实验 15 报告

学号：2017K8009922027 2017K8009929011
姓名：张磊、郭豪
箱子号：39

一、实验任务（10%）

按照课程讲义规定的接口设计 Cache 模块，通过简单的读写测试。

二、实验设计（40%）

（一）总体设计思路

1. Cache 采用两路组相连的方式实现，每一路大小为 4KB，cache 行的大小为 16 字节，对于 32 位地址，tag(20), index(8),offset(4)的划分如下图。

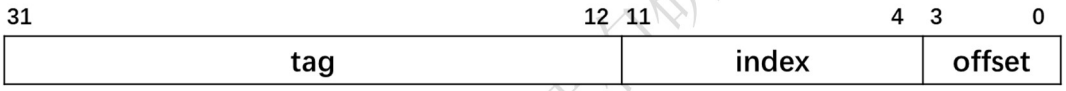

图 1 32 位地址划分

2. Tag 和 valid 使用 21 位，256 项的 ram 实现，高 20 位表示 tag,最低位为 valid；每一路的 dirty 位使用 regfile 的方式实现; cache 行通过添加 4 个 256 项，每一项 32 位的 ram 实现。
3. Cache 设置中采用一个五态状态机的方式实现，状态机的转移如图 2 所示，各状态的的功能如下。
   a) IDLE 状态表示当前 cache 没有任何操作，此时如果有 valid 的请求传来，则保存请求的相关信息，反馈 addr_ok 后进入 LOOKUP 状态。
   b) 在 LOOKUP 状态，有三种情况。若没有命中，则进入 MISS 状态，并发出对应 cache 行写回内存的请求；若命中了且没有新的请求，则执行相应的读/写请求，并返回到 IDLE 状态；若命中了且有新的请求到来，则保存新请求的数据，反馈 addr_ok 并保留在 LOOKUP 状态继续执行。
   c) MISS 状态主要负责将 cache 行写入主存中，当写反馈 wr_rdy 信号反馈回来后转入 REPLACE 状态。
   d) REPLACE 状态负责将访问内存读取相应 cache 行的内容，当 ret_last 信号到来时，说明 axi 总线的读取结束，可以跳转到 REFILL 阶段。
   e) REFILL 阶段主要负责将总线中读取的数据写入相应的 cache 行，若发生 cache 缺失的是写操作则直接修改写入 cache 行的数据并标记上 dirty。

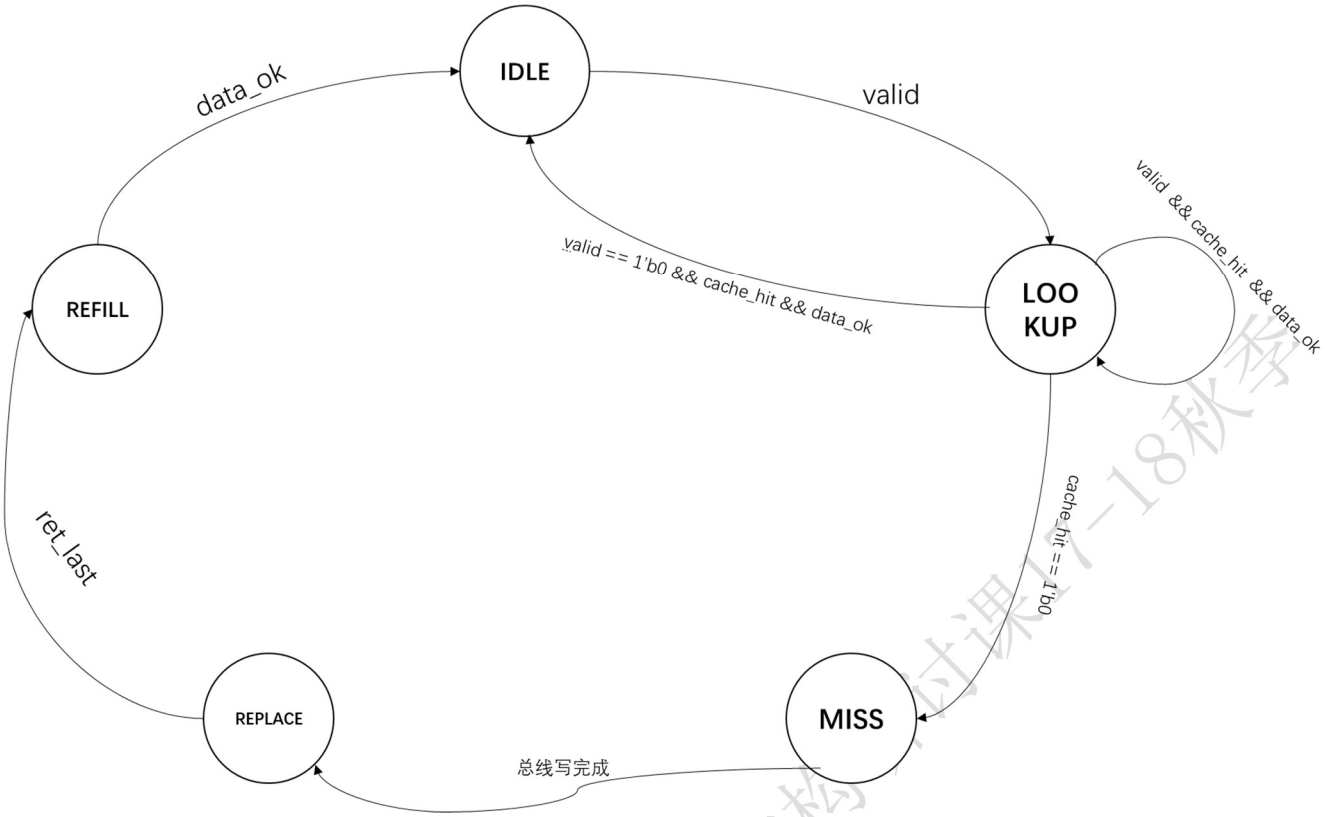

图 2 cache 状态机状态转移图

4. Addr_ok 主要在两种情况下拉高，一种是 IDLE 阶段成功保存了请求的相关数据后，另一种是 LOOKUP 阶段保存相关数据后，向 cpu 反馈 addr_ok 信号。
5. Data_ok 代表着读/写操作的结束。对于读操作，命中当拍即可拉高，对于写操作，当写完成后下一拍拉高 data_ok 信号。若发生 cache miss 时，只有等到 REFILL 阶段，cache 的写入信号拉高后下一拍（写入完成）再拉高 data_ok 信号。

（二） 重要模块 1 设计：cache 状态转移模块

1. 工作原理

具体的状态转移条件和每个状态完成的动作已在实验设计中给出，故这里只贴出代码部分。

```
always @(posedge clk)
begin
    if(resetn == 0)begin
        cstate <= IDLE;
        index_tra <= 1'b0;
    end
    else if(cstate == IDLE) begin
        if(valid) begin
            cstate <= LOOKUP;
        end
    end
```

```
    else if(cstate == LOOKUP) begin
        if( cache_hit == 1'b0) begin
            cstate <= MISS;
            index_tra <= cnt;
        end
        else if(valid == 1'b0 && cache_hit == 1'b1 && data_ok == 1) begin
            cstate <= IDLE;
        end
        else if(valid == 1'b1 && cache_hit == 1'b1 && data_ok == 1) begin
            cstate <= LOOKUP;
        end
    end
    else if(cstate == MISS) begin
        //cache write back to memory
        if(cache_D0[index_r] && way0_valid && index_tra == 0) begin
            if(mark_for_write == 2'b10 && wr_rdy) begin
                cstate <= REPLACE;
            end
        end
        else if(cache_D1[index_r] && way1_valid && index_tra == 1) begin
            if(mark_for_write == 2'b10 && wr_rdy) begin
                cstate <= REPLACE;
            end
        end
        else begin
            cstate <= REPLACE;
        end
    end
    else if(cstate == REPLACE) begin
        //read from memory
        if(ret_last == 1'b1)begin
            cstate <= REFILL;
        end
    end
    else if(cstate == REFILL) begin
        //write back tp cache
        if(data_ok == 1'b1) begin
            cstate <= IDLE;
        end
    end
end
```

2. 接口定义

名称	位宽	功能描述
cstate	5	Cache 模块状态机当前状态
Index_tra	2	用于随机选取 cache 行

表 1 模块一重要信号

3. 功能描述

Cache 模块状态机的状态转移模块。

(三)重要模块 2 设计：cache 写回内存模块

1. 工作原理

当发生 cache miss 且随机选取的 cache 行的 V 和 D 位都是 1 时，需要将该 cache 行写回到主存中。具体实现方式是拉高 wr_req 发送请求并对 wrdata 进行赋值，当总线发回 wr_rdy 信号后，代表写握手成功，此时可以拉低 wr_req 信号，代码展示如下：

```
always @(posedge clk) begin
    if(resetn == 0) begin
        wr_req_r <= 1'b0;
        mark_for_write <= 2'b00;
    end
    else if(cstate == MISS) begin
        if( cache_hit == 1'b0 && wr_req_r == 0 && mark_for_write == 2'b00) begi
n
            if(index_tra == 0) begin
                if(cache_D0[index_r] && way0_valid) begin
                    wr_req_r <= 1'b1;
                    mark_for_write <= 2'b01;
                end
            end
            else if(index_tra == 1) begin
                if(cache_D1[index_r] && way1_valid) begin
                    wr_req_r <= 1'b1;
                    mark_for_write <= 2'b01;
                end
            end
        end
        else if(wr_req && wr_rdy ) begin
            wr_req_r <= 1'b0;
            mark_for_write <= 2'b10;
        end
        else if(mark_for_write == 2'b10 && wr_rdy) begin
            mark_for_write <= 2'b00;
        end
    end
end
```

2. 接口定义

名称	位宽	功能描述
Wr_req_r	1	axi 写请求信号
Index_tra	1	随机选取两路中的一路的序号
wr_rdy	1	Axi 接收写请求时发送的信号
Mark_for_write	2	用于标记当前写请求的状态

表 2 模块二重要信号

3. 功能描述

写请求模块，用于将被替换的 cache 行写回内存。

（四）重要模块 3 设计：读内存模块

1. 工作原理

当 cache miss 且 cache 进入 REPLACE 阶段时，需要从内存中读出所需的 cache 行内容并写入 cache。具体而言，进入 REPLACE 阶段时发出读请求(拉高 rd_req)，并在 rd_req 和 rd_rdy 握手成功时拉低 rd_req。当 ret_last 信号拉高时,代表此时传送的是最后一个读数据，也代表着读总线请求的结束。

```
//axi read
always @(posedge clk) begin
    if(resetn == 0) begin
        rd_req_r <= 1'b0;
        rd_type_r <= 3'b0;
        rd_addr_r <= 32'b0;
        has_axi_read_req <= 2'b00;
    end
    else if(cstate == REPLACE && (has_axi_read_req == 2'b00)) begin
        rd_req_r <= 1'b1;
        rd_type_r <= 3'b100;
        rd_addr_r <= {tlb_tag_r, index_r, 4'b0000};
        has_axi_read_req <= 2'b01;
    end
    else if(rd_rdy && rd_req_r && (has_axi_read_req == 2'b01)) begin
        rd_req_r <= 1'b0;
        has_axi_read_req <= 2'b10;
    end
    else if((ret_last == 1'b1) && (has_axi_read_req == 2'b10))begin
        has_axi_read_req <= 2'b11;
    end
    else if(has_axi_read_req == 2'b11 && cstate == REPLACE) begin
        has_axi_read_req <= 2'b00;
    end
end
```

2. 接口定义

名称	位宽	功能描述
rd_req_r	1	向 axi 发送的读请求信号
rd_type_r	3	读请求类型
rd_addr_r	32	读请求起始地址
rd_rdy	1	读请求是否被接收的握手信号
Has_axi_read_req	2	读请求状态机状态

表 3 模块 3 重要信号

3. 功能描述

从内存中读取 cache 缺失的 cache 行内容。

三、实验过程（50%）

（一）实验流水账

1. 2019 年 12 月 19 日 18:00-23:00 ， 代码编写；
2. 2019 年 12 月 20 日 09:30-12：00，完成代码编写;
3. 2019 年 12 月 20 日 14:00-21:00，debug。
4. 2019 年 12 月 21 日，14:00- 17:00，完成实验报告。

（二）错误记录（只记录部分有代表性的错误）

1. 错误 1：data_ok 拉高条件有误

（1）错误现象

Index = 0 时发生 replace 错误

```
run all
replace wrong at index 00
============================================================
Test end!
----FAIL!!!
$finish called at time : 2435 ns : File "F:/tixijiegou/cache_test/testbench/cache_testbench.v" Line 43
```

图 3 错误 1 报错

（2）分析定位过程

如图， offset_r = 0 的时候，即一次请求还没完成，offset 就变成了 1000，提前更新了，说明 data_ok 拉高的条件有误，导致请求的更新。

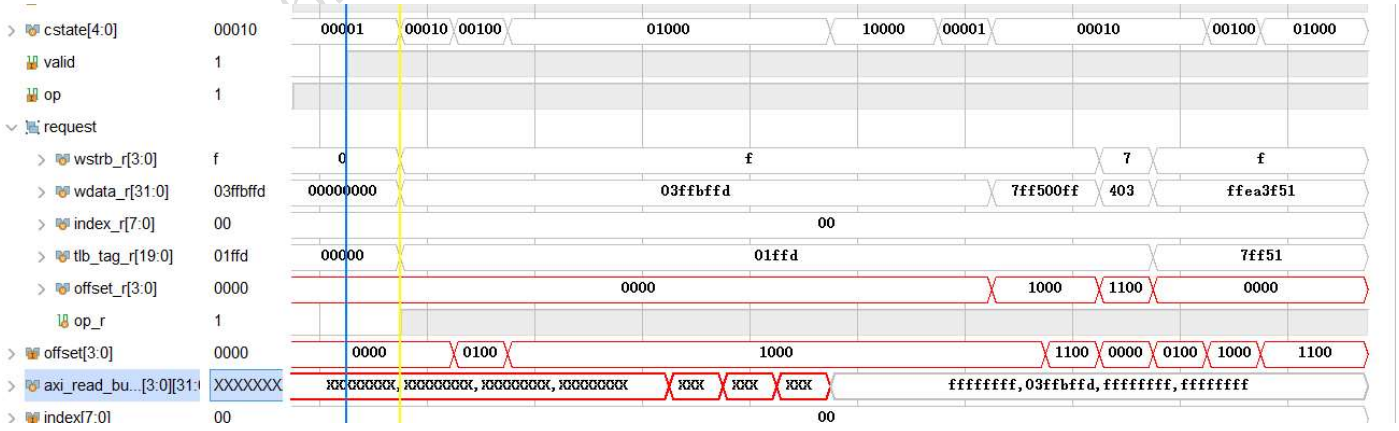

图 4 错误 1 波形图

（3）错误原因

Data_ok 拉高条件有误，data_ok 只有在 cache 命中时，完成读写请求后拉高，或者在 cache miss 时，只有 refill 结束后才拉高 data_ok 代表一次事务的结束。

2. 错误 2：握手条件有误

（1）错误现象

和错误 1 现象相同

（2）分析定位过程

状态机转移条件是 wr_rdy 和 wr_req 同时为 1，握手成功时，但是由于 wr_rdy 始终为 1，导致提前握手产生错误。

图 5 错误二波形图

（3）错误原因

开始理解的是拉高 wr_req 前,wr_rdy 一直是 0， 当写完成后才会拉高 wr_rdy，然后这次实验发现理解错误了，wr_rdy 在请求到来之前一直是 1，收到 req 后下一拍才拉低，然后完成后再次拉高，所以按照以前的理解导致提前握手。

（4）修正效果

报错，依旧是 index = 0。

3. 错误 3：refill 过程 data_ok 拉高条件问题

（1） 错误现象

进入死循环没有报错。

（2）分析定位过程

如波形图所示，四次写 cache 完成后，该进行读操作，却进入了死循环，查看测试文件的波形得知，当 4 次写 cache 结束后，wroud_state 会转化到读状态，但是由于 data_ok 拉高过早，res_counter_j 并没有达到 3，状态转移条件错开导致的错误。

修改方式是，在 cache 的 REFILL 状态写入 cache 完成后，错开一拍再拉高 data_ok，通过延迟一拍的方式保证数据写入的完成。

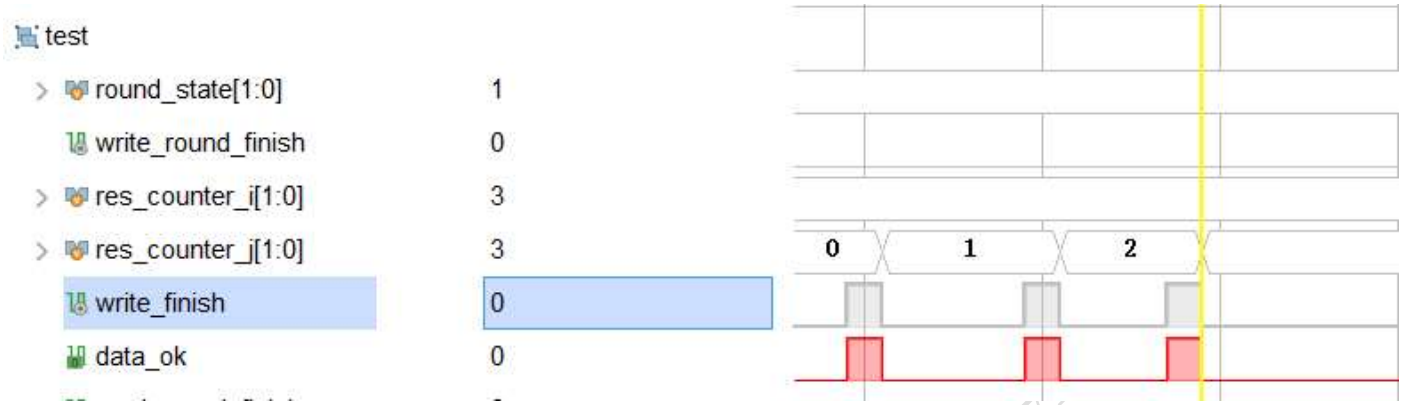

图 6 错误 3 波形图

（3）错误原因

Data_ok 拉高的条件不太准确造成的。

（4）修正效果

顺利通过所有读写测试。

四、实验总结（可选）

1.感觉对总线的握手信号理解不太准确，纠正理解花费了一些时间，感觉下周队友连接上 cpu 可能在握手信号上会花费一些功夫。

2.感觉讲义写的不太清除？或者我对讲义的 cache 理解不太准确，最后按照自己理解写的，状态机的状态转移以及 data_ok 和 addr_ok 拉高条件和讲义用的不太一致。