实验 11 报告

学号：2017K80099290013　2017K8009929008

姓名：

箱子号：72

一、实验任务（10%）

将原有 CPU 顶层修改为 AXI 接口，在 CPU 内部完成取值和数据访问的仲裁；并完成固定延迟和随机延迟的功能测试；

二、实验设计（40%）

（一）总体设计思路

1、 cpu 顶层：

* 将原来的 sram 交互接口改为 AXI 交互接口

* 内部定义类 sram 信号：这些信号在 IF、EXE、MEM 和转接桥模块间连接，完成类 sram 信号的生成并和转接桥交互

* AXI 接口信号由转接桥根据类 sram 信号生成并传输

2、 Pre IF 和 IF 阶段：

* 接收来自 ID，WB 阶段的数据信号

* preIF 阶段生成类 sram 指令请求接口，IF 阶段接收转接桥传回的指令数据

* 两个阶段都需要通过握手传输，等待握手信号时需要阻塞

3、 EXE 阶段：

* 生成类 sram 数据请求接口，通过握手传输，等待握手信号时需要阻塞

4、MEM 阶段：

* 接收转接桥传回的数据并使用，数据通过握手传输，等待握手信号时需要阻塞

（二）重要模块设计：IF 阶段

1、部分代码

```
assign to_fs_valid  = ~reset&&inst_addr_ok;
```

图一：to_fs_valid 的判断逻辑

```
assign true_nextpc= buf_npc_valid?buf_nextpc:nextpc;
always @(posedge clk) begin
    if (reset) begin
        buf_npc_valid_r <= 1'b0;
    end
    else if((to_fs_valid&&fs_allowin))begin
        buf_npc_valid_r<=1'b0;
    end
    else if (!buf_npc_valid) begin
        buf_npc_valid_r<= 1'b1;
    end
    if(!buf_npc_valid)begin
        buf_nextpc<=nextpc;
    end
end
wire buf_npc_valid;
assign buf_npc_valid=buf_npc_valid_r&&~(ws_to_if_bus[34]&& (ws_to_if_bus[32]||ws_to_if_bus[33]));
```

图二：next_pc 的判断逻辑

```
assign true_inst_rdata= buf_rdata_valid ? buf_inst_rdata:inst_rdata
always @(posedge clk) begin
    if (reset) begin
        buf_rdata_valid <= 1'b0;
    end
    else if(fs_valid&&ds_allowin)begin
        buf_rdata_valid<=1'b0;
    end
    else if (!buf_rdata_valid&&inst_data_ok) begin
        buf_rdata_valid<= 1'b1;
    end
    if(!buf_rdata_valid&&inst_data_ok)begin
        buf_inst_rdata<=inst_rdata;
    end
end
```

图三：读取到指令

```
reg buf_fsreadygo;   //inst_data_ok_buf  && valid
always @(posedge clk) begin
    if(reset)
        buf_fsreadygo<=1'b0;
    else if(ds_allowin)
        buf_fsreadygo<=1'b0;
    else if(!buf_fsreadygo && inst_data_ok)
        buf_fsreadygo<=1'b1;
end
assign fs_ready_go    = buf_fsreadygo || inst_data_ok;
```

图四：IF 阶段 ready_go 的判断逻辑

2、功能描述：

* 当转接桥发来请求握手信号时，将 to_fs_valid 置 1，表示指令请求传输成功，可以进入 IF 阶段；

* 当 IF 阶段的没有准备好时（allowin 信号不拉高），mextpc 寄存器和 true_nextpc 的值保持不变

* 当数据通道传来握手信号，数据传输完成，将 IF 阶段的 ready_go 置 1；

* 当 IF 阶数据传输完成，但 ID 阶段没有准备好时，inst rdata 寄存器和 true_inst_rdata 的值保持不变；

（三）重要模块设计：EXE 阶段：

1、部分代码：

```
assign data_req=es_valid && write_reg && ms_allowin && (es_load_op||es_store_op);

assign data_size=(sw_op||lw_op||
        ((lwl_op||swl_op)&&(last_addr==2||last_addr==3))||
        ((lwr_op||swr_op)&&(last_addr==0||last_addr==1)))?2'b10
        :(
        (
            (lh_op||sh_op||lhu_op)||
            ((lwl_op||swl_op)&&last_addr==1)||
            ((lwr_op||swr_op)&&last_addr==2))?2'b01:2'b00
        );
```

图五：EXE 阶段向 AXI 输入的部分信号生成逻辑

```
assign es_ready_go    =(( (es_div_op && s_dout_tvalid==0) || (es_divu_op==1 && u_dout_tvalid==0)
            || block_mfhi_lo
            ||(data_req&&~data_addr_ok))&& (~es_valid || write_reg))? 0 : 1;
```

图六：EXE 阶段 ready_go 的判断逻辑

2、功能描述：

* 如果 es 阶段是访存指令且有效，且后续阶段没有出现中断、ere 信号，且没有阻塞，则向转接桥发出数据请求；

* 根据 EXE 阶段的指令和访存地址生成 data_size；

* ready_go 增加阻塞条件，如果请求传输未完成，则 EXE 阶段保持阻塞状态；

（四）重要模块设计：MEM 阶段

1、部分代码：

```
assign true_data_rdata=buf_rdata_valid?buf_data_rdata:data_rdata;
always @(posedge clk) begin
    if (reset) begin
        buf_rdata_valid <= 1'b0;
    end
    else if(ms_valid&&ws_allowin)begin
        buf_rdata_valid <= 1'b0;
    end
    else if (!buf_rdata_valid&&data_data_ok) begin
        buf_rdata_valid <= 1'b1;
    end
    if(!buf_rdata_valid&&data_data_ok)begin
        buf_data_rdata  <= data_rdata;
    end
end
```

图七：MEM 阶段得到访存结果

```
assign ms_ready_go    = ((load_op||store_op)&&(~data_data_ok&&(~ms_ex_op||~ms_valid)))?0:1;
```

图八：MEM 阶段 ready_go 的判断逻辑

2、功能描述：

* 用寄存器保存转接桥返回的数据，当 WB 阶段没有准备好，寄存器和 true_data_rdata 保持不变；

* 如果 MEM 阶段是访存指令，且没有中断、eret 信号，且数据传输未完成，则 MEM 阶段保持阻塞状态；

三、实验过程（50%）

（一）实验流水账

周一下午：阅读任务书，更新环境，整理实验思路；

周二到周六：逻辑设计，代码实现，仿真调试，上板运行；

周六晚上：小组交流代码实现和报告内容；

周一晚上：写报告；

（二）错误记录

1、错误 1：br_bus，wb 阶段传来的中断信息等易变信号没有用寄存器保持（以 br_bus 为例）：

错误现象：仿真报错：

```
------------------------------------------------------------
[   2297 ns] Error!!!
    reference: PC = 0xbfc006b8, wb_rf_wnum = 0x04, wb_rf_wdata = 0xbfb00000
    mycpu    : PC = 0xbfc00010, wb_rf_wnum = 0x08, wb_rf_wdata = 0x80000000
```

图九：错误现象 1

错误查找过程：

图十：错误波形 1

根据提示查看错误指令的波形，发现是因为此处的 br_taken 只拉高了一拍，而拉高时的 nextpc 并没有及时更新到 nextpc buf 中，也没有及时使用掉，导致跳转未成功

修改方法：

在 preIF 阶段添加 br_bus 的 buf，在跳转标志拉高时，如果不能及时使用，寄存到 buf 中，未被成功使用前要保持。

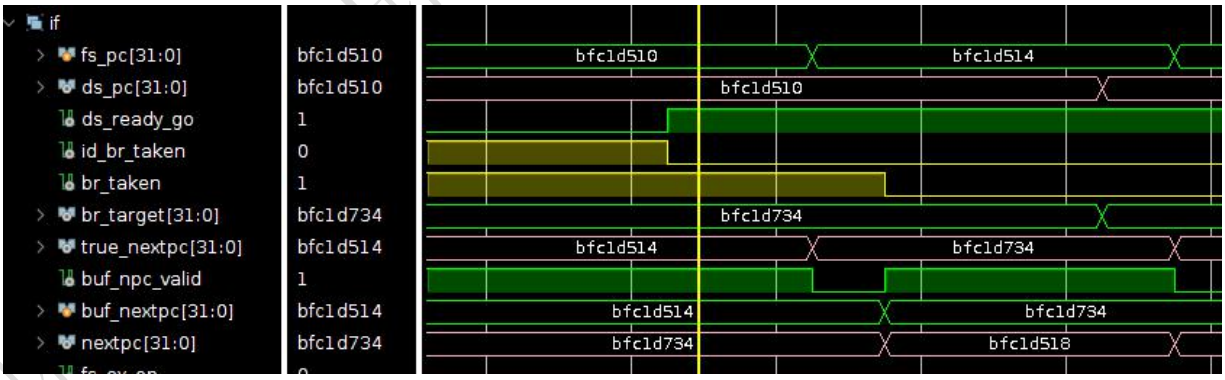
2、错误 2：id 阶段 br_taken 的拉高条件没有加 id_readygo

错误现象：仿真报错：

```
[2925327 ns] Error!!!
    reference: PC = 0xbfc1d518, wb_rf_wnum = 0x15, wb_rf_wdata = 0x00000001
    mycpu    : PC = 0xbfc1d734, wb_rf_wnum = 0x09, wb_rf_wdata = 0x14000000
```

图十一：错误现象 2

错误查找过程：



图十二：错误波形 2

定位到报错指令处，发现是跳转错误：id 阶段在等待 mem 阶段的前递数据，用来判断是否跳转，数据更新后跳转条件不成立，而因为之前跳转判断中没有考虑 readygo，导致用更新前的旧数据判断跳转条件成立，送往 preif 后被寄存，导致发生一次本不该发生的跳转

修改方法：

br_taken 的拉高条件与上 ds_ready_go：

```
assign br_taken = (   (inst_beq  &&  rs_eq_rt)
                   || (inst_bne  && !rs_eq_rt)
                   || (inst_bgez && (~rs_value[31]))
                   || (inst_bgtz && ((~rs_value[31])&(|rs_value)))
                   || (inst_blez && (rs_value[31]|(~(|rs_value)) ) )
                   || (inst_bltz && (rs_value[31]))
                   || (inst_bltzal && (rs_value[31]))
                   || (inst_bgezal && (~rs_value[31]))
                   || inst_jal
                   || inst_jr
                   || inst_jalr
                   || inst_j
                  ) && ds_valid && ds_ready_go;
```

图十三：修改方法 2

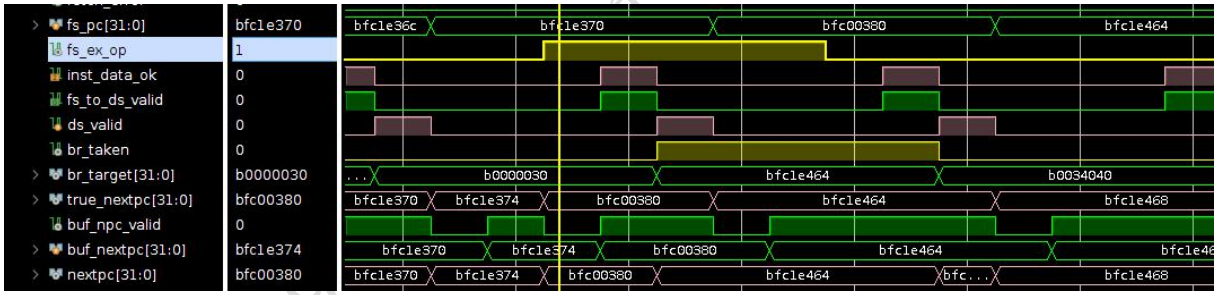
3、错误 3：if 阶段收到中断发生的信息后，仍然会因为握手成功拉高 fs_to_id_valid

错误现象：仿真报错：

```
[7849697 ns] Error!!!
    reference: PC = 0xbfc00384, wb_rf_wnum = 0x1b, wb_rf_wdata = 0x00000000
    mycpu    : PC = 0xbfc1e464, wb_rf_wnum = 0x09, wb_rf_wdata = 0x45000000
```

图十四：错误现象 3

错误查找过程：



图十五：错误波形 3

查看仿真报错的指令处的波形图，发现 if 阶段在收到 wb 阶段传来的中断信息，拉高 ex_op 后，仍然因为握手成功拉高了 fs_to_id_valid，导致本来因为中断失效的 id 阶段重新拉高 valid，并因此传来跳转命令，导致错误的跳转

修改方法：

fs_to_id_valid 的赋值逻辑中与上中断有效的非，如下：

```
assign fs_to_ds_valid = (fs_valid && fs_ready_go) && ~(ws_to_if_bus_true[34] && (fs_ex_op||fs_eret));
```

图十六：修改方法 3