# 中国科学院大学
# 《计算机体系结构基础(研讨课)》实验报告

姓名 艾华春,李霄宇,王敬华

学号 2022K8009916011,2022K8009929029,2022K8009925009

实验项目编号 5 实验名称 AXI 总线接口设计

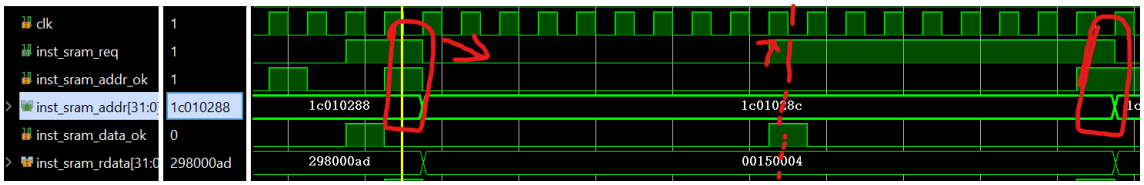
## 一、 逻辑电路结构与仿真波形的截图及说明

- **取值模块添加 sram 总线支持。**

1. 在 pre_if 级发送访存请求

   相比于之前的设计,调整发送请求的时机。在类 sram 总线上连续读时,为了控制已发送请求但未响应的事务累积以简化设计,主方在上一个请求未响应之前,拉低 req 信号来暂停发送新事物的请求。

```
assign inst_sram_req =  resetn & ~pre_if_reqed_reg   // pre if 没有已经发出请求的指令
                        & ( inst_sram_data_ok // 上一个请求恰好返回
                            | if_ir_valid      // 上一个请求已经返回，且未进入id级
                            | if_allowin)  // 上一个请求已经返回，且已经进入id级
                        & ~br_stall;      // 转移计算已经完成
```




   如时序图中所示,在发送第一个请求后(第一个红色方框),主动拉低 req 信号,等到数据相应握手成功后,立即拉高 req 请求信号,准备发送新事务的请求。

2. pre_if 级与 if 级的握手信号

```
 assign if_allowin    =   ~if_valid
                        | if_ready_go & id_allowin ; // if与id级握手成功
assign pre_if_readygo = pre_if_reqed_reg           // pre_if级已经与sram握手成功
                        |inst_sram_req & inst_sram_addr_ok; // pre_if级刚好与sram握手成功
assign to_if_valid   =    resetn;
```


   如时序图所示,在 if 级与 id 级完成握手后,立即拉高 if 的 alloin 信号。

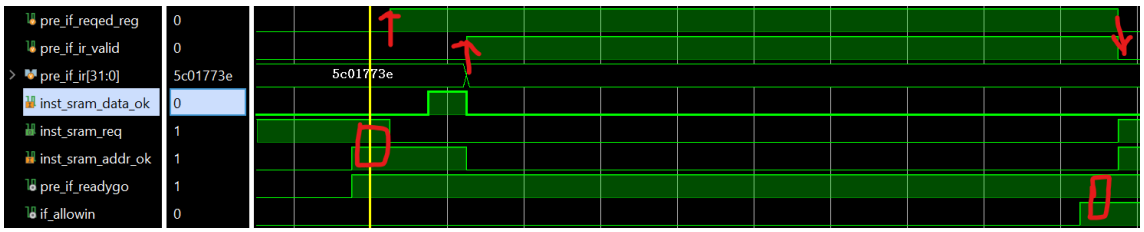
   pre_if 级的 req 信号与 inst_sram_addr_ok完成握手后,立即拉高 pre_if 级的 readygo 信号。

   pre_if 级与 if 级完成握手后,数据由 pre_if 级传递给 if 级,相应的握手信号在下一个 clk 双双拉低。

3. 在 pre_if 就返回请求的指令,(此时 if_allowin = 0)

   在 pre_if 级设置一个指令缓存,保存这个已经取回但无法进入 if 级的指令码。

```
  always @(posedge clk) begin // pre if 已经发出请求，且没有进入if级
  if(~resetn)             // 同时可以表明，当前inst_sram返回的指令是属于pre_if级的，而不是if级的
     pre_if_reqed_reg <= 1'b0;
  else if(pre_if_readygo && if_allowin) // move forward to if
     pre_if_reqed_reg <= 1'b0;
  else if(inst_sram_req && inst_sram_addr_ok) // 握手成功，且不能进入if级
     pre_if_reqed_reg <= 1'b1;
end
always @(posedge clk) begin
  if(~resetn)begin
     pre_if_ir_valid <= 1'b0;
     pre_if_ir <= 32'b0;
  end
  else if(  inst_sram_data_ok // 指令返回
            & pre_if_reqed_reg // pre if 已经发出请求，且没有进入if级
            & ~if_allowin    // 不能进入if级
            & ~inst_cancel)  begin
     pre_if_ir_valid <= 1'b1;
     pre_if_ir <= inst_sram_rdata;
  end
  else if(if_allowin & pre_if_readygo)begin
     pre_if_ir_valid <= 1'b0;
  end
end
```




如图，在 pre if 与 sram 地址握手成功后，但是 if_allowin = 0，不能进入 if 级，则在下一个 clk 拉高 pre_if_reqed_reg 信号,表示 pre if 级当前已经发送了一条请求,且如果此时返回指令,则是对应 pre if 级的 pc,而不是 if 级的。

当 if 级的 allowin 拉高以后,pre_if 级的指令缓存则传递给 if 级的指令缓存。

4. if 级接受到指令,但是 id 级还不让进入

   与上一个问题一样,在 if 级同样设置一个指令缓存,来暂存返回的指令。

5. 异常清空流水线

   当需要清空流水线时,或者跳转有效时,首先将 if 和 pre_if 级的指令缓存的 valid 信号置低。

   新增一个触发器 inst_cancel,如果 if 级或者 pre_if 级有已经发出请求,但是没有返回的事务,则将该触发器拉高,等到请求返回后,把他拉低。

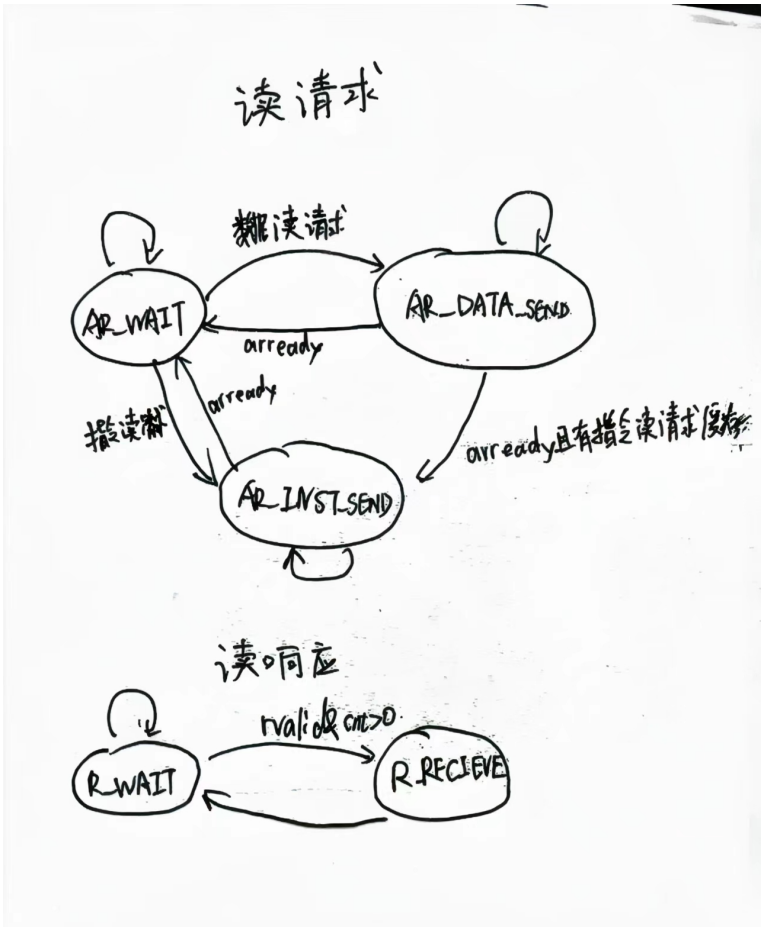
并且使用该信号拉低 vaild 信号。达到丢弃该返回的指令的目的。

```
 /* 清空流水线时，第一个指令需要丢弃*/
always @(posedge clk) begin
 if(~resetn)
    inst_cancel <= 1'b0;
 else if ( (if_valid & ~if_ir_valid & ~inst_sram_data_ok // if正在等待指令返回
           |pre_if_reqed_reg & ~inst_sram_data_ok)
       & (flush | br_taken))
    inst_cancel <= 1'b1;
 else if(inst_sram_data_ok) // 异常后第一个需要被舍弃的指令返回
    inst_cancel <= 1'b0;
end

assign if_to_id_valid = if_ready_go & ~inst_cancel;
```

- **添加 AXI 总线支持**。

1. 读请求和读响应




分别用两个独立的状态机来控制读请求和读响应通道的行为。

(a) 读请求通道

当读请求和写请求状态机都处于 wait 状态时，拉高两个类 SRAM 从方的 addr_ok 信号，表示可以接受读请求和写请求。

```
assign inst_sram_addr_ok = ar_current_state == AR_WAIT && aw_current_state == AW_WAIT;
assign data_sram_addr_ok = ar_current_state == AR_WAIT && aw_current_state == AW_WAIT;
```

如果在 AR_WAIT 中同时接受到数据读请求和指令读请求，则将指令读请求的地址暂存起来。在 AR_DATA_SEND 阶段将数据读请求的地址握手完成后，再进入 AR_INST_SEND 阶段发送指令读请求的地址。

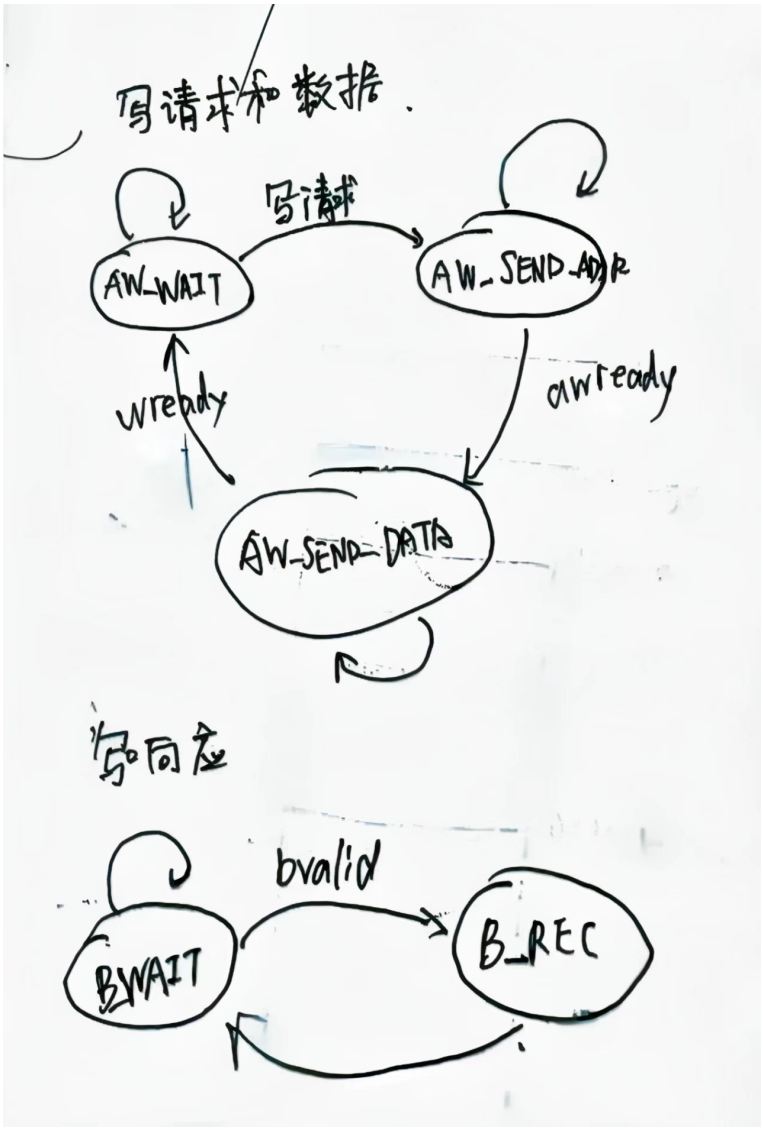
保证数据端发过来的读请求的优先级固定高于取值端发来的读请求。

(b) 读响应通道在 R_WAIT 阶段拉高 rready，表示可以接受从 AXI 从端发送的读数据。

```
assign rready = r_current_state == R_WAIT;
```

当 AXI 从方将 rvalid 置为有效，则在下一个 clk 跳转到 R_RECIEVE，将读数据缓存，并根据 ID 值发给对应的类 SRAM 主方。

2. 写请求和写响应通道




分别用两个独立的状态机来控制写请求和写响应通道的行为。

(a) 写请求和写数据通道

在 wait 阶段接受到写请求后，状态机跳转到 SEND_ADDR 阶段发送访存的地址。

在地址发送握手成功后，状态机跳转到 SEND_DATA 阶段发送写数据。

(b) 写响应

在 wait 阶段接受到 AXI 从方发送的写响应后，状态机跳转到 B_REC 状态，向类 SRAM 总线主方发送data_sram_data_ok有效信号

```
always @(*)begin
    case (b_current_state)
        B_WAIT:begin
            if(bvalid)
                b_next_state = B_REC;
            else
                b_next_state <= B_WAIT;
        end
        B_REC:begin
            b_next_state = B_WAIT;
        end
    endcase
end
```

## 二、 实验过程中遇到的问题、对问题的思考过程及解决方法（比如 RTL 代码中出现的逻辑 bug，逻辑仿真和 FPGA 调试过程中的难点等）

- **清空流水线或跳转指令将指令缓存置无效 bug。**

当 flush 或者 br_taken 有效时，如果直接在指令缓存更新的代码中，添加在下一排将指令缓存 valid 置 0 的代码后，会出现错误的丢弃跳转后取得第一条指令。

对照波形图排查后发现，是因为在 inst_cancel 更新得逻辑中，上述操作只是将指令缓存if_ir_valid置为无效，而不将 if 流水级得有效信号if_valid 置低，会导致该逻辑判断 if 级有已经发送请求但还没有返回指令的事务，则会丢弃接下来接收到的第一条指令。

```
/* 清空流水线时，第一个指令需要丢弃*/
always @(posedge clk) begin
    if(~resetn)
        inst_cancel <= 1'b0;
    else if ( (if_valid & ~if_ir_valid & ~inst_sram_data_ok // if正在等待指令返回
              |pre_if_reqed_reg & ~inst_sram_data_ok) // pre if 正在等待指令返回
           & (flush | br_taken))
        inst_cancel <= 1'b1;
    else if(inst_sram_data_ok) // 异常后第一个需要被舍弃的指令返回
        inst_cancel <= 1'b0;
end
```

最后，当 flush 或者 br_taken 有效时，不改变if_ir_valid的值，而是将 id 级的 allowin 拉高，并且将 id 级的下一排的 valid 置 0，如果 if 级有指令缓存，则会在下一个 clk 传递给 id 级而消耗掉。

```
assign id_allowin    =   ~id_valid
```

```
| id_ready_go & ex_allowin
| br_taken | flush;         // 消耗if级错误指令缓存

always @(posedge clk) begin
if(~resetn)
id_valid <= 1'b0;
else if(br_taken || flush) // 发生异常或者跳转时，下一排将id级valid置0
id_valid <= 1'b0;
else if(id_allowin)
id_valid <= if_to_id_valid;
end
```

- 指令译码定义太宽泛导致异常判断出错。

## 三、 小组成员分工合作情况

王敬华负责 exp13 的中断处理和整体的 debug 工作
李霄宇负责 exp13 的异常处理和计时器指令的实现
艾华春负责 exp12:添加系统调用异常支持
实验报告为根据每人负责代码的部分,写相应部分的报告。