

浙江大学

本科实验报告

Cache Controller

课程名称： 计算机组成与设计

姓 名： 周灿松

学 院： 信息与工程学院

系：

专 业： 信息工程

学 号： 3190105055

指导老师： 刘鹏

2021 年 12 月 24 日

目录

一 有限状态机设计	3
1. 状态编码设计	3
2. 状态转换表	4
3. FSM 图	4
二 电路设计	5
三 Verilog 实现	5
1. 信号含义详细阐释	5
2. Verilog 代码	6
3. 测试样例	6
4. 测试结果	6
四 心得体会	6
五 附录	6
1. cache 代码	6
2. 测试代码	8

一 有限状态机设计

1. 状态编码设计

如图 1 所示, 我参考教材 5.9 节设计了一共四个状态, 分别是: 空闲状态 IDLE, Tag 位比较状态 CompareTag, 写回状态 WriteBack 以及分配状态 Allocate, 各个状态作用如下

- (1) IDLE: 等待来自 CPU 的控制信号 ld 以及 st, 当这两个信号有效时, 转移到 CompareTag 状态。
- (2) CompareTag: 比较来自 Cache 的对应 Block 的 Tag 位与来自 CPU 的地址 Addr 的 [31:11] 位, 同时根据对应 Block 的 Valid 与 Dirty 位判断是 Hit 或者 Miss, 以及转移到对应的状态。
- (3) WriteBack: 在此状态将 Cache 中的 Block 写回 L2 Cache 中, 当接收到来自 L2 的写回完成信号 write_done 后, 写回完成, 状态转移到 Allocate。
- (4) Allocate: 此状态用于在 L1 Cache miss 时将 L2 Cache 中的值加载到 L1 Cache 中, 当得到写入完成信号 l2_ack 后, 说明写入完成, 状态切换到 CompareTag。

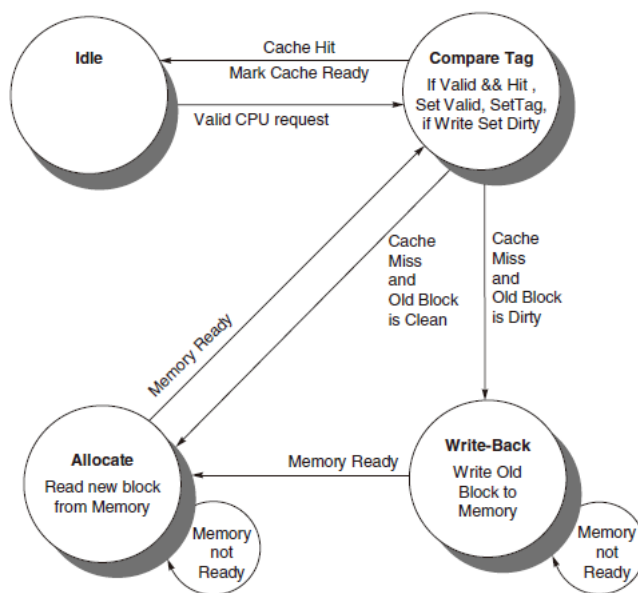


图 1: state diagram

各个状态的状态编码如表 1 所示

状态编码	
IDLE	00
CompareTag	01
WriteBack	10
Allocate	11

表 1: 状态编码

2. 状态转换表

状态转移表如表 2所示, ‘-’ 表示在当前状态转移情况下不关心该信号。

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

表 2: 状态转移表

3. FSM 图

FSM 图如图 2所示。

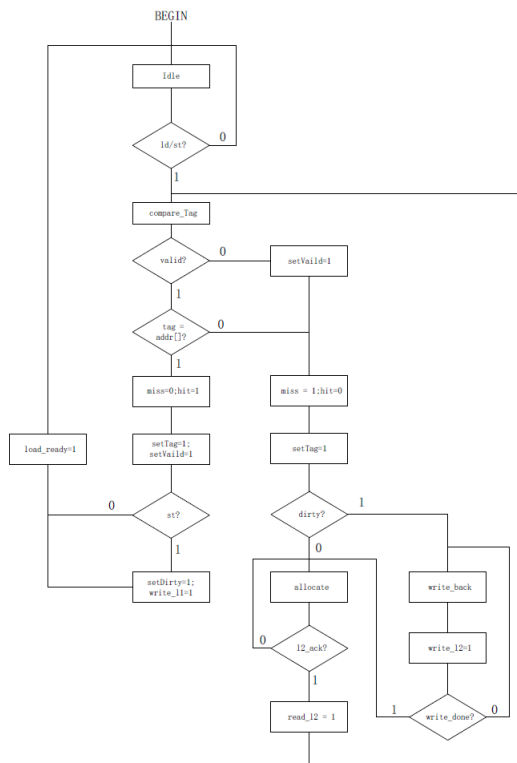


图 2: FSM 图

二 电路设计

我通过 Vivado 软件进行了电路设计，电路图如图 3 所示。

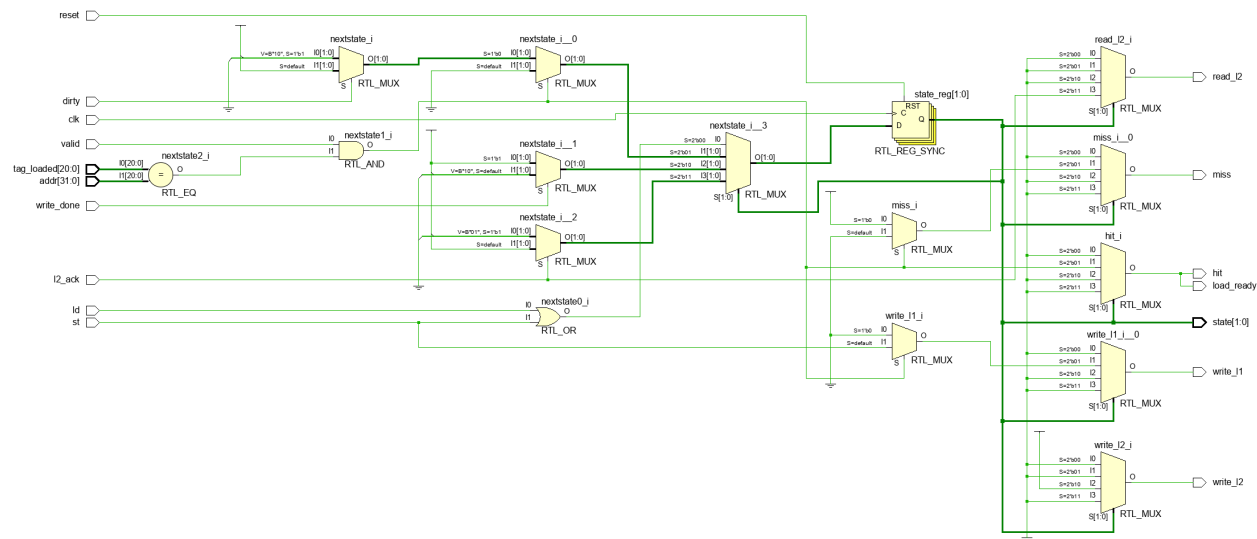


图 3: 电路图

三 Verilog 实现

1. 信号含义详细阐释

- (1) clk: 时钟信号
- (2) reset: 重置信号
- (3) ld: 数据由 L1 加载到 CPU
- (4) st: 数据由 CPU 写入 L1
- (5) addr: 数据所在地址
- (6) valid:L1 的有效位
- (7) dirty:L1 的脏位
- (8) tag_loaded:L1 对应位置的 Tag
- (9) l2_ack: 数据是否完成从 L2 写入 L1 的过程
- (10) write_done: 数据是否完成从 L1 写回 buffer 的过程
- (11) hit:hit 信号
- (12) miss:miss 信号

(13) load_ready: L1 数据是否准备就绪

(14) write_l1: 使能 L1 的写入功能

(15) read_l2: 发向 L2 的 load 指令

(16) write_l2: 使能 L1 写回 buffer

2. Verilog 代码

Cache 控制器的代码 Listing 1所示。

3. 测试样例

所用测试代码如 Listing 2所示。

4. 测试结果

Cache Controller 测试结果如图 4所示，各个控制信号正确，功能能够正常实现



图 4: 测试结果

四 心得体会

通过本次实验，我对 Cache 的整个控制机制有了一个更加具体的理解，同时也补足了自己在学习 Cache 相关内容时所遗漏的知识点。

除此之外，这次实验也是我第一次编写 testbench 文件，对于 Verilog 语言的测试方式有了一个更加透彻的掌握。

五 附录

1. cache 代码

Listing 1: Cache 控制器代码

```
1  module cache_controller (
    input clk,reset,
3   input ld , st,
    input [31:0] addr,
5   input valid , dirty,
    input [20:0] tag_loaded,
7   input l2_ack, write_done,
    output reg hit , miss,
9   output reg load_ready , write_l1,
    output reg write_l2 , read_l2,
11  output state
);
13  parameter IDLE = 0 , CompareTag = 1 , WriteBack = 2 , Allocate = 3;
    reg [1:0] state , nextstate;
15
    always @(posedge clk) begin
17         if(reset) state = IDLE;
        else state = nextstate;
19    end

21    always @(*) begin
        hit = 0 ; miss = 0;
23        load_ready = 0 ; write_l1 = 0;
        write_l2 = 0 ; read_l2 = 0;
25        case (state)
            IDLE : begin
27                if(ld || st) begin
                    nextstate = CompareTag;
29                end
                else nextstate = IDLE;
31            end

33            CompareTag : begin
                if(!(valid && tag_loaded == addr[31:11]))begin
35                    miss = 1 ;
                    if(dirty)begin
```

```
37         nextstate = WriteBack;
           end
39         else begin
           nextstate = Allocate;
41         end
       end
43     else begin
       if(st)begin
45         hit = 1 ;
         load_ready = 1 ; write_l1 = 1;
47       end
       else begin
49         hit = 1 ; miss = 0;
         load_ready = 1;write_l1 = 0;
51         write_l2 = 0 ; read_l2 = 0;
       end
53     nextstate = IDLE;
       end
55   end

57   WriteBack : begin
       write_l2 = 1;
59   if(write_done)begin
       nextstate = Allocate;
61   end
       else begin
63     nextstate = WriteBack;
       end
65   end

67   Allocate : begin
       if(l2_ack)begin
69     read_l2=1;
       nextstate = CompareTag;
71   end
       else begin
73     nextstate = Allocate;
       end
75   end
       endcase
77   end
endmodule
```


2. 测试代码

Listing 2: Cache 控制器测试文件

```
`timescale 1ns / 1ps
2
module testbench;
4     parameter delay = 100;
    //inputs
6     reg clk,reset;
    reg ld , st;
8     reg [31:0] addr;
    reg valid , dirty;
10    reg [20:0] tag_loaded;
    reg l2_ack;
12    reg write_done;

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

20    //instantiation of the cache controller
    cache_controller cache(
22        .clk(clk),
        .reset(reset),
24        .ld(ld),
        .st(st),
26        .addr(addr),
        .valid(valid),
28        .dirty(dirty),
        .tag_loaded(tag_loaded),
30        .l2_ack(l2_ack),
        .hit(hit),
32        .miss(miss),
        .load_ready(load_ready),
34        .write_l1(write_l1),
        .write_l2(write_l2),
36        .read_l2(read_l2),
        .write_done(write_done),
38        .state(state)
```

```
);
40
    //set the free running clock
42    initial begin
        clk = 1;
44        forever begin
            #(delay/2) clk = ~clk;
46        end
    end
48
    //set up the reset signal
50    initial begin
        reset = 1 ;
52        #(delay*2) reset = 0;
    end
54
    initial begin
56        //initial input
        ld = 0;
58        st = 0;
        addr = 32'b0000_0000_0000_0000_0000_0000_0000_0000;
60        tag_loaded = 21'b0000_0000_0000_0000_0000_0;
        valid = 0;
62        dirty = 0;
        l2_ack = 0;
64        write_done = 0;

66
        //read hit
68        #(delay*2)
            ld = 1 ; st = 0;
70            addr = 32'b0000_0000_0000_0000_0101_1000_0000_0000;
            tag_loaded = 21'b0000_0000_0000_0000_0101_1;
72            valid = 1;
        //wait for next instruction
74        #delay
            ld = 0 ; st = 0;
76        //write hit
        #delay
78            ld = 0 ; st = 1;
        //wait for next instruction
80        #delay
            ld = 0 ; st = 0;
```

```
82
    //read compulsory miss
84    #(delay)
        ld = 1 ; st = 0;
86        valid = 0;
    //wait for the l2_ack
88    #(delay*3)
        l2_ack = 1;
90        valid = 1;
    //wait for next instruction
92    #delay
        ld = 0 ; st = 0;
94
    //write compulsory miss
96    #(delay)
        ld = 0 ; st = 1;
98        valid = 0;
        l2_ack = 0;
100    //wait for the l2_ack
    #(delay*4)
102        l2_ack = 1;
        valid = 1;
104    //wait for next instruction
    #delay
106        ld = 0 ; st = 0;
        l2_ack = 0;
108
    //conflict miss with dirty=0
110    #(delay)
        ld = 1 ; st = 0;
112        tag_loaded = 21'b0000_0000_0000_0000_0101_0;
        valid = 1;
114    //wait for the l2_ack
    #(delay*3)
116        l2_ack = 1;
        tag_loaded = 21'b0000_0000_0000_0000_0101_1;
118    #delay
        ld = 0 ; st = 0;
120        l2_ack=0;

    //conflict miss with dirty=1
122    #(delay)
124        ld = 0 ; st = 1;
```

```
        tag_loaded = 21'b0000_0000_0000_0000_0101_0;
126        dirty = 1;
        //wait for the write_done signal
128        #(delay)
            write_done = 1;
130        #(delay*3)
            write_done = 0;
132        l2_ack = 1;
            tag_loaded = 21'b0000_0000_0000_0000_0101_1;
134        #(delay*2)
            ld = 0 ;st = 0;
136            l2_ack=0;

138        #(delay*3)

140            $stop;

142        end

144    endmodule
```
