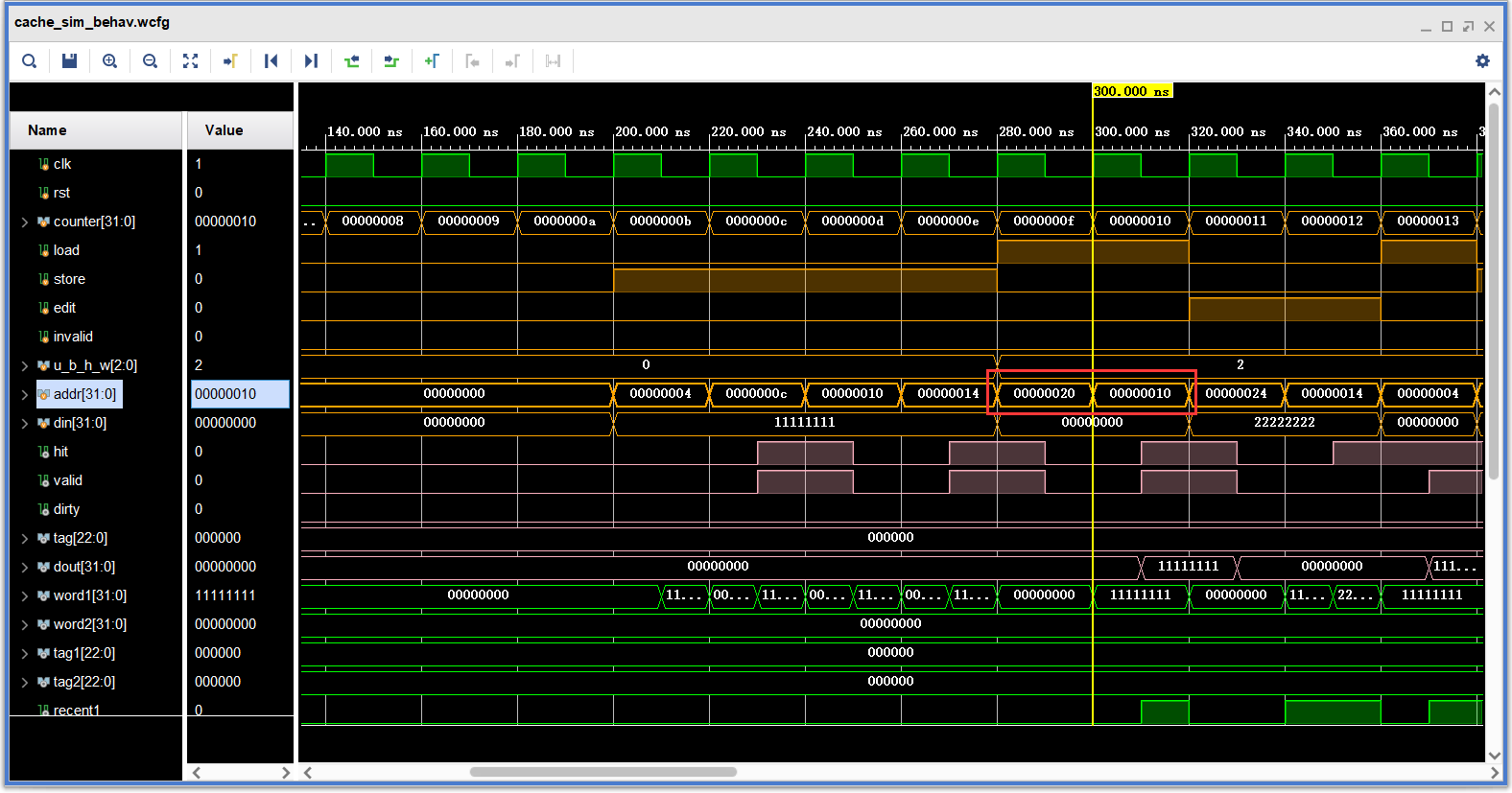
    inner\_tag[addr\_element1] <= addr\_tag;

end

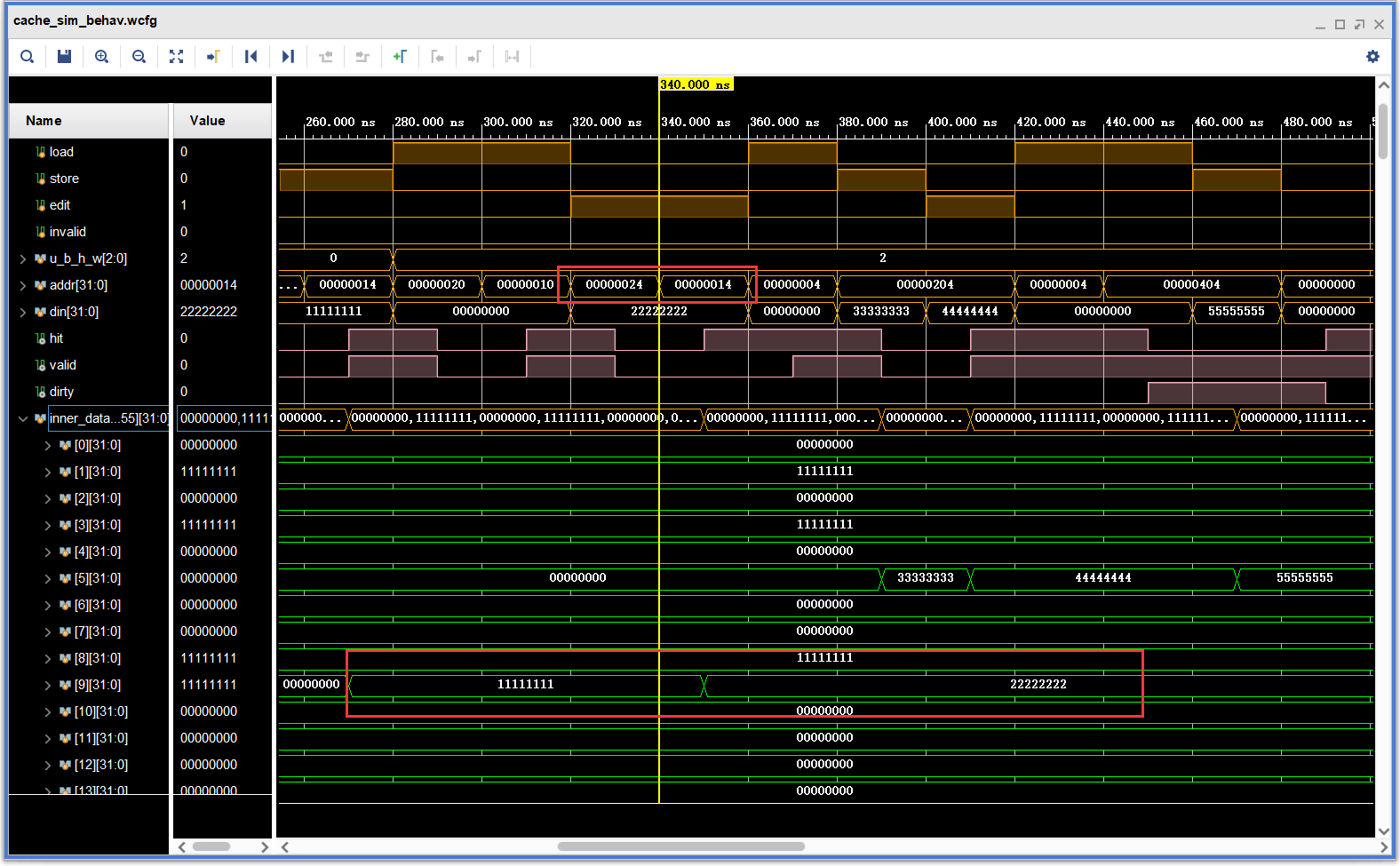
若element1最近使用过，则将数据写入element2的数据区，将其有效位赋1，脏位赋0，tag改为输入的tag。

1. 实验结果分析

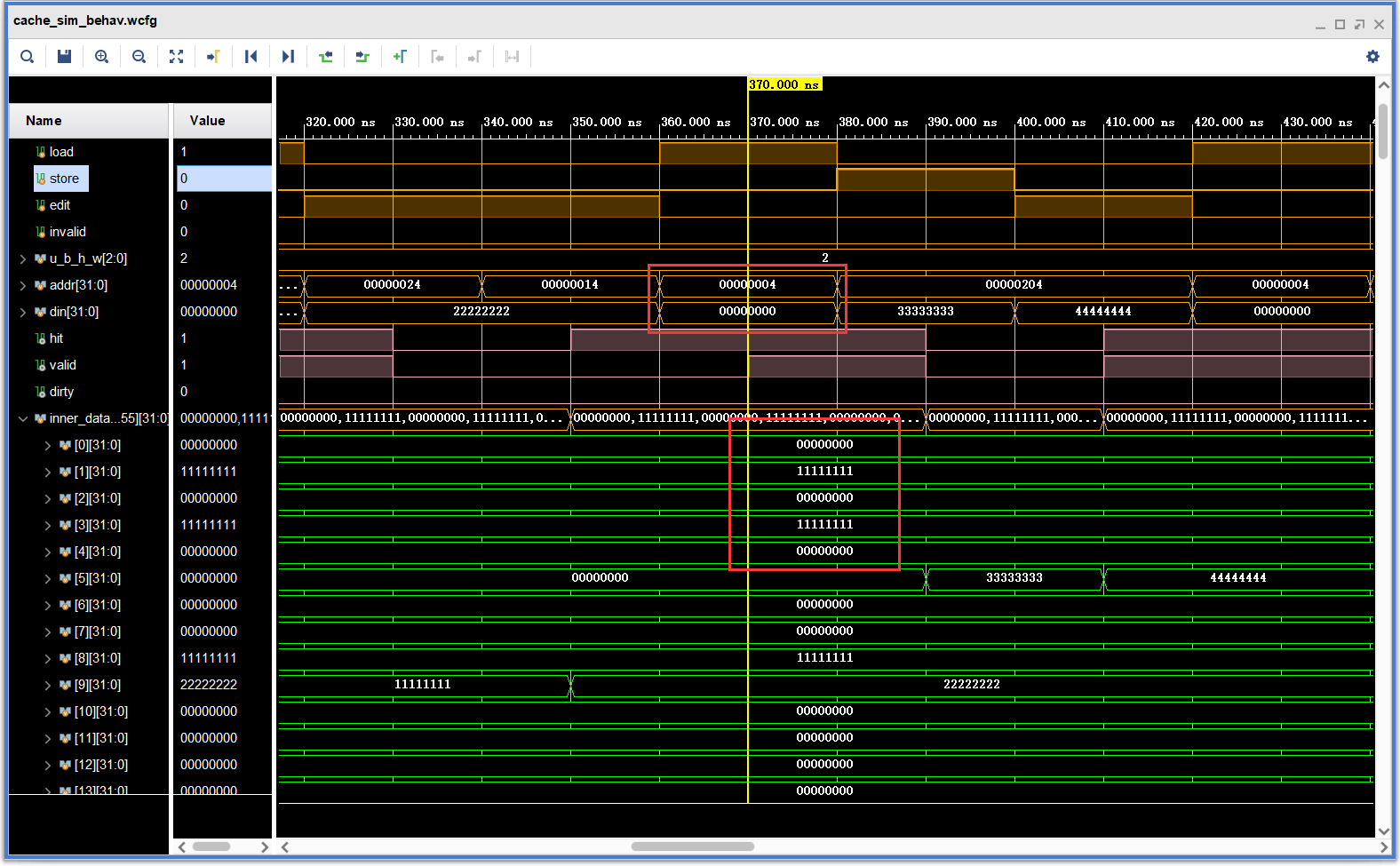
1. read miss and hit



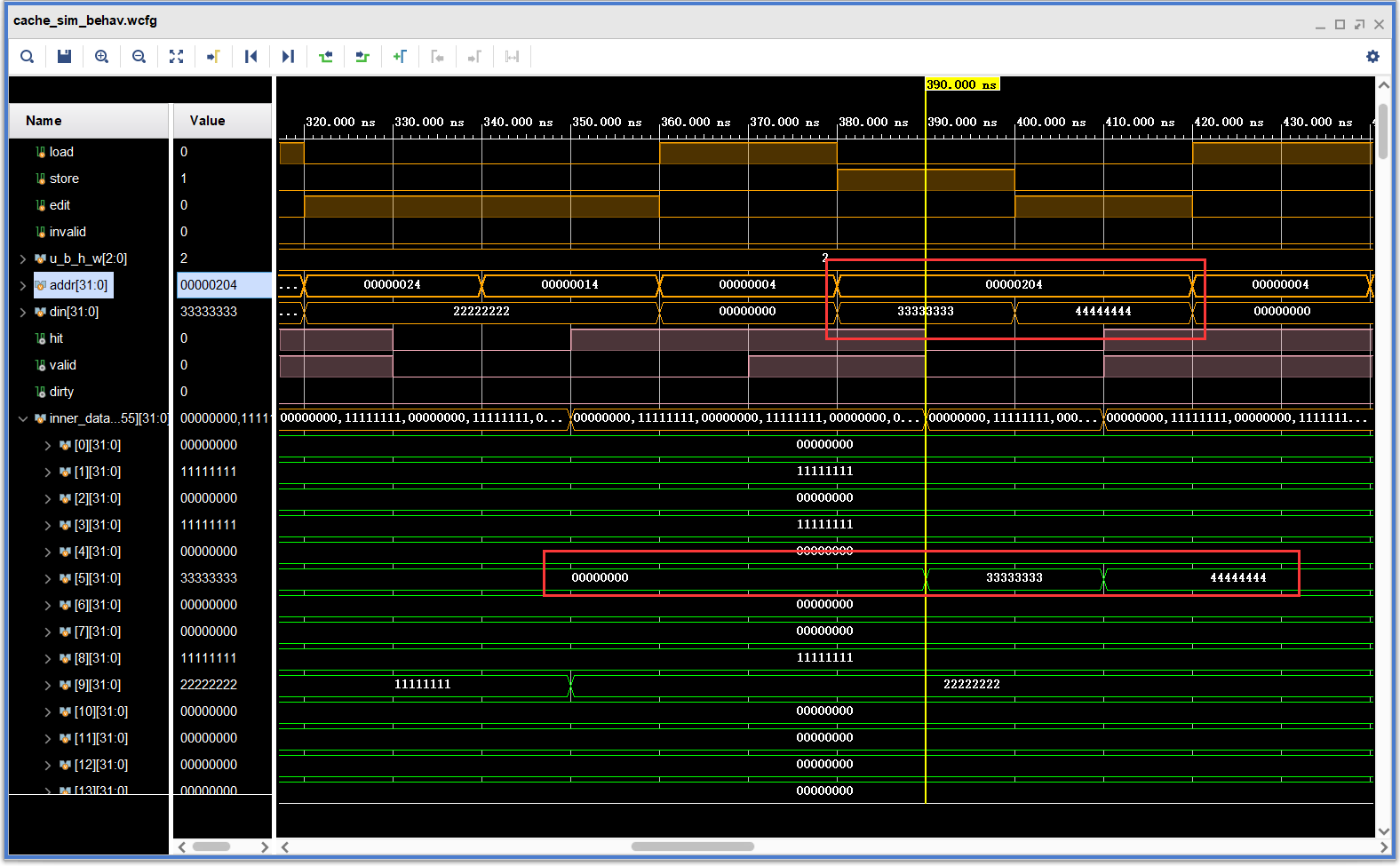
2. write miss and hit



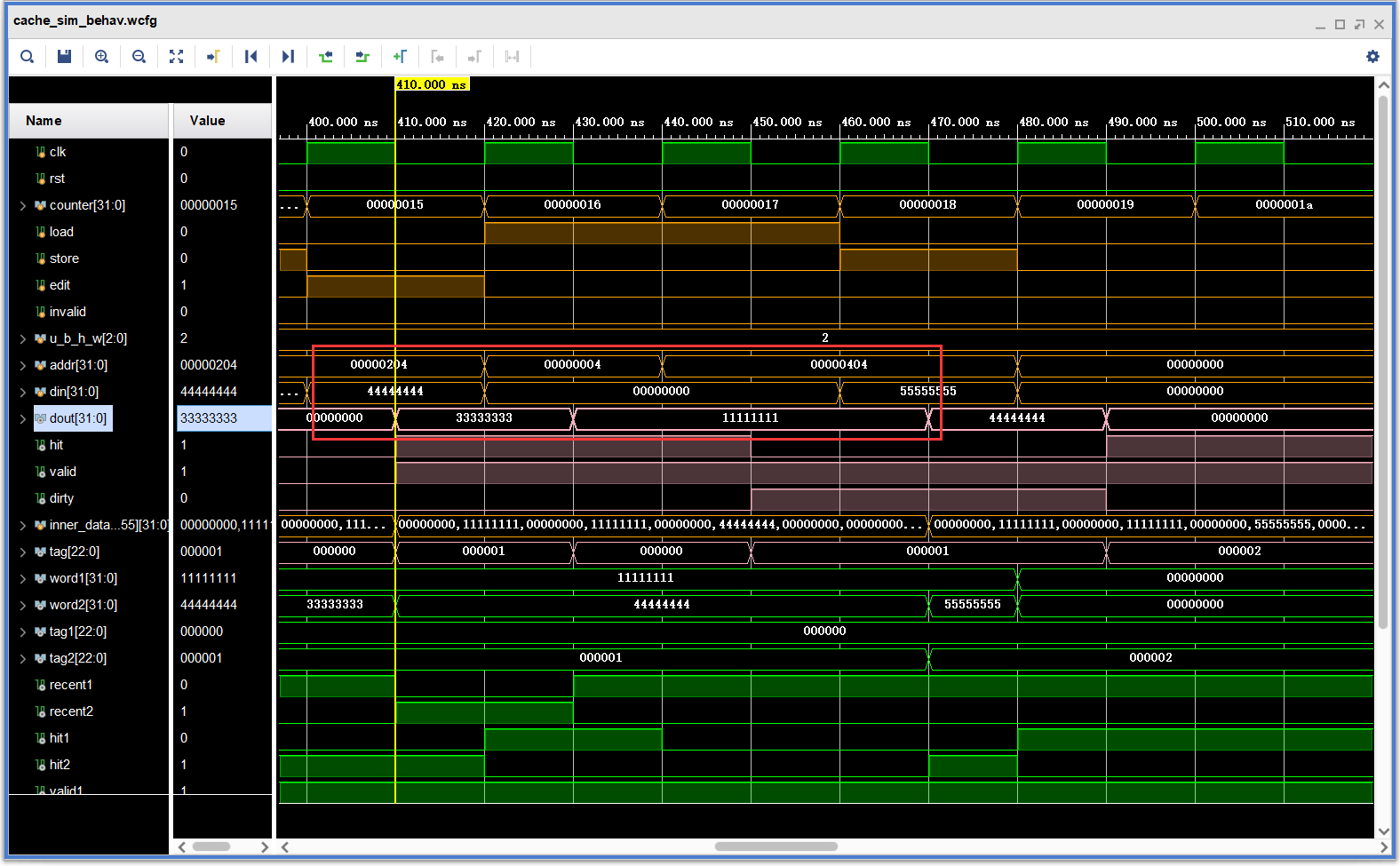
1. read line 0 of set 0, set recent bit



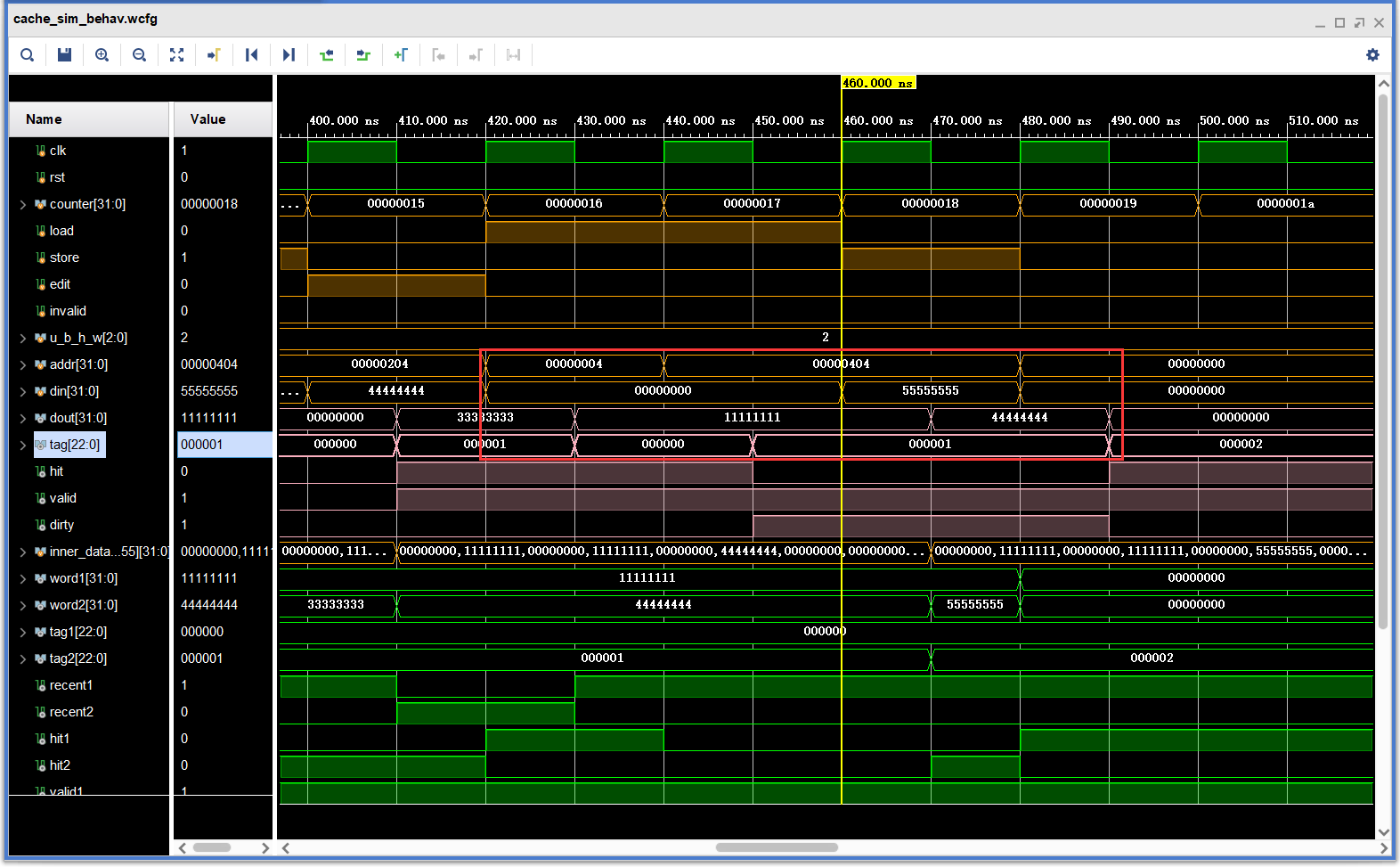
1. store to line 1 of set 0 due to line 0 recent



1. edit line 1 of set 0, set dirty & recent



1. tag mismatch. output tag, valid and dirty == 1



1. 讨论与心得

本实验是Cache的设计，但从实验过程来说，只需要根据老师给出的一半代码进行对应的一点变换即可，但是我们还是需要搞清楚其中的原理，由于实验结构比较简单，所以很多问题都可以直接通过仿真找到原因，直接进行修改，联系课内学到的知识，能够比较容易解决。本次实验的难点在于对于read miss、write miss等各种情况的判断，需要根据每种情况，联系课内知识，给出对应的操作。