中国科学院大学
《计算机组成原理(研讨课)》实验报告

姓名 孙广润, 刘钰, 蔡合森 箱子编号 18 实验项目编号 实践任务 12,13

实验名称 异常与中断设计专题

一、 添加系统调用异常支持

• 代码修改说明。

按照书中说明,我们先增加了一些异常指令有效信号

```
//新添加异常指令有效信号
assign inst_csrrd =op_31_26_d[6'h01] &op_25_24_d[2'h0] &rj==5'h0;
assign inst_csrwr =op_31_26_d[6'h01] &op_25_24_d[2'h0] &rj==5'h1;
assign inst_csrxchg =op_31_26_d[6'h01] &op_25_24_d[2'h0]&~inst_csrrd &~inst_csrwr;
assign inst_ertn = op_31_26_d[6'h01] &op_25_22_d[4'h9] &op_21_20_d[2'h0] &op_19_15_d[5'h10] &op_14_10_d[5'h0e] &rj==5'h0 &op_4_0_d[5'h0];
assign inst_syscall =op_31_26_d[6'h0] &op_25_22_d[4'h0] &op_21_20_d[2'h2] &op_19_15_d[5'h16];
```

将不同的 CSR 寄存器根据 csr_num 进行选择,将相应寄存器的值组合并输出。同时,两个输出信号 ex_entry 和 ex_epc 分别用于处理异常的入口地址和异常发生时的程序计数器(PC)。

```
assign csr_crmd_rvalue ={23'b0,csr_crmd_datm,csr_crmd_datf,csr_crmd_pg,csr_crmd_da,csr_crmd_ie,csr_crmd_plv};
assign csr_prmd_rvalue ={29'b0,csr_prmd_pie,csr_prmd_pplv};
assign csr_estat_rvalue ={1'b0,csr_estat_esubcode,csr_estat_ecode,3'b0,csr_estat_is[12:11],1'b0,csr_estat_is[9:0]};
assign csr_era_rvalue =csr_era_pc;
assign csr_eentry_rvalue ={csr_eentry_va,6'b0};

assign csr_rvalue =  csr_num ==`CSR_CRMD ?csr_crmd_rvalue :
                     csr_num ==`CSR_PRMD ?csr_prmd_rvalue :
                     csr_num ==`CSR_ESTAT ?csr_estat_rvalue :
                     csr_num ==`CSR_ERA  ? csr_era_rvalue :
                     csr_num ==`CSR_EENTRY ?csr_eentry_rvalue :
                     csr_num ==`CSR_SAVE0 ?csr_save0 :
                     csr_num ==`CSR_SAVE1 ?csr_save1 :
                     csr_num ==`CSR_SAVE2 ?csr_save2 :
                     csr_num ==`CSR_SAVE3 ?csr_save3 :
                     32'b0;
assign ex_entry =csr_eentry_rvalue;
assign ex_epc =csr_era_rvalue;
```

接下来是对控制状态寄存器(CSR)进行读写操作的 Verilog 实现,对以下寄存器(CSR_CRMD、CSR_PRMD、CSR_ESTAT、CSR_ERA、CSR_EENTRY、以及 CSR_SAVE0 3)的多个字段的更新逻辑。

```
//CRMD 的 PLV 域以及 IE 域
//csr_crmd_plv: 表示当前特权等级; csr_crmd_ie: 表示全局中断使能。
//当 reset 时, 将 csr_crmd_plv 设为 2'b00 (表示最高特权等级), 将 csr_crmd_ie 设为关闭状态。
//当异常 (wb_ex) 发生时, 将 csr_crmd_plv 和 csr_crmd_ie 都设置为最低特权等级和禁用中断。
//当 ertn_flush (异常返回) 时, 从 PRMD 寄存器中恢复以前的特权等级和中断使能状态。
//如果执行CSR写操作 (csr_we) 并且操作的目标是 CSR_CRMD, 根据写掩码 (csr_wmask) 对 PLV 和 IE 进行有选择性的更新
always @(posedge clk) begin
    if (reset) begin
        csr_crmd_plv <=2'b00;
        csr_crmd_ie <=1'b0;
    end
    else if(wb_ex)begin
        csr_crmd_plv <=2'b00;
        csr_crmd_ie <=1'b0;
    end
    else if(ertn_flush) begin
        csr_crmd_plv <=csr_prmd_pplv;
```

```
        csr_crmd_ie <=csr_prmd_pie;
    end
    else if(csr_we &&csr_num ==`CSR_CRMD) begin
        csr_crmd_plv <=csr_wmask[`CSR_CRMD_PLV] &csr_wvalue[`CSR_CRMD_PLV]
                    | ~csr_wmask[`CSR_CRMD_PLV] &csr_crmd_plv;
        csr_crmd_ie <=csr_wmask[`CSR_CRMD_IE] &csr_wvalue[`CSR_CRMD_IE]
                    | ~csr_wmask[`CSR_CRMD_IE] &csr_crmd_ie;
    end
end

//CRMD 的DA、PG、DATF、DATM 域
//当 reset 时，字段 csr_crmd_da、csr_crmd_pg、csr_crmd_datf 和 csr_crmd_datm 被初始化为各自的默认值
always @(posedge clk) begin
    if (reset) begin
        csr_crmd_da <=1'b1;
        csr_crmd_pg <=1'b0;
        csr_crmd_datf <=2'b00;
        csr_crmd_datm <=2'b00;
    end
end

//PRMD 的 PPLV 域以及 PIE 域
//当发生异常（wb_ex）时，将 CSR_CRMD 的 PLV 和 IE 保存到 CSR_PRMD 中
//如果执行CSR写操作并且目标是 CSR_PRMD，根据写掩码选择性更新 PPLV 和 PIE
always @(posedge clk) begin
    if(wb_ex)begin
        csr_prmd_pplv <=csr_crmd_plv;
        csr_prmd_pie <=csr_crmd_ie;
    end
    else if(csr_we &&csr_num==`CSR_PRMD) begin
        csr_prmd_pplv <=csr_wmask[`CSR_PRMD_PPLV] &csr_wvalue[`CSR_PRMD_PPLV]
                    |  ~csr_wmask[`CSR_PRMD_PPLV] &csr_prmd_pplv;
        csr_prmd_pie <=csr_wmask[`CSR_PRMD_PIE] &csr_wvalue[`CSR_PRMD_PIE]
                    |  ~csr_wmask[`CSR_PRMD_PIE] &csr_prmd_pie;
    end
end

//ESTAT 的 IS 域
//当发生异常（wb_ex）时，异常代码 wb_ecode 和子代码 wb_esubcode 被写入 csr_estat_ecode 和 csr_estat_esubcode
//低位 csr_estat_is[1:0] 会在CSR写操作时选择性更新。其他中断状态位被初始化为0，或保留位为0
always @(posedge clk) begin
    if(wb_ex)begin
        csr_estat_is[1:0] <=2'b00;
    end
    else if(csr_we &&csr_num==`CSR_ESTAT) begin
        csr_estat_is[1:0] <=csr_wmask[`CSR_ESTAT_IS10] &csr_wvalue[`CSR_ESTAT_IS10]
                        |  ~csr_wmask[`CSR_ESTAT_IS10] &csr_estat_is[1:0];
    end
    csr_estat_is[9:2] <=8'b0;
    csr_estat_is[10] <=1'b0;
    csr_estat_is[12:11] <=2'b0;
end

//ESTAT 的 ECODE 域以及 ESUBCODE 域
always @(posedge clk) begin
    if(wb_ex)begin
        csr_estat_ecode <=wb_ecode;
        csr_estat_esubcode <=wb_esubcode;
    end
end

//ERA 的 PC 域
//当发生异常（wb_ex）时，将当前的PC（wb_pc）存入 CSR_ERA。
//如果执行CSR写操作并且目标是 CSR_ERA，根据写掩码选择性更新PC值
always @(posedge clk) begin
    if(wb_ex)begin
        csr_era_pc <=wb_pc;
    end
```

```
    else if(csr_we &&csr_num==`CSR_ERA) begin
        csr_era_pc <=csr_wmask[`CSR_ERA_PC] &csr_wvalue[`CSR_ERA_PC]
                | ~csr_wmask[`CSR_ERA_PC] &csr_era_pc;
    end
end

//EENTRY 的 VA 域
//当执行CSR写操作并且目标是 CSR_EENTRY 时，根据写掩码选择性更新异常入口地址的高位
always @(posedge clk) begin
    if(csr_we &&csr_num==`CSR_EENTRY) begin
        csr_eentry_va <=csr_wmask[`CSR_EENTRY_VA] &csr_wvalue[`CSR_EENTRY_VA]
                |  ~csr_wmask[`CSR_EENTRY_VA] &csr_eentry_va;
    end
end

//SAVE0~3 的数据域
//如果执行CSR写操作并且目标是 CSR_SAVE0~CSR_SAVE3，根据写掩码选择性更新这些寄存器的数据域
always @(posedge clk) begin
    if(csr_we &&csr_num==`CSR_SAVE0) begin
        csr_save0 <=csr_wmask[`CSR_SAVE0_DATA] &csr_wvalue[`CSR_SAVE0_DATA]
                | ~csr_wmask[`CSR_SAVE0_DATA] &csr_save0;
    end
    else if(csr_we &&csr_num==`CSR_SAVE1) begin
        csr_save1 <=csr_wmask[`CSR_SAVE1_DATA] &csr_wvalue[`CSR_SAVE1_DATA]
                | ~csr_wmask[`CSR_SAVE1_DATA] &csr_save1;
    end
    else if(csr_we &&csr_num==`CSR_SAVE2) begin
        csr_save2 <=csr_wmask[`CSR_SAVE2_DATA] &csr_wvalue[`CSR_SAVE2_DATA]
                | ~csr_wmask[`CSR_SAVE2_DATA] &csr_save2;
    end
    else if(csr_we &&csr_num==`CSR_SAVE3) begin
        csr_save3 <=csr_wmask[`CSR_SAVE3_DATA] &csr_wvalue[`CSR_SAVE3_DATA]
                | ~csr_wmask[`CSR_SAVE3_DATA] &csr_save3;
    end
end
```

相关信号的一些修改，都在末尾添加相关的信号有效 (比如我们对于 need_rd 与 src_reg_is_rd 两个信号添加 inst_csrwr 与 Inst_csrxchg)

```
assign need_rd =  inst_beq |inst_bne |inst_st_b |inst_st_h |inst_st_w |inst_blt |inst_bge |inst_bltu |inst_bgeu |inst_csrwr |
     inst_csrxchg;

assign src_reg_is_rd =inst_beq |inst_bne |inst_st_b |inst_st_h |inst_st_w |inst_blt |inst_bge |inst_bltu |inst_bgeu |inst_csrwr |
     inst_csrxchg;

assign gr_we      = ~inst_st_b &~inst_st_h &~inst_st_w &~inst_beq &~inst_bne &~inst_b &~inst_blt &~inst_bge &~inst_bltu &~
     inst_bgeu &~inst_ertn &~inst_syscall;

assign br_target =(inst_beq ||inst_bne ||inst_bl ||inst_b ||inst_blt ||inst_bge ||inst_bltu ||inst_bgeu) ?(pc_ID +br_offs) :
                (inst_ertn)?ex_epc:
                (inst_syscall)?ex_entry:
                (rj_value +jirl_offs); /*inst_jirl*/

 assign EX_final_result =div_signed_r ?(get_div_or_mod_r ?sdiv_result[63:32] :sdiv_result[31:0]):
                      div_unsigned_r ?(get_div_or_mod_r ?udiv_result[63:32] :udiv_result[31:0]):
+                     (is_csr_EX)?csr_rvalue_EX://csr指令直接从csr中取值
                      alu_result;
```

我们对上次命名进行修改，以下一级进入阶段的名称命名，更加符合惯例，在整个工程中都进行了修改，在 patch 文件中都可以看到，这里仅以一个 patch 文件中的例子说明，不做多余赘述。

```
-assign dest_EX_ID =dest_EX &{5{gr_we_EX}} &{5{ID_valid}};
-assign dest_MEM_ID =dest_MEM &{5{gr_we_MEM}} &{5{EX_valid}};
-assign dest_WB_ID =dest_WB &{5{gr_we_WB}} &{5{MEM_valid}};
+assign dest_EX_ID =dest_EX &{5{gr_we_EX}} &{5{EX_valid}};
```

```
+assign dest_MEM_ID =dest_MEM &{5{gr_we_MEM}} &{5{MEM_valid}};
+assign dest_WB_ID =dest_WB &{5{gr_we_WB}} &{5{WB_valid}};
```

二、 添加其他异常支持

我们的 cpu 对于 CSR 寄存器的管理统一设置在 ID 阶段。其他异常里边，大部分我们都可以在 ID 阶段检测到，为简单起见，直接在 ID 阶段跳转并复用分支指令的通路。对于比较特殊的 ale 异常最早只能在 EX 阶段检测到，专门仿照 branch 的通路给它添加 EX 级跳转，EX 跳转要同时清空前两级的错取指令。

添加 rdcntvl.w、rdcntvh.w 和 rdcntid 三条指令。这里 rdcntvl.w 和 rdcntvh.w 两条指令分别读取计时器的低 32 位和高 32 位值写入到第 rd 项寄存器中。rdcntid 指令读取的就是 TID 控制状态寄存器中的内容 rdcntvl.w 和 rdcntvh.w 指令在执行、访存、写回级读取 64 位的计时器的值 rdcntid 指令所读的 TID 控制状态寄存器可以被 CSR 指令修改，我们将 rdcntid 的读取是放在 EX 阶段。在对 EX_Final_result 进行赋值的时候进行替换。

```
assign inst_break =op_31_26_d[6'h0] &op_25_22_d[4'h0] &op_21_20_d[2'h2] &op_19_15_d[5'h14];
//新添加rdcnt指令有效信号
assign inst_rdcntvl_w =op_31_26_d[6'h0] &op_25_22_d[4'h0] &op_21_20_d[2'h0] &op_19_15_d[5'h0] &op_14_10_d[5'h18] &op_9_5_d[5'h0];
assign inst_rdcntvh_w =op_31_26_d[6'h0] &op_25_22_d[4'h0] &op_21_20_d[2'h0] &op_19_15_d[5'h0] &op_14_10_d[5'h19] &op_9_5_d[5'h0];
assign inst_rdcntid_w =op_31_26_d[6'h0] &op_25_22_d[4'h0] &op_21_20_d[2'h0] &op_19_15_d[5'h0] &op_14_10_d[5'h18] &op_4_0_d[5'h0];
```

根据 csr_num 选择对应的值即可。

```
assign csr_crmd_rvalue ={23'b0,csr_crmd_datm,csr_crmd_datf,csr_crmd_pg,csr_crmd_da,csr_crmd_ie,csr_crmd_plv};
assign csr_prmd_rvalue ={29'b0,csr_prmd_pie,csr_prmd_pplv};
assign csr_ecfg_rvalue ={19'b0,csr_ecfg_lie};
assign csr_estat_rvalue ={1'b0,csr_estat_esubcode,csr_estat_ecode,3'b0,csr_estat_is[12:11],1'b0,csr_estat_is[9:0]};
assign csr_era_rvalue =csr_era_pc;
assign csr_badv_rvalue =csr_badv_vaddr;
assign csr_eentry_rvalue ={csr_eentry_va,6'b0};
assign csr_tid_rvalue =csr_tid_tid;
assign csr_tcfg_rvalue ={csr_tcfg_initval,csr_tcfg_periodic,csr_tcfg_en};
assign csr_tval_rvalue =csr_tval_timeval;
assign csr_ticlr_rvalue ={31'b0,csr_ticlr_clr};

assign csr_rvalue =  csr_num ==`CSR_CRMD ?csr_crmd_rvalue :
                     csr_num ==`CSR_PRMD ?csr_prmd_rvalue :
                     csr_num ==`CSR_ECFG ?csr_ecfg_rvalue :
                     csr_num ==`CSR_ESTAT ?csr_estat_rvalue :
                     csr_num ==`CSR_ERA  ? csr_era_rvalue :
                     csr_num ==`CSR_BADV ?csr_badv_rvalue :
                     csr_num ==`CSR_EENTRY ?csr_eentry_rvalue :
                     csr_num ==`CSR_SAVE0 ?csr_save0 :
                     csr_num ==`CSR_SAVE1 ?csr_save1 :
                     csr_num ==`CSR_SAVE2 ?csr_save2 :
                     csr_num ==`CSR_SAVE3 ?csr_save3 :
                     csr_num ==`CSR_TID  ? csr_tid_rvalue :
                     csr_num ==`CSR_TCFG ?csr_tcfg_rvalue :
                     csr_num ==`CSR_TVAL ?csr_tval_rvalue :
                     csr_num ==`CSR_TICLR ?csr_ticlr_rvalue :
                     32'b0;
```

对原来的信号做出的修改，添加新信号

```
//添加对于dst_is_rj的判定
assign dst_is_rj  = inst_rdcntid_w;
assign gr_we      = ~inst_st_b &~inst_st_h &~inst_st_w &~inst_beq &~inst_bne &~inst_b &~inst_blt &~inst_bge &~inst_bltu &~
    inst_bgeu &~inst_ertn &~inst_syscall;
assign dest       = dst_is_r1 ?5'd1 :
```

```
                    dst_is_rj ?rj :
                    rd;
// - 将跳转分为在ID的跳转以及在EX的跳转，EX的跳转相比ID的跳转额外取消一条错取指令
assign br_taken_ID =(inst_beq && rj_eq_rd
                  || inst_bne &&!rj_eq_rd
                  || inst_blt && rj_lt_rd
                  || inst_bge && !rj_lt_rd
assign br_taken_EX =exc_at_EX;
// - 调整了一下br_target的优先级
assign br_target =(wb_ex) ?ex_entry :
                  (inst_ertn) ?ex_epc :
                  (inst_jirl) ?(rj_value +jirl_offs) :
                  (pc_ID +br_offs);//branch
assign data_sram_type_tag ={{inst_ld_b |inst_ld_bu |inst_st_b}, {inst_ld_h |inst_ld_hu |inst_st_h}, {inst_st_w |inst_ld_w}, {inst_ld_bu
      |inst_ld_hu}};// {byte_en, half_en, unsigned_en}

assign EX_final_result =div_signed_r ?(get_div_or_mod_r ?sdiv_result[63:32] :sdiv_result[31:0]):
                        div_unsigned_r ?(get_div_or_mod_r ?udiv_result[63:32] :udiv_result[31:0]):
                        (is_csr_EX)?csr_rvalue_EX://csr指令直接从csr中取值
                        (is_rdcntid_EX)?counter_id:
                        (is_rdcntvl_EX)?counter_vl:
                        (is_rdcntvh_EX)?counter_vh:
                        alu_result;
assign data_sram_rdata_off =data_sram_rdata >>(data_sram_addroffset_WB *8);
assign mem_result =data_sram_type_tag_WB[3]? {{24{data_sram_rdata_off[7] &~data_sram_type_tag_WB[0]}}, data_sram_rdata_off[7:0]} :
                   data_sram_type_tag_WB[2]? {{16{data_sram_rdata_off[15] &~data_sram_type_tag_WB[0]}}, data_sram_rdata_off[15:0]} :
                   data_sram_rdata_off;
//ESTAT 的 IS 域
//逻辑用于控制 ESTAT 寄存器中的 IS 域，处理与系统中断相关的状态。
通过 csr_wmask 来控制位的选择，确保写入操作能够有选择性地修改寄存器的特定位，而非全局覆盖。
always @(posedge clk) begin
   if(reset)begin
      csr_estat_is[1:0] <=2'b0;
   end
   else if(csr_we &&csr_num==`CSR_ESTAT) begin
      csr_estat_is[1:0] <=csr_wmask[`CSR_ESTAT_IS10] &csr_wvalue[`CSR_ESTAT_IS10]
                     |  ~csr_wmask[`CSR_ESTAT_IS10] &csr_estat_is[1:0];
   end
   csr_estat_is[9:2] <=8'b0;
   csr_estat_is[10] <=1'b0;

   if(timer_cnt ==32'b0) begin
      csr_estat_is[11] <=1'b1;
   end
   else if(csr_we &&csr_num==`CSR_TICLR &&csr_wmask[`CSR_TICLR_CLR] &&csr_wvalue[`CSR_TICLR_CLR]) begin
      csr_estat_is[11] <=1'b0;
   end

   csr_estat_is[12] <=1'b0;
end
```

修改寄存器控制模块

```
always @(posedge clk) begin //寄存器控制
    if(reset) begin
       is_csr_EX <=1'b0;
       csr_rvalue_EX <=32'h0;
      is_rdcntid_EX <=1'b0;
      is_rdcntvl_EX <=1'b0;
      is_rdcntvh_EX <=1'b0;
    end
    else if(EX_allowin &&ID_valid &&ID_readygo) begin
       is_csr_EX <=csr_re;
       csr_rvalue_EX <=csr_rvalue;
      is_rdcntid_EX <=inst_rdcntid_w;
      is_rdcntvl_EX <=inst_rdcntvl_w;
      is_rdcntvh_EX <=inst_rdcntvh_w;
```

```
    end
end
```

has_int 是一个布尔信号,表示当前是否有中断且全局中断使能,counter_id 是从 csr_tid_tid 寄存器中读取的计数器 ID。

```
+assign has_int =(csr_estat_is[12:0] &csr_ecfg_lie[12:0])!= 13'b0 &&csr_crmd_ie;
+assign counter_id =csr_tid_tid;
```

当系统复位时，将 csr_ecfg_lie 清零。在写操作时，根据 csr_wmask 来决定哪些位会被更新，更新的值由 csr_wvalue 提供,且只更新有效的 12 位

```
//ECFG 的 LIE 域
always @(posedge clk) begin
    if(reset)begin
        csr_ecfg_lie <=13'b0;
    end
    else if(csr_we &&csr_num==`CSR_ECFG)begin
         csr_ecfg_lie <=csr_wmask[`CSR_ECFG_LIE] &13'h1bff &csr_wvalue[`CSR_ECFG_LIE]
                  |  ~csr_wmask[`CSR_ECFG_LIE] &13'h1bff &csr_ecfg_lie;
    end
end


//当写回阶段 (wb_ex) 发生地址异常 (ADE 或 ALE) 时，更新 BADV 寄存器的 VADDR 域:
如果是取指地址错误，存储 PC。
如果是其他地址错误，存储导致异常的虚拟地址。
//BADV 的 VADDR 域
always @(posedge clk) begin
    if(wb_ex &&(wb_ecode==`ECODE_ADE ||wb_ecode==`ECODE_ALE))begin
        csr_badv_vaddr <=(wb_ecode==`ECODE_ADE &&wb_esubcode==`ESUBCODE_ADEF) ?wb_pc :
                        wb_vaddr;
    end
end
//TID 寄存器控制定时器的唯一ID。
//TCFG 寄存器控制定时器的启用、周期性和初始计数值。
//TVAL 寄存器直接暴露定时器当前的计数值。
//定时器逻辑通过时钟的上升沿逐步递减计数器，并根据启用状态、周期性和初始化值执行计时
//
//TID 的数据域
always @(posedge clk) begin
    if(reset) begin
        csr_tid_tid <=32'b0;
    end
    else if(csr_we &&csr_num==`CSR_TID) begin
        csr_tid_tid <=csr_wmask[`CSR_TID_TID] &csr_wvalue[`CSR_TID_TID]
              |   ~csr_wmask[`CSR_TID_TID] &csr_tid_tid;
    end
end

//TCFG 的 EN、PERIODIC、INITVAL 域
always @(posedge clk) begin
    if(reset) begin
        csr_tcfg_en <=1'b0;
    end
    else if(csr_we &&csr_num==`CSR_TCFG) begin
        csr_tcfg_en <=csr_wmask[`CSR_TCFG_EN] &csr_wvalue[`CSR_TCFG_EN]
              |   ~csr_wmask[`CSR_TCFG_EN] &csr_tcfg_en;
    end

    if (csr_we &&csr_num==`CSR_TCFG) begin
        csr_tcfg_periodic <=csr_wmask[`CSR_TCFG_PERIODIC] &csr_wvalue[`CSR_TCFG_PERIODIC]
                      |  ~csr_wmask[`CSR_TCFG_PERIODIC] &csr_tcfg_periodic;
        csr_tcfg_initval <=csr_wmask[`CSR_TCFG_INITVAL] &csr_wvalue[`CSR_TCFG_INITVAL]
                      | ~csr_wmask[`CSR_TCFG_INITVAL] &csr_tcfg_initval;
```

```
    end
end

//TVAL 的 TIMEVAL 域
assign csr_tval_timeval =timer_cnt[31:0];

//定时器
assign tcfg_next_value =csr_wmask[31:0] &csr_wvalue[31:0]
                |  ~csr_wmask[31:0] &csr_tcfg_rvalue;
always @(posedge clk) begin
    if(reset) begin
        timer_cnt <=32'hffffffff;
    end
    else if(csr_we &&csr_num==`CSR_TCFG &&tcfg_next_value[`CSR_TCFG_EN]) begin
        timer_cnt <={tcfg_next_value[`CSR_TCFG_INITVAL], 2'b0};
    end
    else if(csr_tcfg_en &&timer_cnt !=32'hffffffff) begin
        if(timer_cnt ==32'b0 &&csr_tcfg_periodic) begin
            timer_cnt <={csr_tcfg_initval, 2'b0};
        end
        else begin
            timer_cnt <=timer_cnt -1'b1;
        end
    end
end

//TICLR 的 CLR 域
assign csr_ticlr_clr =1'b0;
```

计时器设计如下：该计时器实现了一个稳定递增的 64 位计数，每个时钟周期加 1。无论是低 32 位计数 (counter_vl) 还是高 32 位计数 (counter_vh)，都可以分别被访问，用于不同的逻辑模块中。reset 信号有效时，计数器会清零；当复位结束后，计数器恢复递增

```
+//计时器

reg [63:0] stable_counter;
wire [31:0] counter_vl;
wire [31:0] counter_vh;

always @(posedge clk) begin
    if (reset)
        stable_counter <=64'h0;
    else
        stable_counter <=stable_counter +64'h1;
end

assign counter_vl =stable_counter[31:0];
assign counter_vh =stable_counter[63:32];
```

取指地址错误异常 exc_adef：

如果程序计数器 PC 的低两位不为 00，即 PC 不是对齐到 4 字节边界时，发生取指地址错误（ADEF，Address Error Fetch），这种情况会触发 exc_adef

数据访问地址不对齐异常 exc_ale：

如果正在执行的数据访问类型不符合对齐要求（如 64 位访问要求地址按 8 字节对齐），则发生地址不对齐异常（ALE，Address Alignment Error）。这个条件依赖于 data_sram_type_tag_EX 和 data_sram_addr_EX，其中 data_sram_type_tag_EX 表示访问类型，data_sram_addr_EX 表示访问地址，EX_valid 表示执行阶段有效。

无效指令异常 exc_ine：

如果当前指令不是支持的有效指令集合中的一员（如加法、减法、跳转等），则发生无效指令异常（INE，Instruction Not Exist）。ID_valid 用于确保指令译码阶段有效。

断点异常 exc_break：

如果当前指令是断点指令(inst_break)，并且指令译码阶段有效，则触发断点异

系统调用异常 exc_syscall：如果当前指令是系统调用指令(inst_syscall)，并且指令译码阶段有效，则发生系统调用异常。

值得注意的是这些异常都要特殊处理一下触发异常的指令，对于 adef 干脆不能取的异常，我们直接在 inst_sram_en 型号上改进，给它赋值 ID_allowin !exc_adef。ale 与 ine 都要清空这条指令的有效位，在下一周期把下一级的 valid 清空。

```
//异常判断
assign exc_adef =pc[1:0] !=2'b00;
assign exc_ale =(data_sram_type_tag_EX[2] &&data_sram_addr_EX[0] !=1'b0
             || data_sram_type_tag_EX[1] &&data_sram_addr_EX[1:0] !=2'b0) &&EX_valid;
assign exc_ine =~(inst_add_w |inst_sub_w |inst_slt |inst_sltu |inst_nor |inst_and |inst_or |inst_xor
               | inst_slli_w |inst_srli_w |inst_srai_w |inst_addi_w |inst_ld_w |inst_st_w
               | inst_jirl |inst_b |inst_bl |inst_beq |inst_bne |inst_lu12i_w
               | inst_slti |inst_sltiu |inst_andi |inst_ori |inst_xori |inst_sll_w |inst_srl_w |inst_sra_w |inst_pcaddu12i
               | inst_mul_w |inst_mulh_w |inst_mulh_wu |inst_div_w |inst_mod_w |inst_div_wu |inst_mod_wu
               | inst_ld_b |inst_ld_h |inst_ld_bu |inst_ld_hu |inst_st_b |inst_st_h
               | inst_blt |inst_bge |inst_bltu |inst_bgeu
               | inst_csrrd |inst_csrwr |inst_csrxchg
               | inst_ertn |inst_syscall |inst_break
               | inst_rdcntvl_w |inst_rdcntvh_w |inst_rdcntid_w) &&ID_valid;
assign exc_break =inst_break &&ID_valid;
assign exc_syscall =inst_syscall &&ID_valid;

assign exc_at_ID =exc_break ||exc_syscall ||exc_adef ||exc_ine;
assign exc_at_EX =exc_ale;

assign wb_ex =exc_at_ID ||exc_at_EX ||has_int;
assign wb_pc =(exc_adef) ?pc :
            (!br_taken_EX &&ID_valid) ?pc_ID :
             pc_EX;
assign ertn_flush =inst_ertn &&ID_valid;
assign wb_ecode =has_int  ? `ECODE_INT :
                 exc_adef  ? `ECODE_ADE :
                 exc_ale   ? `ECODE_ALE :
                 exc_syscall ?`ECODE_SYS :
                 exc_break ? `ECODE_BRK :
                 exc_ine   ? `ECODE_INE :
                 6'h0;
assign wb_esubcode =9'h0;
```

对于流水级的控制，我们着重在清空 valid 信号上：对于 ID 阶段在 ID 或者 EX 阶段发生，也直接置零；

值得注意的是在 EX 阶段，添加了如果发生无效指令预测，也立即将 vaild 置零；在 EXE 阶段，如果发生地址非对齐异常的问题，也立即将 vaild 置零。

```
//流水级控制
always @(posedge clk) begin
    if (reset)
        ID_valid <=1'b0;
    else if ((br_taken_ID ||br_taken_EX) &&ID_allowin)//分支跳转则把预取的错误指令取消
        ID_valid <=1'b0;
    else if(ID_allowin)
        ID_valid <=1'b1;
end
always @(posedge clk) begin
    if (reset)
        EX_valid <=1'b0;
    else if((br_taken_EX ||exc_ine) &&EX_allowin )//EX跳转或者无效指令则取消
        EX_valid <=1'b0;
    else if(EX_allowin)
```

```
16          EX_valid <=ID_valid &&ID_readygo;
17  end
18  always @(posedge clk) begin
19      if (reset)
20          MEM_valid <=1'b0;
21      else if(exc_ale &&MEM_allowin)//地址非对齐异常则取消该访存指令
22          MEM_valid <=1'b0;
23      else if(MEM_allowin)
24          MEM_valid <=EX_valid &&EX_readygo;
25  end
26  always @(posedge clk) begin
27      if (reset)
28          WB_valid <=1'b0;
29      else if(WB_allowin)
30          WB_valid <=MEM_valid &&MEM_readygo;
31  end
```

三、 实验分工

- **实践十二，十三。**
  - 刘钰：添加系统调用异常支持；
  - 孙广润：添加其他异常支持；
  - 蔡合森，孙广润：写实验报告；