浙江大学

本科实验报告

Cache Controller 的设计

课程名称：	计算机组成与设计
姓　　名：	
学　　院：	信息与电子工程学院
专　　业：	信息工程
学　　号：	
指导老师：	沈继忠、赵武锋

2023 年 6 月 4 日

浙江大学实验报告

专业：信息工程
姓名：
学号：
日期：2023 年 6 月 4 日
地点：

课程名称：计算机组成与设计　指导老师：沈继忠、赵武锋　成绩：
实验名称：Cache Controller 的设计　实验类型：设计仿真　同组学生姓名：

一、 实验任务

使用 Verilog HDL 设计一个一级数据缓存控制器（first-level data cache controller）。

二、 设计思路

1. 状态编码

参考教材 5.9 节，可以设计四个状态，分别为：

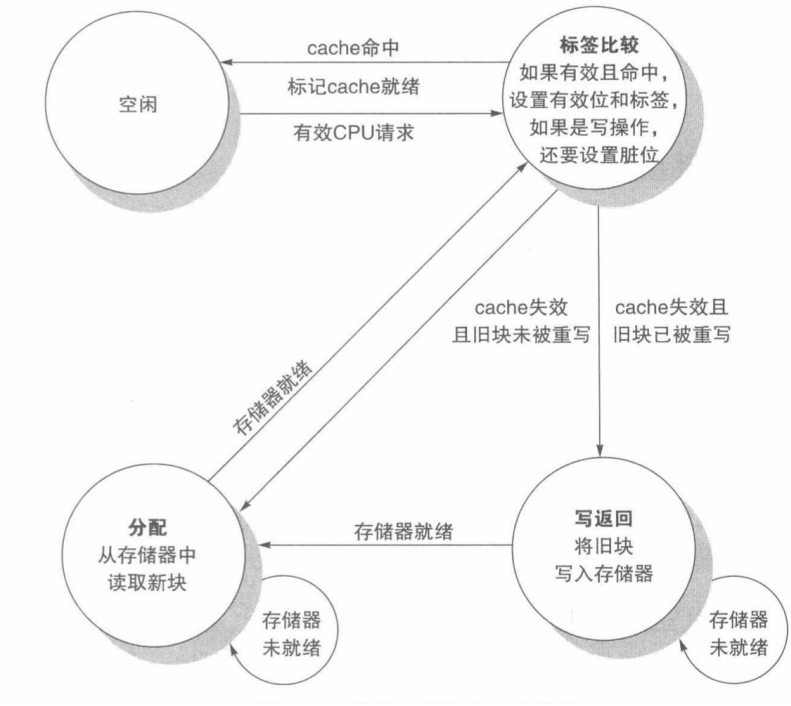

图 1: 控制器的四个状态

Idle：该状态等待处理器发出的有效读或写信号，之后有限状态自动机跳转到标签比较状态。

CompareTag：正如名称所示，该状态检测读或写请求是命中还是失效。地址的索引部分选择用于比较的标签。如果地址中的索引部分引用的 cache 块中的数据是有效的，并且地址中的标签部分与标签相匹配，则命中。

如果是加载指令就从选择的字中读取数据，如果是存储指令就将数据写人选择的字中。之后设置 cache 就绪信号。如果这是个写操作，脏位还要设置为 1。需要注意的是，写命中也需要设置有效位和标签字段，即使这看上去并不需要。这是因为标签使用单独的存储器，因此在改变脏位时也需要同时改变有效位和标签字段。如果发生命中并且当前块是有效的，有限状态自动机会返回空闲状态。失效时先更新 cache 标签，之后如果当前块的脏位为 1，则跳转到写回状态，如果该位为 0，则跳转到分配状态。

WriteBack：该状态使用由标签和 cache 索引组成的地址将 128 位的块写回存储器。之后继续停留在该状态等待存储器发出就绪信号。等待存储器写操作完成后，有限状态自动机跳转到分配状态。

Allocate：从存储器中取出一个新块。之后继续停留在该状态等待存储器发出就绪信号。等待存储器读操作完成后，有限状态自动机跳转到标签比较状态。尽管我们可以不重新使用标签比较状态而跳转到一个新的状态完成操作，但是分配状态之后的操作与标签比较状态的操作有大量的重香，包括当访问为写操作时更新块中相应的字。

并对四个状态状态编码为 Idle = 0 , CompareTag = 1 , WriteBack = 2 , Allocate = 3。

2. 状态转换表

写出状态转移表如下表，“-”表示在当前状态转移情况下不关心该信号。

state	ld	st	addr[31:11] == tag	valid	dirty	l2_ack	write_done	nextstate
Idle	0	0	-	-	-	-	-	Idle
	0	1	-	-		-	-	CompareTag
	1	0	-	-		-	-	
	1	1	-	-		-	-	
CompareTag	-	-	-	0	0	-	-	Allocate
	-	-	-	0	1	-	-	WriteBack
	-	-	-	1		-	-	Idle
WriteBack	-	-	-	-	-	-	0	WriteBack
	-	-	-	-	-	-	1	Allocate
Allocate	-	-	-	-	-	0	-	Allocate
	-	-	-	-	-	1	-	CompareTag

3. FSM 流程图

利用 drawio 绘制出 FSM 流程图如图所示。

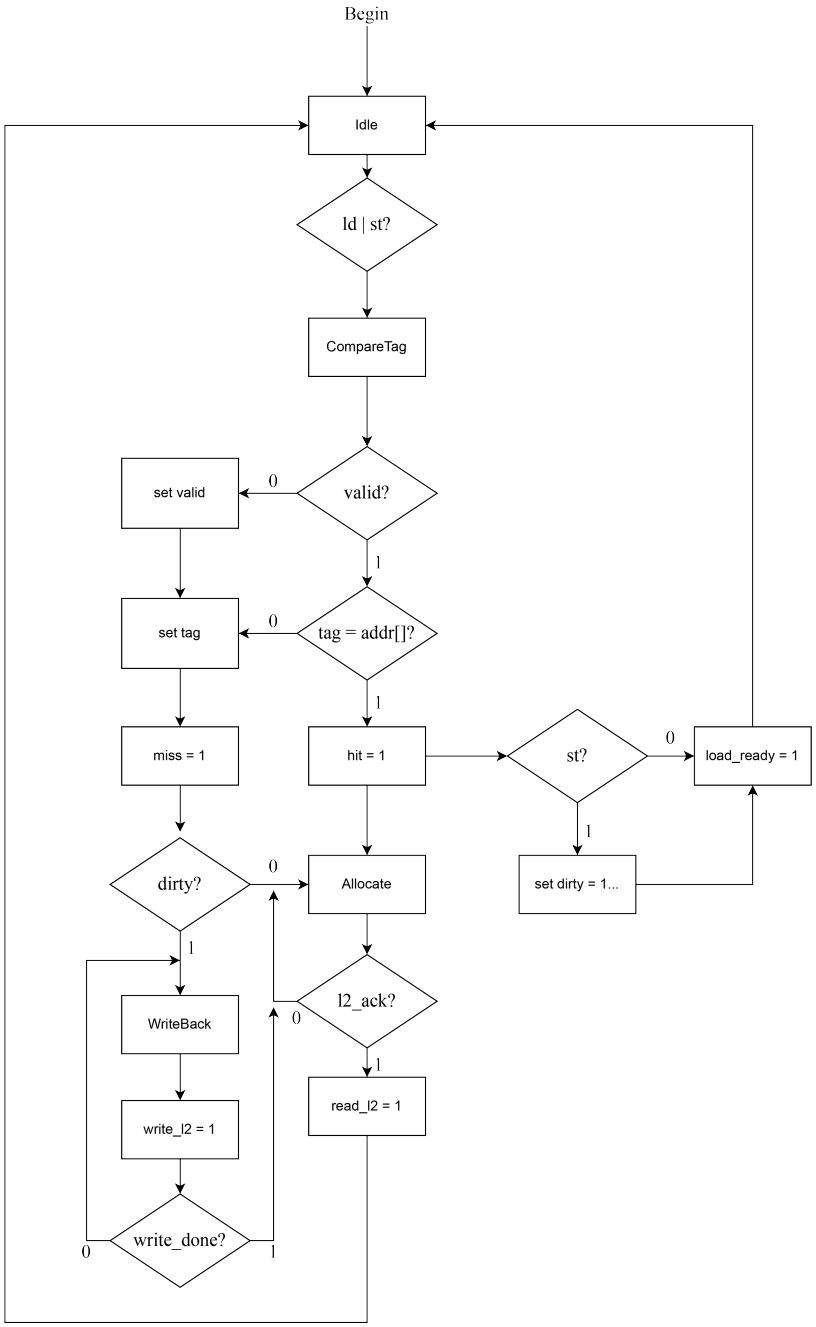

图 2: FSM 流程图

三、 电路设计

通过 vivado 软件进行电路设计，如下图

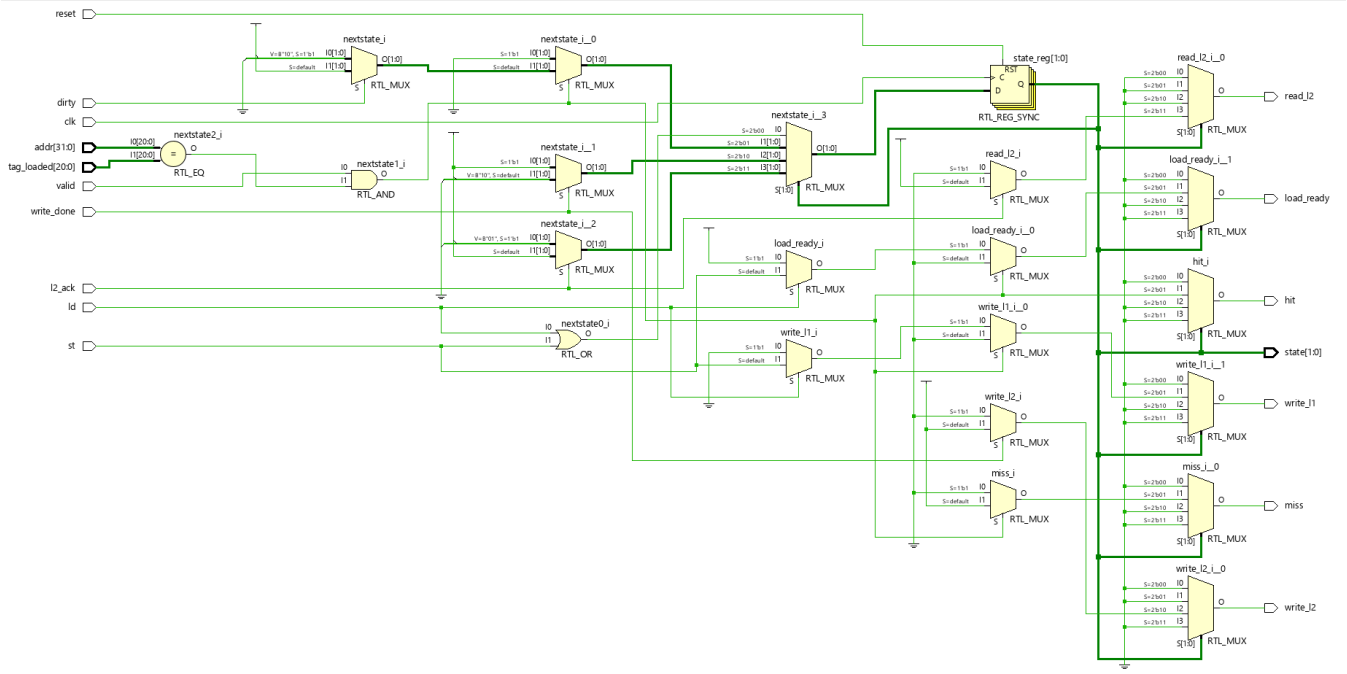

图 3: 电路图

四、 主要仪器设备

装有 vivado 的计算机，

五、 Verilog 设计与仿真

1. 代码实现

1.1 cache 控制器代码

```
module CacheController(
    input clk,              \\ 时钟信号
    input reset,            \\ 复位信号
    input ld,               \\ 加载请求
    input st,               \\ 储存请求
    input [31:0] addr,      \\ cache访问地址
    input [20:0] tag_loaded, \\ cache中存放的date地址
    input valid,            \\ 有效位
    input dirty,            \\ 脏值位
    input l2_ack,           \\ 从L2加载的数据已到达
    input write_done,       \\ 写入L2的数据已完成
    output reg hit,         \\ 是否命中
    output reg miss,        \\ 是否失效
    output reg load_ready,  \\ 数据已成功加载
    output reg write_l1,    \\ 启用L1高速缓存写入
    output reg read_l2,     \\ 加载请求到L2高速缓存
```

```
    output reg write_l2,    \\ 启用回写缓冲区
    output state            \\ 状态机状态
);

    // 状态编码
    parameter Idle = 0, CompareTag = 1, WriteBack = 2, Allocate = 3;
    reg [1:0] state, nextstate;

    // 第一段
    always @(posedge clk) begin
        if(reset) state = Idle;
        else state = nextstate;
    end

    // 第二段
    always @(*) begin
        hit = 0 ; miss = 0;
        load_ready = 0 ; write_l1 = 0;
        write_l2 = 0 ;read_l2 = 0;
        case(state)
            Idle: begin
                if(ld | st) begin
                    nextstate = CompareTag;
                end
                else nextstate = Idle;
            end

            CompareTag: begin
                if(valid && addr[31:11] == tag_loaded) begin
                    hit = 1;
                    miss = 0;
                    if(ld) begin
                        load_ready = 1;
                    end
                    else if(st) begin
                        load_ready = 1;
                        write_l1 = 1;
                    end
                    nextstate = Idle;
                end
                else begin
                    miss = 1;
                    hit = 0;
                    if(dirty) begin
                        nextstate = WriteBack;
                    end
                    else nextstate = Allocate;
                end
            end

```

```
            WriteBack: begin
                write_l2 = 1;
                if(write_done) begin
                    write_l2 = 0;
                    nextstate = Allocate;
                end
                else begin
                    write_l2 = 1;
                    nextstate = WriteBack;
                end
            end

            Allocate: begin
                read_l2 = 1;
                if(l2_ack) begin
                    read_l2 = 0;
                    nextstate = CompareTag;
                end
                else begin
                    read_l2 = 1;
                    nextstate = Allocate;
                end
            end
        endcase
    end
endmodule
```

1.2 仿真测试代码

‘timescale 1ns / 1ps

```
module testbench;
    parameter delay = 100;
    //inputs
    reg clk,reset;
    reg ld , st;
    reg [31:0] addr;
    reg valid , dirty;
    reg [20:0] tag_loaded;
    reg l2_ack;
    reg write_done;

    //outputs
    wire hit , miss;
    wire load_ready , write_l1;
    wire write_l2 , read_l2;
    wire [1:0]state;

    //instantiation of the cache controller
    CacheController cache(
```

```
20          .clk(clk),
21          .reset(reset),
22          .ld(ld),
23          .st(st),
24          .addr(addr),
25          .valid(valid),
26          .dirty(dirty),
27          .tag_loaded(tag_loaded),
28          .l2_ack(l2_ack),
29          .hit(hit),
30          .miss(miss),
31          .load_ready(load_ready),
32          .write_l1(write_l1),
33          .write_l2(write_l2),
34          .read_l2(read_l2),
35          .write_done(write_done),
36          .state(state)
37      );
38
39      //set the free running clock
40      initial begin
41          clk = 1;
42          forever begin
43              #(delay/2) clk = ~clk;
44          end
45      end
46
47      //set up the reset signal
48      initial begin
49          reset = 1 ;
50          #(delay*2) reset = 0;
51      end
52
53      initial begin
54          //initial input
55          ld = 0;
56      st = 0;
57      addr = 32'b0000_0000_0000_0000_0000_0000_0000_0000;
58      tag_loaded = 21'b0000_0000_0000_0000_0000_0;
59      valid = 0;
60      dirty = 0;
61      l2_ack = 0;
62          write_done = 0;
63
64          //read hit
65          #(delay*2)
66              ld = 1 ; st = 0;
67              addr = 32'b0000_0000_0000_0000_0001_1000_0000_0000;
68              tag_loaded = 21'b0000_0000_0000_0000_0001_1;
69              valid = 1;
```

```
70          //wait for next instruction
71          #delay
72              ld = 0 ; st = 0;
73          //write hit
74          #delay
75              ld = 0 ; st = 1;
76          //wait for next instruction
77          #delay
78              ld = 0 ; st = 0;
79
80          //read compulsory miss
81          #(delay)
82              ld = 1 ; st = 0;
83              valid = 0;
84          //wait for the l2_ack
85          #(delay*3)
86              l2_ack = 1;
87              valid = 1;
88          //wait for next instruction
89          #delay
90              ld = 0 ; st = 0;
91
92          //write compulsory miss
93          #(delay)
94              ld = 0 ; st = 1;
95              valid = 0;
96              l2_ack = 0;
97          //wait for the l2_ack
98          #(delay*4)
99              l2_ack = 1;
100             valid = 1;
101         //wait for next instruction
102         #delay
103             ld = 0 ; st = 0;
104             l2_ack = 0;
105
106         //conflict miss with dirty=0
107         #(delay)
108             ld = 1 ; st = 0;
109             tag_loaded = 21'b0000_0000_0000_0000_0001_0;
110             valid = 1;
111             dirty = 0;
112         //wait for the l2_ack
113         #(delay*3)
114             l2_ack = 1;
115             tag_loaded = 21'b0000_0000_0000_0000_0001_1;
116         #delay
117             ld = 0 ; st = 0;
118             l2_ack=0;
119
```

```
120         //conflict miss with dirty=1
121         #(delay)
122             ld = 0 ; st = 1;
123             tag_loaded = 21'b0000_0000_0000_0000_0001_0;
124             valid = 1;
125             dirty = 1;
126         //wait for the write_done signal
127         #(delay)
128             write_done = 1;
129         #(delay*3)
130             write_done = 0;
131             l2_ack = 1;
132             tag_loaded = 21'b0000_0000_0000_0000_0001_1;
133         #(delay*2)
134             ld = 0 ;st = 0;
135             l2_ack=0;
136 
137         #(delay*3)
138 
139             $stop;
140         end
141 endmodule
```

2. 测试结果

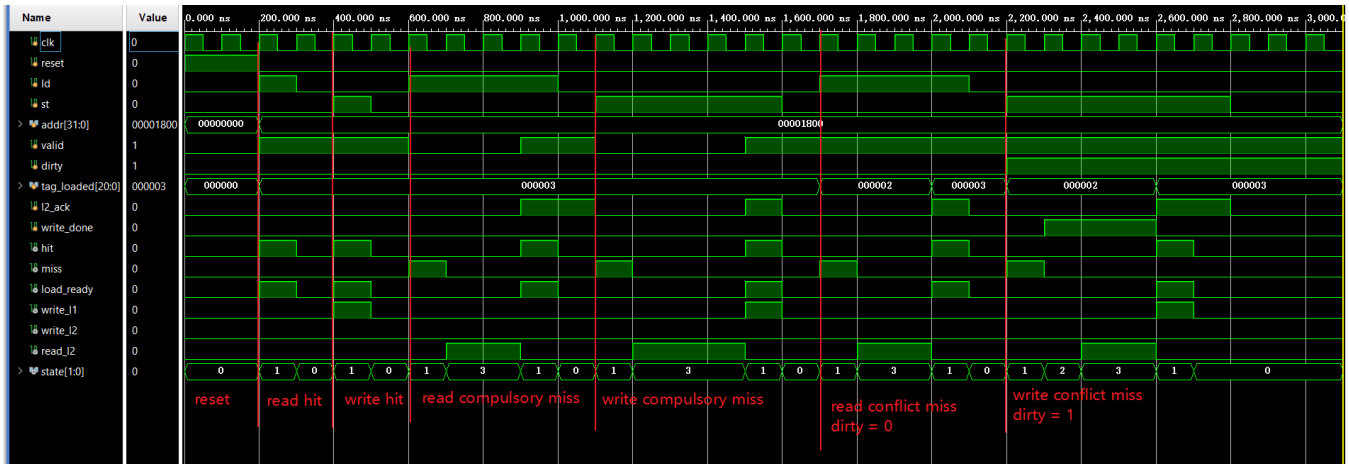

图 4: 测试结果

观察测试结果仿真图，可以看出控制信号正确。

六、 心得体会

通过本次实验，我对 cache 有了更加深入的理解。同时本次实验的测试文件是自己编写的，是自己第一次编写，也学会了许多。