实验十六报告

学号　2017K8009929044
2017K8009929025
姓名　李昊宸　李颖彦
箱子号　33

一、实验任务（10%）

1.在 CPU 中增加 icache，采用 2 路组相连，每路大小 4KB， LRU 或伪随机替换算法。

二、实验设计（40%）

（一）总体设计思路

使用 lab15 写成的 cache 模块，加入到 if 模块中，作为 icache 提高取指速度。同时修改转接桥，使其支持 burst 传输。总体上形成 cpu->cache->axi 转接桥的格局。

（二）重要模块 1 设计：if 模块

1、工作原理

在 if 模块中新增 cache，需要更新原有的输入输出接口，以及修正若干由时序变好而出现的 bug。

2、具体代码实现

a) 输入输出接口更改：

原接口：

```
input    inst_data_ok,
input    inst_addr_ok,
output          inst_req     ,
output [ 3:0] inst_sram_wen    ,
output [31:0] inst_sram_addr ,
output [31:0] inst_sram_wdata,
input    [31:0] inst_sram_rdata
```

现接口：

```
output              rd_req,
output[ 2:0]    rd_type,//3'b000-BYTE    3'b001-HALFWORD 3'b010-WORD 3'b100-cache-row
output[31:0]    rd_addr,
input               rd_rdy,//read_req can be accepted
input               ret_valid,
input [ 1:0]    ret_last,
input [31:0]    ret_data
```

可见如今输出的接口全部是 cache 的发往转接桥的请求。

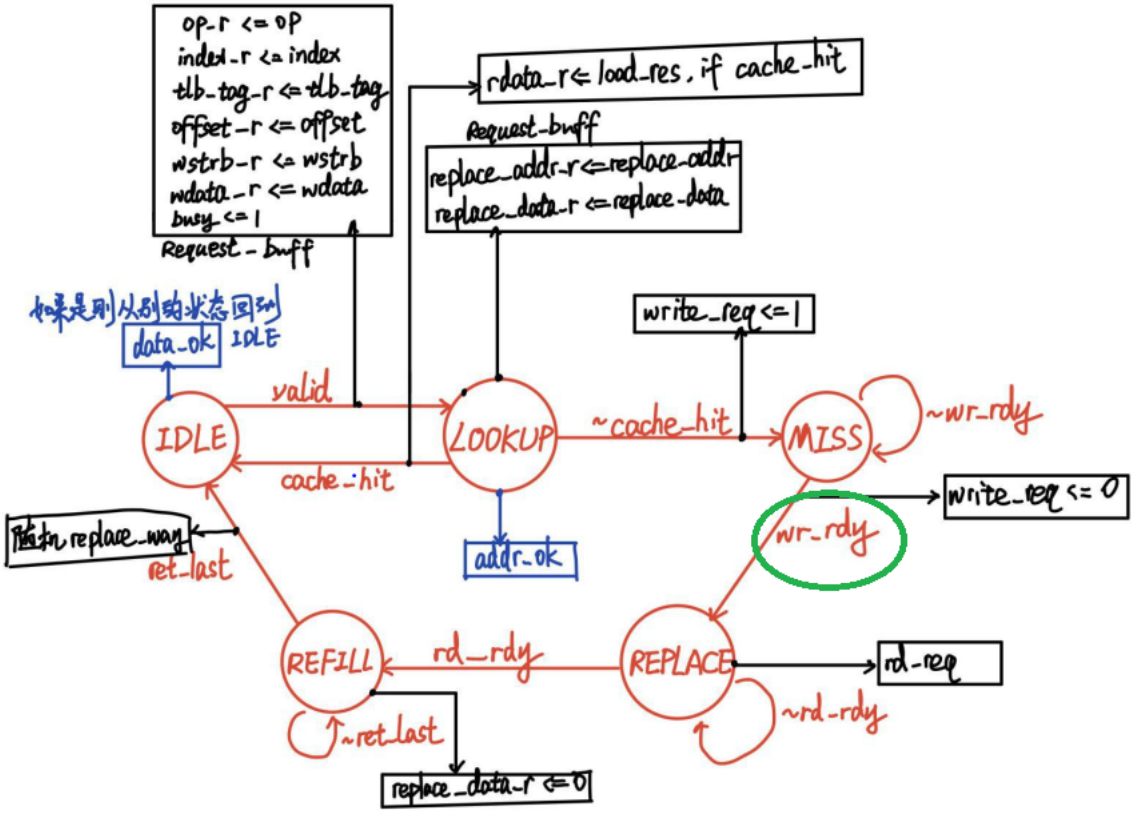
b) cache 模块调用

在 if 模块中，新增下列代码：

```
    cache icache(
        //global
        .clk(clk),
        .resetn(~reset),
        //CPU<->CACHE
        .valid(inst_req),                   //这是原来的 inst_req，现在改为直接向 cache 发送请求。
        .op(0),//1 write 0 read
        .index(inst_sram_addr[11:4]), //addr[11:4]
        .tlb_tag(inst_sram_addr[31:12]), //pfn
        .offset(inst_sram_addr[3:0]), //addr[3:0]
        .wstrb(0),
        .wdata(0),                          //icache 只需读，不需要写
        .addr_ok(inst_addr_ok),
        .data_ok(inst_data_ok),
        .rdata(inst_sram_rdata),
        //CACHE<->AXI-BRIDGE
        //read      //这一部分是 cache 向总线发送的请求
        .rd_req(rd_req),   //这是读的请求，在 cache miss 的时候发出
        .rd_type(rd_type),//3'b000-BYTE    3'b001-HALFWORD 3'b010-WORD 3'b100-cache-row
        .rd_addr(rd_addr),
        .rd_rdy(rd_rdy),//read_req can be accepted
        .ret_valid(ret_valid),
        .ret_last(ret_last),
        .ret_data(ret_data),
        //write
```

```
        .wr_rdy(1)//write_req can be accepted, actually nonsense in inst_cache
);
```

将 wr_rdy 置为常量 1 是因为在 cache 的状态机中：



如果 wr_rdy 这一 cache 的输入信号恒为一，则在 miss 的时候不需要写握手（因为 icache 没有写回的操作）

if 阶段还有数据通路上的 bug 修正，这些在实验过程中介绍

（二）重要模块 2 设计：mycpu_top 模块

1、工作原理

mycpu_top 模块是联系 cache 和转接桥的桥梁。

2、具体代码实现

这是 if 模块的接口信号：

```
.rd_req(rd_req),
.rd_type(rd_type),//3'b000-BYTE    3'b001-HALFWORD 3'b010-WORD 3'b100-cache-row
.rd_addr(rd_addr),
.rd_rdy(rd_rdy),//read_req can be accepted
.ret_valid(ret_valid),
.ret_last(ret_last),
.ret_data(ret_data)
```

下面则是转接桥的接口信号：

```
.inst_req        (rd_req    ),
.inst_wr         (inst_wr         ),
.inst_size       (inst_size     ),
.inst_addr       (rd_addr    ),
.inst_wdata      (0 ),
.inst_rdata      (ret_data ),
.inst_addr_ok (rd_rdy),
.ret_valid(ret_valid),
.ret_last(ret_last)
```

可见，cache 的读请求即转接桥的指令读请求，指令的读地址即 chche 的读地址。而 inst_addr_ok 即为 rd_rdy，代表地址已收到，之后根据 burst 传输的特点，返回 4 次 ret_valid,一次 ret_last。

（三）重要模块 3 设计：转接桥模块

1、工作原理

为了支持 burst 传输，转接桥模块做了相应的修改。

2、 具体代码实现

a)状态机修改

在读状态机中，有下列修改：

之前：

```
case(r_curstate)
…….
Readinst, Readdata:
begin
    if(rvalid)
```

```
            r_nxtstate = ReadEnd;
        else
            r_nxtstate = r_curstate;
    end
```

之后：

```
case(r_curstate)
…….
Readinst, Readdata:
begin
    if(rvalid&&rlast)          //修改处
        r_nxtstate = ReadEnd;
    else
        r_nxtstate = r_curstate;
end
```

只有在收到rlast信号，读状态机才会转移到读结束状态，这符合burst传输的要求。

b)新增ret_valid与ret_last信号

```
always@(posedge clk)
begin
    if(rvalid && arid == 4'd0)begin
        inst_rdata <= rdata;
        ret_valid <= 1'b1;
    end
    else
        ret_valid <= 0;

    if(rvalid && arid == 4'd0 && rlast)begin
        ret_last <= 1'b1;
        end
    else
        ret_last <= 0;
```

因为inst_rdata是在时钟上升沿被赋值为rdata，所以ret_valid与ret_last应该与之同步，从转接桥输出给cache。

c) arsize与arlen

这方面特定上网查了资料，确定arsize应该赋值为2，则每次传2^2=4个字节。arsize赋值为3，则一个burst传输为3+1=拍。

```
assign arlen     =(r_curstate == Readinst||inst_req&&r_curstate == ReadStart)?8'd3: 8'd0;
```

三、实验过程（50%）

（一）实验流水账

2019/12/27　　09:00 – 12:00　未加 icache 下 debug

2019/12/28　　10:00 – 13:00　 代码改动

2019/12/28　　14：00-18：00　　icache 的 debug

（二）错误记录

历史遗留部分（加 icache 前）：

1、错误 1：fs_addr_error

（1）错误现象

在 fs_pc 发生低两位地址不对齐后却报了 tlb 例外，导致 pc 跳转至 bfc00200 处理 tlb miss 的问题

（2）分析定位过程

在 fs 发生例外时应该清空流水线才对

（3）错误原因

之前：

```
assign fs_ex = (ft_address_error|tlb_miss_reg|tlb_invalid_reg) && fs_valid && ~wb_ex;
```

之后:

```
assign fs_ex = (ft_address_error|tlb_miss_reg|tlb_invalid_reg) && fs_valid && ~wb_ex&&~WB_EX;
```

其中 WB_EX 为：

```
always@(posedge clk)begin
    if(reset)
        WB_EX<=1'b0;
    else if(wb_ex)
        WB_EX<=1'b1;
    else if(inst_addr_ok)
        WB_EX<=1'b0;
        end
```

这个错误是因为 wb_ex 持续时间较短导致错过的问题，以前测试并未发现。

加入 icache 后产生的问题：

2、错误 2：burst 传输失败

（1）错误现象

依旧是每次一个 ret_valid 和 ret_last

（2）分析定位过程

这个信号由转接桥产生，定位到转接桥，查看波形，发现 arlen 赋值不对。

（3）错误原因

之前：assign arlen　　=(inst_req)?8'd3: 8'd0;

之后:　assign arlen　　=(r_curstate == Readinst||inst_req&&r_curstate == ReadStart)?8'd3: 8'd0;

事实上 inst_req 很早就被拉底，导致 arlen 赋值为 0。修改后，可以让它在 Readinst 整个阶段和之前的 inst_req 阶段均被拉高，这样就万无一失了。

3、错误 3：arsize 的确定

（1）错误现象

取值是的 pc 正确，可指令码却是反汇编代码中每隔 4 个一个

```
9fc00000 <_ftext>:
_ftext():
/home/jia-fan/work/jiafan_20191014/lab16-19/func/start.S:24
9fc00000:   2408ffff    li  t0,-1
/home/jia-fan/work/jiafan_20191014/lab16-19/func/start.S:25
9fc00004:   2408ffff    li  t0,-1
/home/jia-fan/work/jiafan_20191014/lab16-19/func/start.S:26
9fc00008:   100001c2    b   9fc00714 <locate>
/home/jia-fan/work/jiafan_20191014/lab16-19/func/start.S:27
9fc0000c:   00000000    nop
/home/jia-fan/work/jiafan_20191014/lab16-19/func/start.S:30
9fc00010:   3c088000    lui t0,0x8000
/home/jia-fan/work/jiafan_20191014/lab16-19/func/start.S:31
9fc00014:   25290001    addiu   t1,t1,1
/home/jia-fan/work/jiafan_20191014/lab16-19/func/start.S:32
9fc00018:   01005025    move    t2,t0
/home/jia-fan/work/jiafan_20191014/lab16-19/func/start.S:33
9fc0001c:   01ae5821    addu    t3,t5,t6
/home/jia-fan/work/jiafan_20191014/lab16-19/func/start.S:34
9fc00020:   8d0c0000    lw  t4,0(t0)
```

取出的指令码为 2408ffff,3c08800,8d0c000

（2）分析定位过程

这时候确实是 4 个 valid 一个 last，但是每四个一个属实奇怪，应该是数据长度的问题

（3）错误原因

之前：arsize=4

之后： arsize=2

piazza 上有一个关于 arsize 的问题，让我意识到 arsize=4 并不代表是 4 个字节，故上网查阅资料，每拍数据长度为 2^arsize 字节。

3、 错误 4：时序问题 1

前言：INST_ERET 与 inst_eret 的关系与前文的 WB_EX 和 wb_ex 的关系相同：

```
always@(posedge clk)begin
    if(reset)
        WB_EX<=1'b0;
    else if(wb_ex)
        WB_EX<=1'b1;
    else if(inst_addr_ok)
        WB_EX<=1'b0;
        end
```

（1）错误现象

加入 icache 后时序明显变好，但是也出现了一些意想不到的问题：一个 pc 被执行了两次，而前一次是 eret 后一条的指令码。

（2）分析定位过程

此时 fs 应该报出例外，放置 eret 后面的指令执行。

（3）错误原因

之前：

assign fs_inst = (~WB_EX & ~fs_ex &~INST_ERET & ~ws_refetch &~WS_REFETCH &~tlb_miss_reg &~tlb_invalid_reg)? inst_sram_rdata : 32'b0;

之后:

assign fs_inst = (~wb_ex&~WB_EX & ~fs_ex & ~inst_eret&~INST_ERET & ~ws_refetch &~WS_REFETCH &~tlb_miss_reg &~tlb_invalid_reg)? inst_sram_rdata : 32'b0;

因为时序较好，在 wb 阶段报出 inst_eret 的当拍，eret 后面的指令就已经执行了，所以应该&~inst_eret，毕竟 reg 需要在时钟上升沿执行，慢一拍。

5、错误 5：时序问题 2

（1）错误现象

例外被跳过

（2）分析定位过程

true_npc 出了问题，WB_EX 这个信号用了上一个信号的 inst_addr_ok,从而被拉底

得出结论: true_npc 也要做相应的更改，需要加入当拍的信号，而不能只靠寄存器信号。

（3）错误原因

之前：

```
assign true_npc= (WB_EX&&TLB_EX)?32'hbfc00200:
                 (WB_EX&&~TLB_EX)?32'hbfc00380:
                 INST_ERET? cp0_rdata :
                 WS_REFETCH?cp0_rdata:
                 buf_valid ?buf_npc:
                 br_valid & (j_reg | ~br_taken)?buf_br:
                 nextpc;
```

之后:

```
assign true_npc= ((WB_EX||wb_ex )&&(tlb_ex||TLB_EX))?32'hbfc00200:
                 ((WB_EX||wb_ex )&&~TLB_EX&&~tlb_ex)?32'hbfc00380:
                 (INST_ERET||WS_REFETCH)? CP0_RDATA :
                 (inst_eret||ws_refetch)?cp0_rdata:
                 buf_valid ?buf_npc:
                 br_valid & (j_reg | ~br_taken)?buf_br:
                 nextpc;
```

wb_ex 拉高时恰好有 addr_ok 的到来，导致 WB_EX 又立马被拉底，如此修改后，便不会错过 wb_ex。

四、实验总结（可选）

完结撒花~

一路走来，感慨很多，人都是被逼出来的。通过自己动手，一行行代码地书写，，verilog 的代码能力有了很大

的提高，对硬件底层有了深刻认识。

在最后，郑重地感谢助教团队:谢谢你们的无私付出！