lab 9 report

小组成员（按姓氏倒序）:张梓堃、谭茜、林孟颖

箱子号:79

一.实验任务概览

1. 设计cace模块，并集成ICache和DCache
2. TODO

二、实验设计

(一)总体设计思路

Cache部分基本是按讲义思路构造。主状态机、写状态机、Data表和硬件初始化等与讲义一致不多赘述。Cache阻塞通过hit_write_hazard `= ( (curstate== LOOKUP) && hit_write && valid && ~op && {index, offset} == {index_r, offset_r} )|| ((wrbuf_curstate== WRBUF_WRITE) && valid && ~op && offset[3:2]== offset_r[3:2]);` 实现。写相关代码我们要

牢记：“MISS主要是读总线拿到ret_data，REPLACE主要是写总线，REFILL是写cache”，这样才能有效规避一些逻辑上的错误。

稍有区别的是为了代码更好实现，我还是把Tag和Valid分开：Tag放在RAM里，valid放在Regfile。除此之外我也完善了wr_req。

(二) cache读命令

1.tag compare：tag_rdata与tag_r作比较。

> tag_addr `=(curstate== IDLE)|| (curstate== LOOKUP) ? index : index_r;` 因为后续如果MISS则阻塞，要用request buffer保留的 `index_r` .

2.cache_hit:返回rdata=load_res;

> load_res `=data_bank_rdata[way_hit][offset_r[3:2]]`

3.如果没有命中,则返回rdata= ret_data

> 向AXI发送读请求得到ret_data
>
> `assign rdata =ret_valid? ret_data:load_res;`

(三)cache写命令

1.tag compare

2.cache_hit/hit_write:进入write buffer，写入cache并拉高valid和dirty

> wrbuf_wdata <= wdata_r

3.hit_miss: 将要写的数据（wdata_r)和内存充填(ret_data)拼一起写入cache，并拉高valid和dirty

> 3.1.cache向AXI发送读请求得到ret_data
>
> 3.2.cache用替换算法找一行
>
> —3.2.1 如果该行valid&dirty: 向AXI发送写请求， `wr_data= data_bank_rdata[replace_way][0-3]`
>
> —3.2.2 否则直接丢弃该行。**这是讲义中wr_req部分漏掉的，如果直接丢弃就不需要向AXI发送写请求了,wr_req_r直接置为0即可**

```
always @ (posedge clk) begin
     if (rst) begin
         wr_req_r <= 1'b0;
     end
    //讲义给出方案不够细致
     // else if( curstate==MISS & nxtstate==REPLACE)begin
     //     wr_req_r <=1'b1;
     // end
     else if (curstate==MISS & nxtstate==REPLACE & dirty_arr[replace_way][index_r] &
valid_arr[replace_way][index_r] ) begin
         wr_req_r <= 1'b1;
     end
```

```
        else if(curstate==MISS & nxtstate==REPLACE & ( ~dirty_arr[replace_way][index_r]
    | ~valid_arr[replace_way][index_r]) )begin
            wr_req_r <= 1'b0;
        end
        else if(wr_rdy)begin
            wr_req_r <= 1'b0;
        end
     end
```

后来发现其实置wr_req_r为0不如直接在主状态机转换中直接跳过MISS进REPLACE来得快，仿真时间也更短，缩短了30000ns

(四)集成icache更改

1. 总线部分：对比视图 (b84a02b66005bfee6fd97837ecf45b6e840c7549...52d6cc43446b3d0d887fe1279fc4e715700fa31d) · Cassie/CAlab - Gitee.com

LFSR参考csdn此文章

三、实验过程

（一）实验流水

11.20 20:00-23:00阅读讲义exp20有关部分

11.21-11.24 插空完成cache基本框架撰写

11.25 10：00-12：00 exp20 debug&完成

（二） exp20 debug

只记录一些逻辑上的错误

错误1：index=0时replace wrong

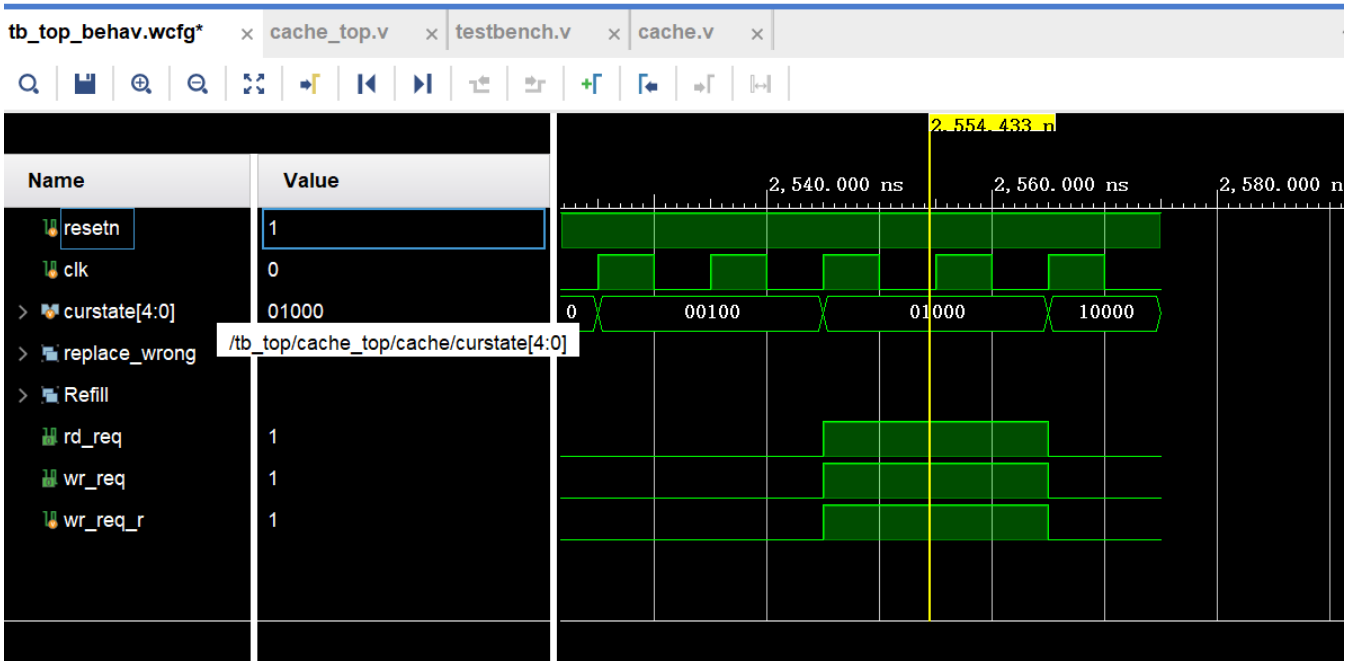
（1）错误现象

仿真刚开始时就显示replace阶段发现错误，第一个数据根本就没写进cache

```
run all
replace wrong at index 00
==========================================================
Test end!
----FAIL!!!
$finish called at time : 2225 ns : File "D:/CALAB/exp20/module_verify/cache_verify/testbench/testbench.v" Line 43
```

（2）分析定位过程

首先看波形停止的周期curstate=10000（REFILL），先查看波形分组Refill跟cache_top.v里生成的信息一致。问题只能是出在REPLACE状态。



（3）错误原因

仿真验证是先写cache进去，此时cache刚初始化，valid和dirty都为0，意味着REPLACE阶段不需要向AXI发起写请求，直接丢弃就好。可这里wr_req拉高了

```
else if( curstate==MISS & nxtstate==REPLACE)begin
            wr_req_r <=1'b1;
      end
```

更改如下：

```
else if (curstate==MISS & nxtstate==REPLACE & dirty_arr[replace_way][index_r] &
valid_arr[replace_way][index_r] ) begin
            wr_req_r <= 1'b1;
        end
        else if(curstate==MISS & nxtstate==REPLACE & ( ~dirty_arr[replace_way]
[index_r] | ~valid_arr[replace_way][index_r]) )begin
            wr_req_r <= 1'b0;
        end
```

（4）修正效果

```
index ff finished
================================================
Test end!
----PASS!!!
$finish called at time : 493515 ns : File "D:/CALAB/exp20/module_verify/cache_verify/testbench/testbench.v" Line 35
```

（5）其他

后来发现可以直接跳过MISS阶段，仿真时间更短。

```
else if (~dirty_arr[replace_way][index_r] | ~valid_arr[replace_way][index_r])
begin
                    nxtstate = REPLACE;
                end
```

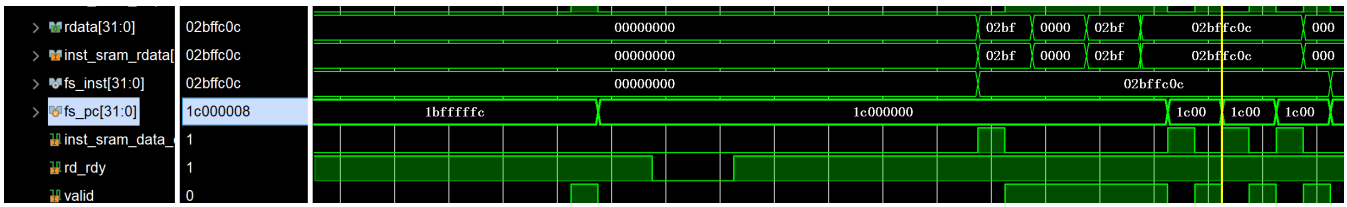
```
index fd finished
index fe finished
index fe finished
index ff finished
================================================
Test end!
----PASS!!!
$finish called at time : 463075 ns : File "D:/CALAB/exp20/module_verify/cache_verify/testbench/testbench.v" Line 35
```

（三）exp21

错误1：未设置好arburst等配置参数

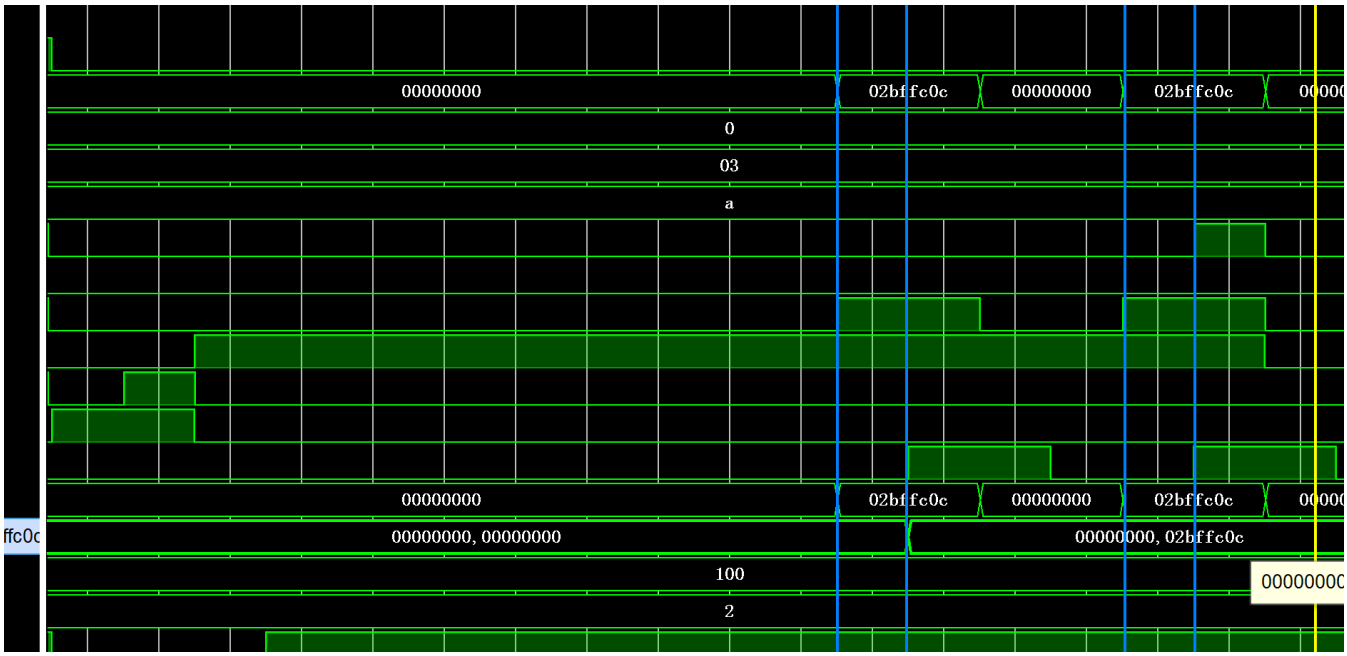
（1）错误现象

返回指令始终为一个值：



（2）分析定位过程

查看握手过程：



发现一个数据valid连续两拍拉高，导致后续返回的data重复。但是看last也正常拉高了，说明这确实已经传输了四次?

AXI协议详解（二） - 知乎 (zhihu.com)

流水线里的指令	计算得到的地址	送到类 SRAM 的地址	size	含义
取指	addr[1:0]==0	addr[1:0]==0	2	访问 4 字节
ld.w、st.w	addr[1:0]==0	addr[1:0]==0	2	访问 4 字节
ld.h、ld.hu、st.h	addr[1:0]==0	addr[1:0]==0	1	访问 2 字节
ld.h、ld.hu、st.h	addr[1:0]==2	addr[1:0]==2	1	访问 2 字节
ld.b、ld.bu、st.b	addr[1:0]==0	addr[1:0]==0	0	访问 1 字节
ld.b、ld.bu、st.b	addr[1:0]==1	addr[1:0]==1	0	访问 1 字节
ld.b、ld.bu、st.b	addr[1:0]==2	addr[1:0]==2	0	访问 1 字节
ld.b、ld.bu、st.b	addr[1:0]==3	addr[1:0]==3	0	访问 1 字节

注意到此时AR_BURST应该设置为1，而波形中为0：

AxBURST	传输类型
2'b00	固定（FIXED）
2'b01	递增（INCR）
2'b10	回环（WRAP）
2'b11	未定义 知乎 @YCHUGH

查看实现：

```
{arlen, arburst, arlock, arprot} <= {8'b0, 2'b1, 1'b0, 1'b0};   // 常值
```

意识到arlock, arprot位宽搞错。

（3）错误原因

（4）修正效果

修改后可往后执行：

```
============================================================
Test begin!
------------------------------------------------------------
[   2517 ns] Error!!!
    reference: PC = 0x1c00f07c, wb_rf_wnum = 0x04, wb_rf_wdata = 0xbfaff000
    mycpu    : PC = 0x1c00f07c, wb_rf_wnum = 0x05, wb_rf_wdata = 0xxxxxxxxx
------------------------------------------------------------
$finish called at time : 2557 ns : File "D:/Desktop/exp21/soc_verify/soc_axi/testbench/mycpu_tb.v" Line 166
```

（5）归纳总结（可选）

错误2：Cache向AXI发送的请求地址有误

（1）错误现象

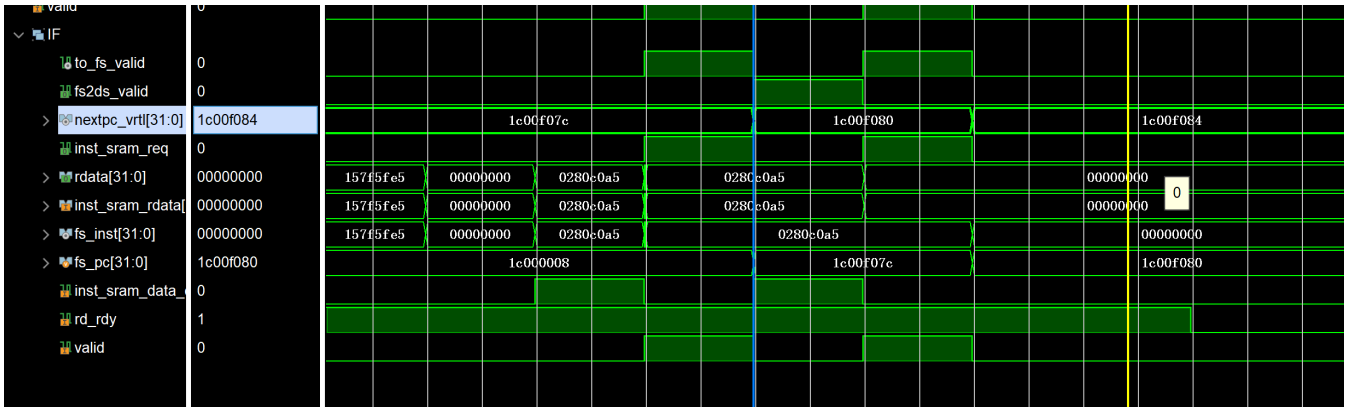
如上述，PC似乎指令不对。

（2）分析定位过程

查看反汇编指令：

```
1c00f07c <locate>:
locate():
1c00f07c:   157f5fe4    lu12i.w $r4,-263425(0xbfaff)
1c00f080:   02810084    addi.w  $r4,$r4,64(0x40)
1c00f084:   157f5fe5    lu12i.w $r5,-263425(0xbfaff)
1c00f088:   0280c0a5    addi.w  $r5,$r5,48(0x30)
1c00f08c:   157f5fe6    lu12i.w $r6,-263425(0xbfaff)
1c00f090:   028080c6    addi.w  $r6,$r6,32(0x20)
```

而此时取回的指令：



猜测cache实现有误，查看发出的读地址：

```
assign rd_addr = {tag_r, index_r, offset_r};
```

（3）错误原因

意识到读取以cache行为单位，故应该强制将offset置零。

（4）修正效果

pc往后更新。

（5）归纳总结（可选）

四.实验总结