浙江大学

本科实验报告

课程名称：计算机体系结构

姓　　名：郑乔尹

学　　院：计算机科学与技术学院

　　系：计算机科学与技术系

专　　业：计算机科学与技术

学　　号：3210104169

指导教师：姜晓红

2023 年　10　　月　　27　　日

浙江大学实验报告

课程名称：计算机体系结构 实验类型：综合

实验项目名称：Lab2：Pipelined CPU supporting exception & interrupt

学生姓名：郑乔尹 专业：计算机科学与技术 学号：3210104169

同组学生姓名：无 指导老师：姜晓红

实验地点：曹西 301 实验日期：2023 年 10 月 24 日

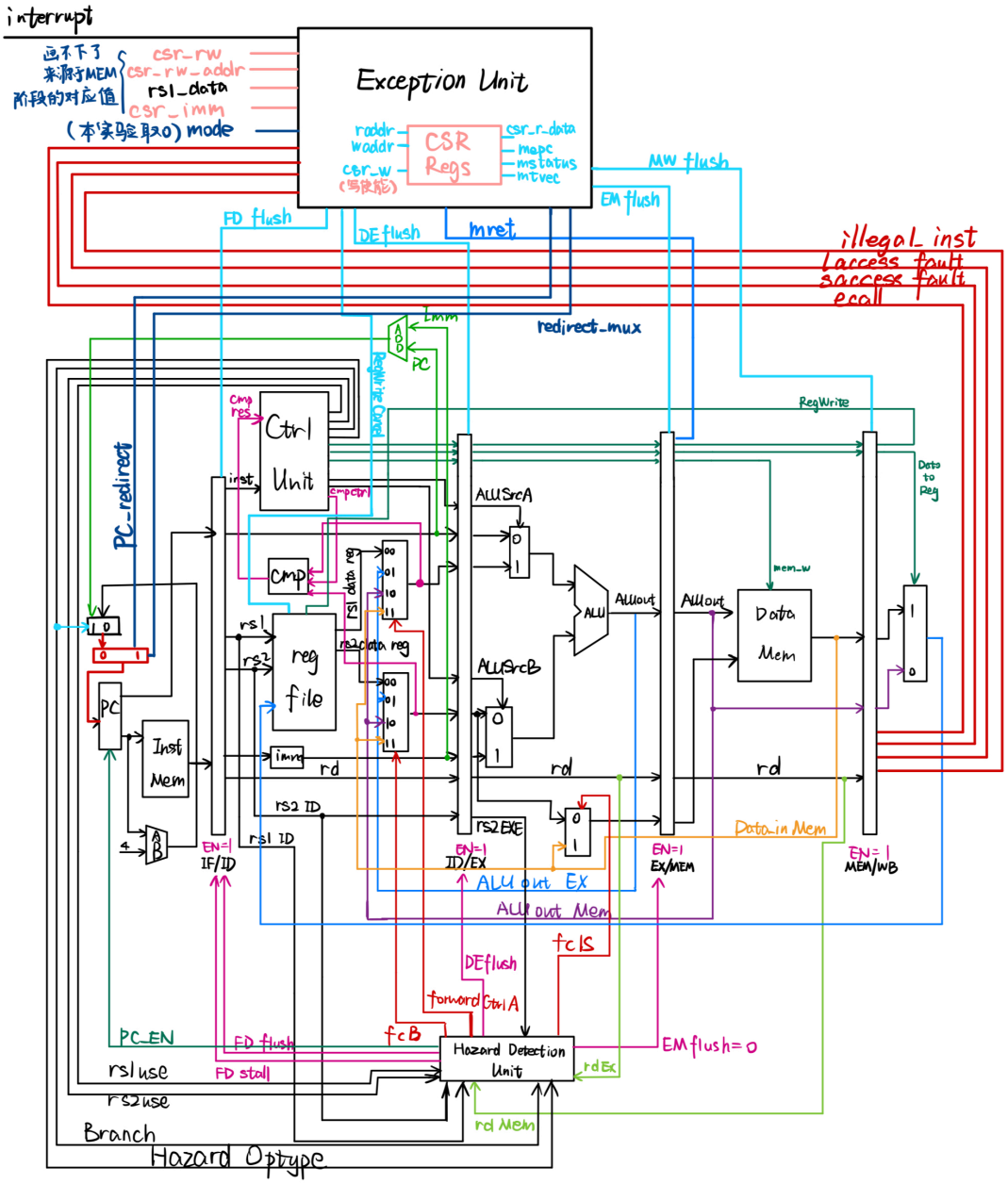
一、 实验目的和要求

Task 1: 写出本次实验的目的与要求 (5 points)

1. 理解 CPU 中断与异常的原理以及它的处理流程
2. 掌握带中断与异常的流水线 CPU 的设计方法
3. 掌握带中断与异常的流水线 CPU 的程序验证方法

二、 实验内容和原理

Task 2: 画出本次实验实现的电路图和异常模块的状态机示意图。（可以使用 PPT 上的线路图进行修改，但是必须和自己的实现保持一致。如果不一致，本题将不给分） (10 points)



Task 3: 请给出 CSR 寄存器模块的代码并加以解释 (5 points)

模块输入与输出信号：

输入：时钟与复位信号，raddr(读地址)，waddr(写地址)，csr_w(写使能)，csr_wsc_mode(中断响应模式，本实验中都发送到同一个中断处理程序，故实际上全部置 0)

输出：rdata(读到的寄存器值)，mstatus，mepc，mtvec 均为对应寄存器的值，引出信号方便 ExceptionUnit 直接取用，而不用再等半个周期。

```
module CSRRegs(
    input clk, rst,
    input[11:0] raddr, waddr,
```

```
    input[31:0] wdata,
    input csr_w,
    input[1:0] csr_wsc_mode,
    output[31:0] rdata,
    output[31:0] mstatus,
    // add mepc and mtvec read
    output[31:0] mepc,
    output[31:0] mtvec
);
```

地址映射：

由于实验中只有 16 个 CSR 寄存器，但是 CSR 寄存器地址实际上是 12bit，即有 4096 个 CSR 寄存器，故这里进行地址映射：

映射逻辑为：addr[11]-addr[7]以及 addr[5]-addr[3]代表地址是否有效，addr[6]与 addr[2]-addr[0]拼接作为映射后的地址。

```
    wire raddr_valid = raddr[11:7] == 5'h6 && raddr[5:3] == 3'h0;
    wire[3:0] raddr_map = (raddr[6] << 3) + raddr[2:0];
    wire waddr_valid = waddr[11:7] == 5'h6 && waddr[5:3] == 3'h0;
    wire[3:0] waddr_map = (waddr[6] << 3) + waddr[2:0];
```

引出对应寄存器信号：

```
    assign mstatus = CSR[0];
    assign mepc = CSR[9];
    assign mtvec = CSR[5];

    assign rdata = CSR[raddr_map];
```

寄存器初始化以及寄存器写入逻辑：

```
always@(posedge clk or posedge rst) begin
        if(rst) begin
            CSR[0] <= 32'h88;
            CSR[1] <= 0;
            CSR[2] <= 0;
            CSR[3] <= 0;
            CSR[4] <= 32'hfff;
            CSR[5] <= 0;
            CSR[6] <= 0;
            CSR[7] <= 0;
            CSR[8] <= 0;
            CSR[9] <= 0;
            CSR[10] <= 0;
            CSR[11] <= 0;
            CSR[12] <= 0;
            CSR[13] <= 0;
```

```
            CSR[14] <= 0;
            CSR[15] <= 0;
        end
        else if(csr_w) begin
            case(csr_wsc_mode)
                2'b01: CSR[waddr_map] = wdata;
                2'b10: CSR[waddr_map] = CSR[waddr_map] | wdata;
                2'b11: CSR[waddr_map] = CSR[waddr_map] & ~wdata;
                default: CSR[waddr_map] = wdata;
            endcase
        end
    end
```

Task 4: 请给出异常触发的逻辑以及解释 (15 points)

异常分为 ecall, l_access_fault, s_access_fault, illegal_inst 四种，在 WB 阶段检测到对应异常后将对应信号传输给 ExceptionUnit，ExceptionUnit 根据异常类型，准备中断产生原因 mcause 的值，并将状态从 IDLE 转化为 STATE_MSTATUS，在此过程中把从 CSRRegs 模块中读到的 mtvec 值作为 PC 重定向的目标地址，并将目前正在执行的流水线 flush，开始运行中断处理程序。在 STATE_MSTATUS 阶段，记录 MIE 位至 MPIE 位，关闭中断使能，停止外部中断的触发，状态转换至 STATE_MCAUSE，然后写入 mcause，随后回到 IDLE 状态，等待 mret。

Task 5: 请给出中断触发的逻辑以及解释，只需要说明与上一题内容的不同之处 (10 points)

通过外部中断信号触发，本实验中其信号连接至 SW[12]，由于中断是外部信号突然触发，不像异常那样可以在指令进入 WB 阶段就准备好异常信号（进入 WB 阶段就准备好异常信号意味着 CSR 写地址可以被提前准备好），故我让其空置了一个状态，并保持外部中断信号，以准备 CSR 写地址，从而将需要的 mepc 正确写入 mepc 对应的地址。

Task 6: 请给出实现 mret 的代码以及解释 (5 points)

为了实现 mret，我的实现是在中断/异常触发并写完所需的 CSR 寄存器后，回到 IDLE 状态，重新检测是否产生 mret，当接收到 mret 信号且处于 IDLE 状态时，开始 mret 流程，读取之前存储的 mepc，回到中断/异常前代码的执行地址（实际上异常需要重新执行触发异常的代码，但是由于实验中没有修复触发异常的代

码的相应实现，采取了直接在异常处理程序中给 mepc+4 的方法，软件跳过了该异常代码，同时这也将导致外部中断最后跳回的地址比预期大 4，即跳过了一条指令）。

检测处于 IDLE 状态时是否有 mret 信号，有则写 mstatus，从 mstatus[MPIE]恢复 mstatus[MIE]:

```
if (STATE_IDLE) begin
    if (is_exception) begin // exception, save PC, write MSTATUS, read MTVEC
        execption_delay <= 1'b1; // delay for a cycle, wait for mstatus[MIE]
disable
        csr_w <= 1'b1;
        csr_wsc <= 2'b00;
        csr_waddr <= MEPC_addr;
        csr_wdata <= EPC;
    end
    else if (mret) begin // mret, write MSTATUS, PC <= MEPC
        csr_w <= 1'b1;
        csr_wsc <= 2'b00;
        csr_waddr <= MSTATUS_addr;
        csr_wdata <= {mstatus[31:8], 1'b1, mstatus[6:4], mstatus[7],
mstatus[2:0]};
    end
end
```

PC 重定向：

如果当前为 mret，选取 CSRRegs 模块中读到的 mepc 作为 PC 重定向的目标地址。同时将重定向使能信号置 1.

```
assign PC_redirect = (STATE_IDLE && is_interrupt) || (STATE_IDLE &&
is_exception) ? mtvec : ((STATE_IDLE && mret) ? mepc : csr_r_data_out);
assign redirect_mux = jump;
```

其中 jump 代表中断指令触发：

```
    wire jump = (STATE_IDLE && is_interrupt) || (STATE_IDLE && is_exception)
|| (STATE_IDLE && mret);
```

Task 7: 请给出实现 6 个 CSR 指令的代码以及解释 (5 points)

在未触发任何中断或者已经写完 CSR 寄存器，回到 IDLE 状态时，通过将 CSRRegs 输入信号设置为 ExceptionUnit 输入的对应信号实现 CSR 指令的执行：

注意，由于我的实现中，为了正确写入 mepc 的值(在外部中断触发时，由于上升沿触发，无法及时更新 csr 寄存器写地址，故通过在下一个上升沿写 mepc 来解决这一问题)，我通过 saved_int 寄存器保存了外部中断信号，所以需要判断

当时是否在外部中断处理过程中，如果不在（saved_int == 0），则直接从模块输入中获取对应信号即可，同时需要根据立即数选择器确定写入的值是寄存器值（csr_w_imm_mux == 0）还是立即数（csr_w_imm_mux == 1）。

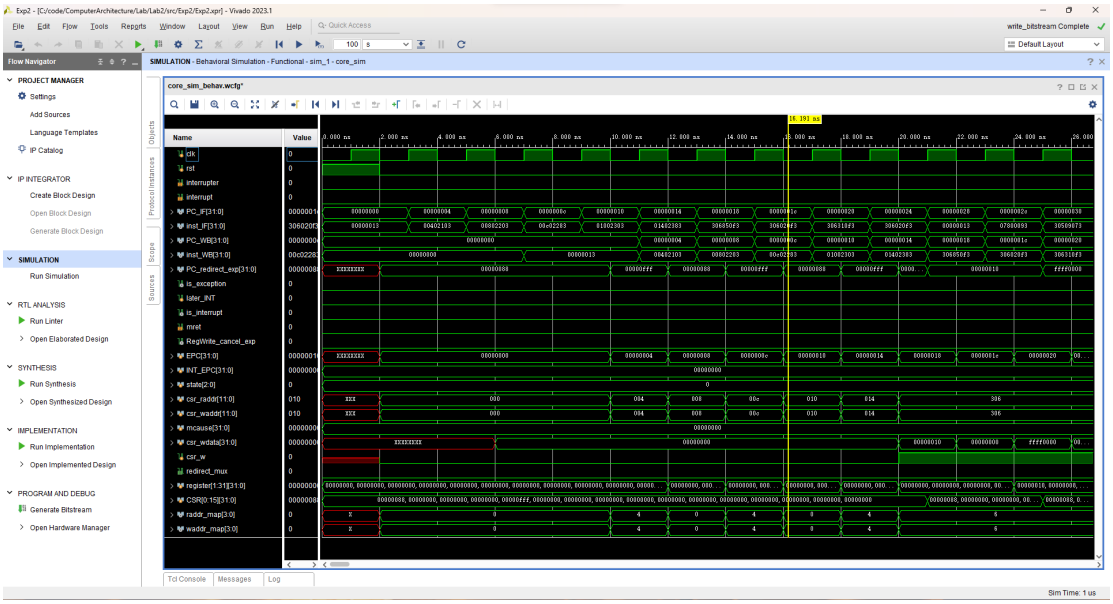
```
    if(saved_int == 1'b0) begin
            csr_raddr <= csr_rw_addr_in;
            csr_w <= csr_rw_in;
            csr_waddr <= csr_rw_addr_in;
        end
        else begin
            csr_raddr <= csr_rw_addr_in;
            csr_w <= 1'b1;
            csr_waddr <= MEPC_addr;
            saved_int <= 1'b0;
        end

        if(csr_w_imm_mux) begin
            csr_wdata <= {27'b0, csr_w_data_imm[4:0]};
        end
        else begin
            csr_wdata <= csr_w_data_reg;
        end

        csr_wsc <= csr_wsc_mode_in;
```

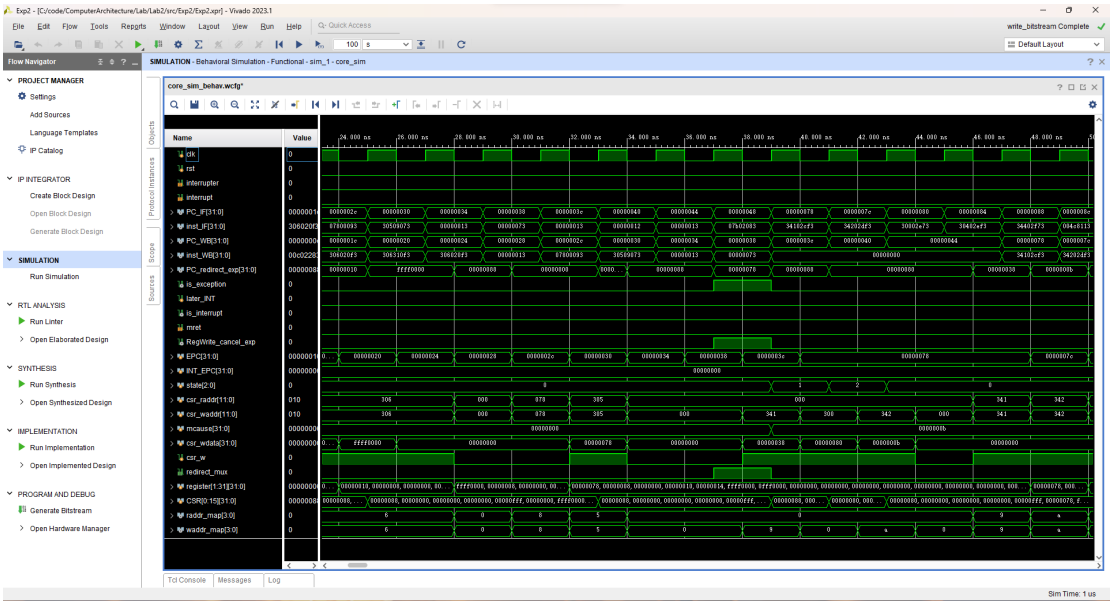
三、 实验过程和数据记录及结果分析

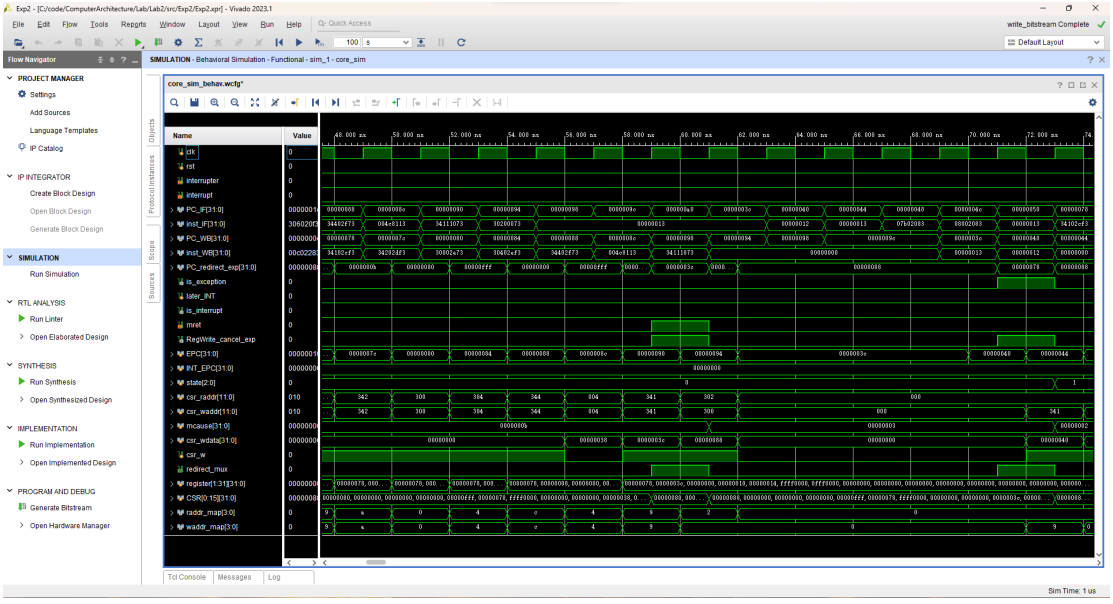
仿真图片应完整包含时间信息和信号名称。

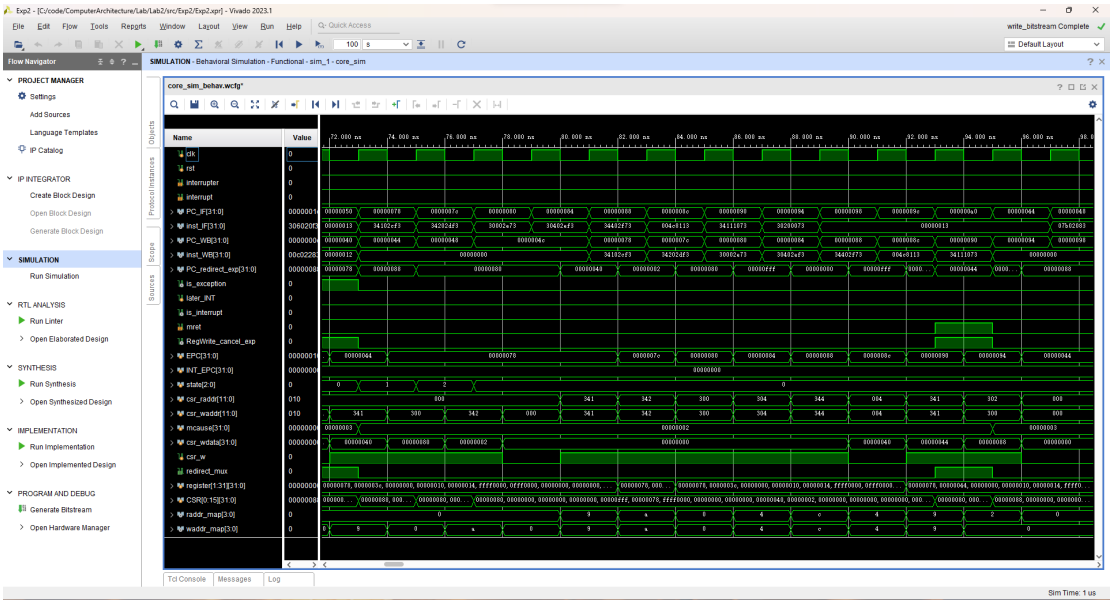
对仿真的解释示例：XXXns，X 信号变为 X，由于 XXX，导致 X 信号变为 XXX，……，我们发现 X 被 forward 到了 X。

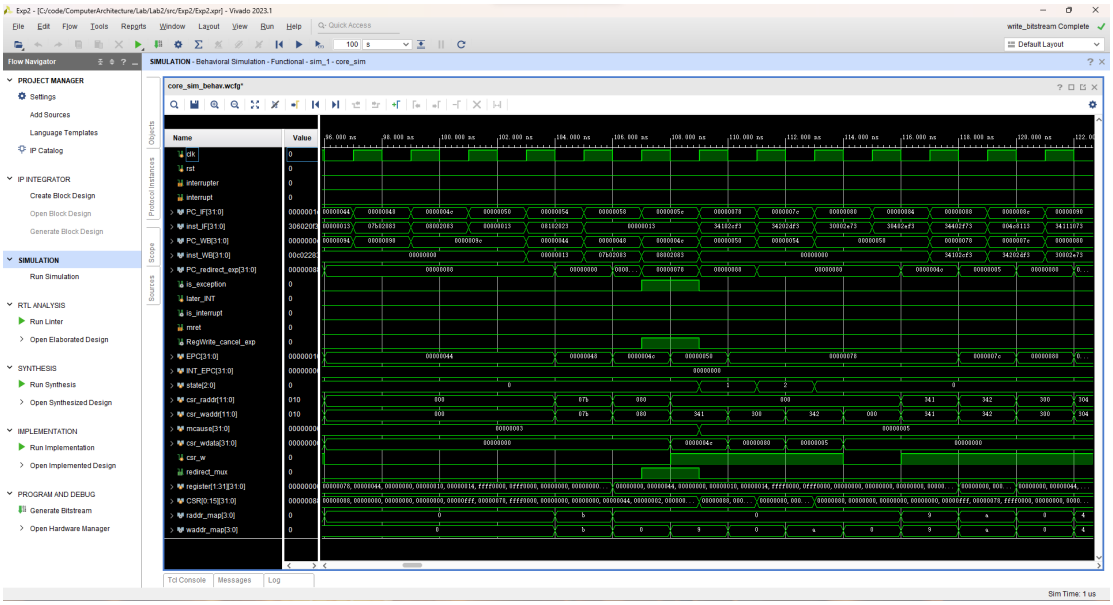
Task 8: 请给出本次实验仿真的完整截图 (5 points)

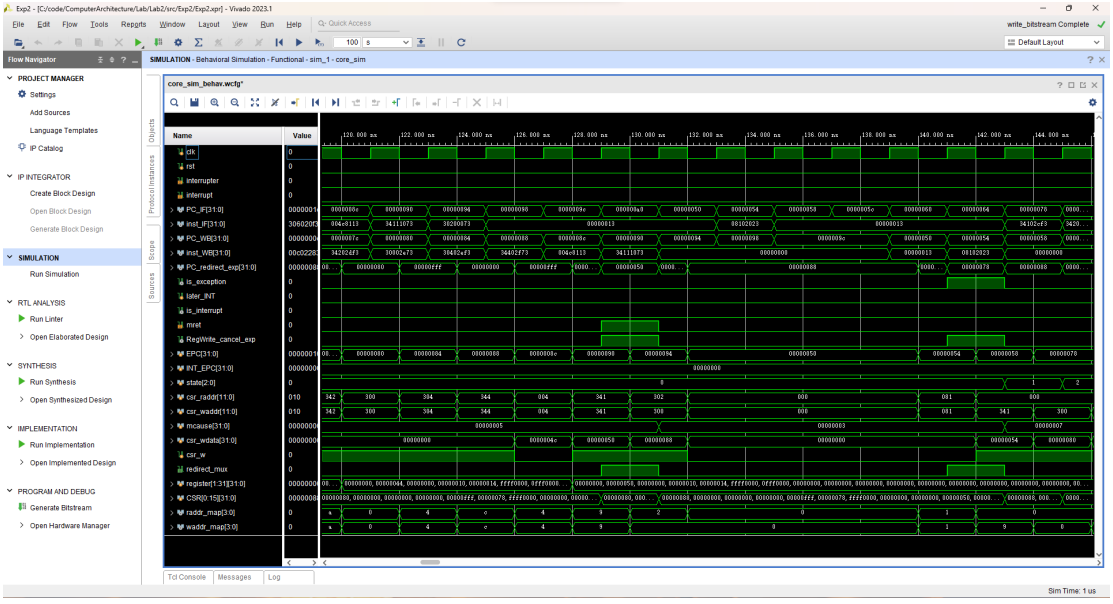

Exp2 - [C:/code/ComputerArchitecture/Lab/Lab2/src/Exp2/Exp2.xpr] - Vivado 2023.1
File Edit Flow Tools Reports Window Layout View Run Help
write_bitstream Complete
Default Layout
Flow Navigator
SIMULATION - Behavioral Simulation - Functional - sim_1 - core_sim
core_sim_behav.wcfg*
Name
Value
Tcl Console
Messages
Log
Sim Time: 1 us

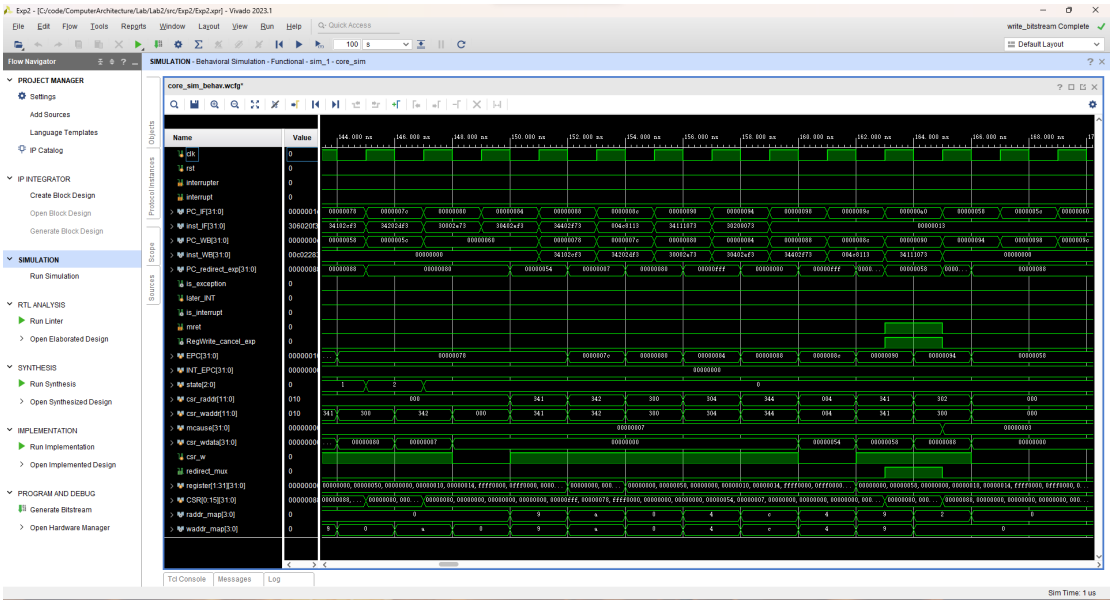

Exp2 - [C:/code/ComputerArchitecture/Lab/Lab2/src/Exp2/Exp2.xpr] - Vivado 2023.1
File Edit Flow Tools Reports Window Layout View Run Help
write_bitstream Complete
Default Layout
Flow Navigator
SIMULATION - Behavioral Simulation - Functional - sim_1 - core_sim
core_sim_behav.wcfg*
Name
Value
Tcl Console
Messages
Log
Sim Time: 1 us

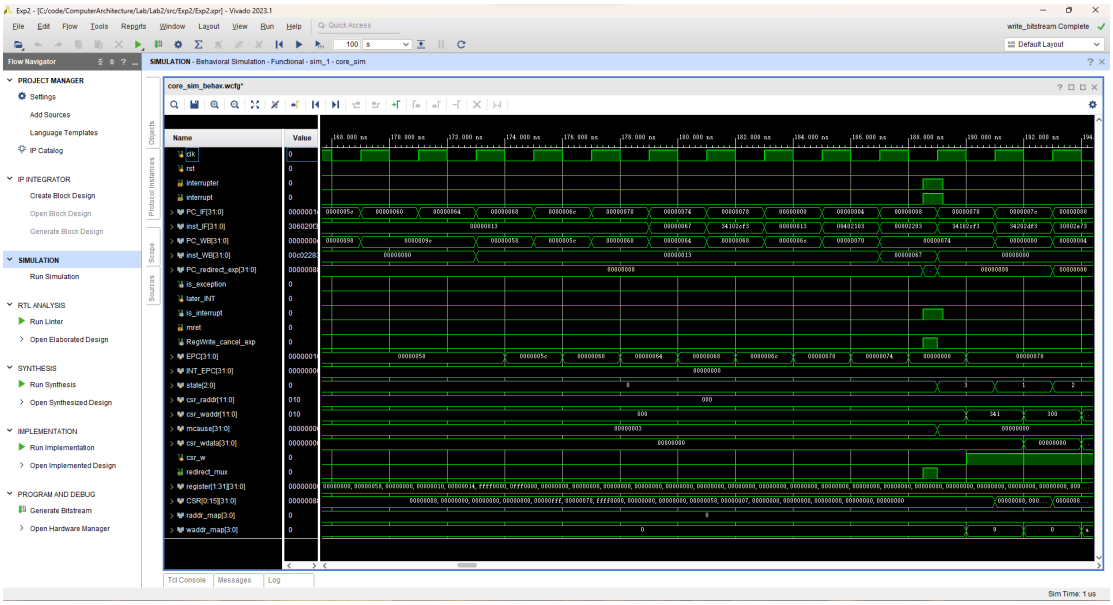

Exp2 - [C:/code/ComputerArchitecture/Lab/Lab2/src/Exp2/Exp2.xpr] - Vivado 2023.1
File Edit Flow Tools Reports Window Layout View Run Help
write_bitstream Complete
Default Layout
Flow Navigator
SIMULATION - Behavioral Simulation - Functional - sim_1 - core_sim
core_sim_behav.wcfg*
Name
Value
Tcl Console
Messages
Log
Sim Time: 1 us

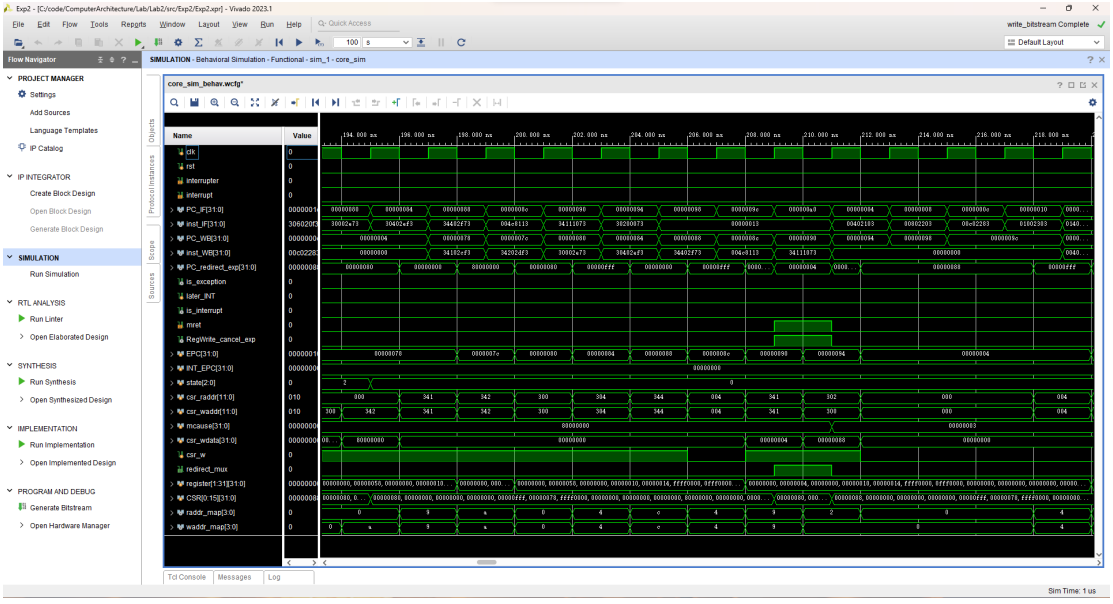

Exp2 - [C:/code/ComputerArchitecture/Lab/Lab2/src/Exp2/Exp2.xpr] - Vivado 2023.1
File Edit Flow Tools Reports Window Layout View Run Help
write_bitstream Complete
Default Layout
Flow Navigator
SIMULATION - Behavioral Simulation - Functional - sim_1 - core_sim
core_sim_behav.wcfg*
Name
Value
Tcl Console
Messages
Log
Sim Time: 1 us

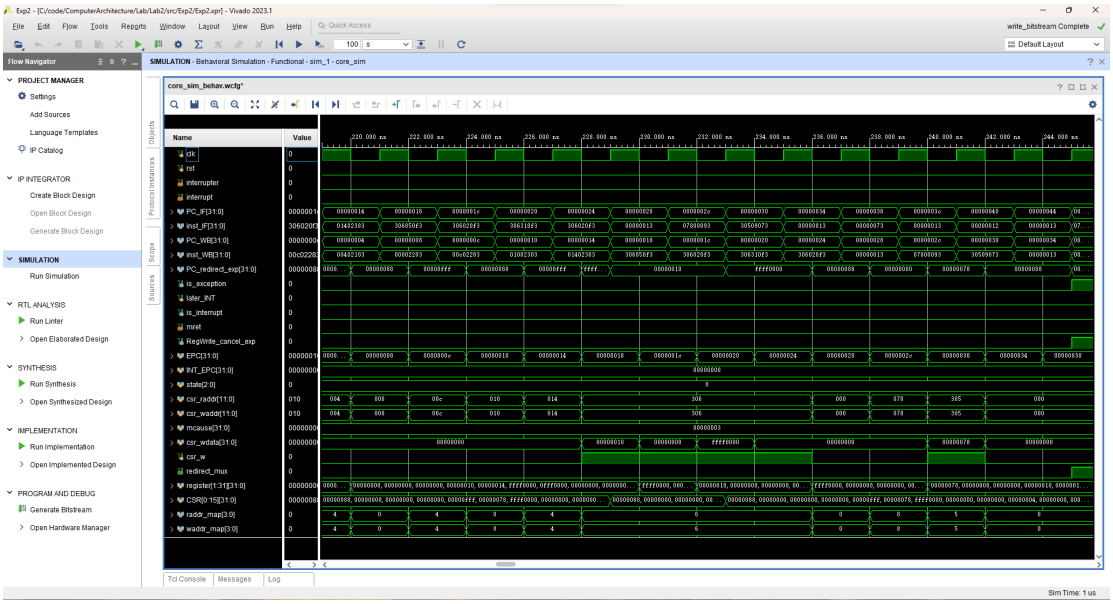

Exp2 - [C:/code/ComputerArchitecture/Lab/Lab2/src/Exp2/Exp2.xpr] - Vivado 2023.1
File Edit Flow Tools Reports Window Layout View Run Help
write_bitstream Complete
Default Layout
Flow Navigator
SIMULATION - Behavioral Simulation - Functional - sim_1 - core_sim
core_sim_behav.wcfg*
Name
Value
Tcl Console
Messages
Log
Sim Time: 1 us


Exp2 - [C:/code/ComputerArchitecture/Lab/Lab2/src/Exp2/Exp2.xpr] - Vivado 2023.1
File Edit Flow Tools Reports Window Layout View Run Help
write_bitstream Complete
Default Layout
Flow Navigator
SIMULATION - Behavioral Simulation - Functional - sim_1 - core_sim
core_sim_behav.wcfg*
Name
Value
Tcl Console
Messages
Log
Sim Time: 1 us

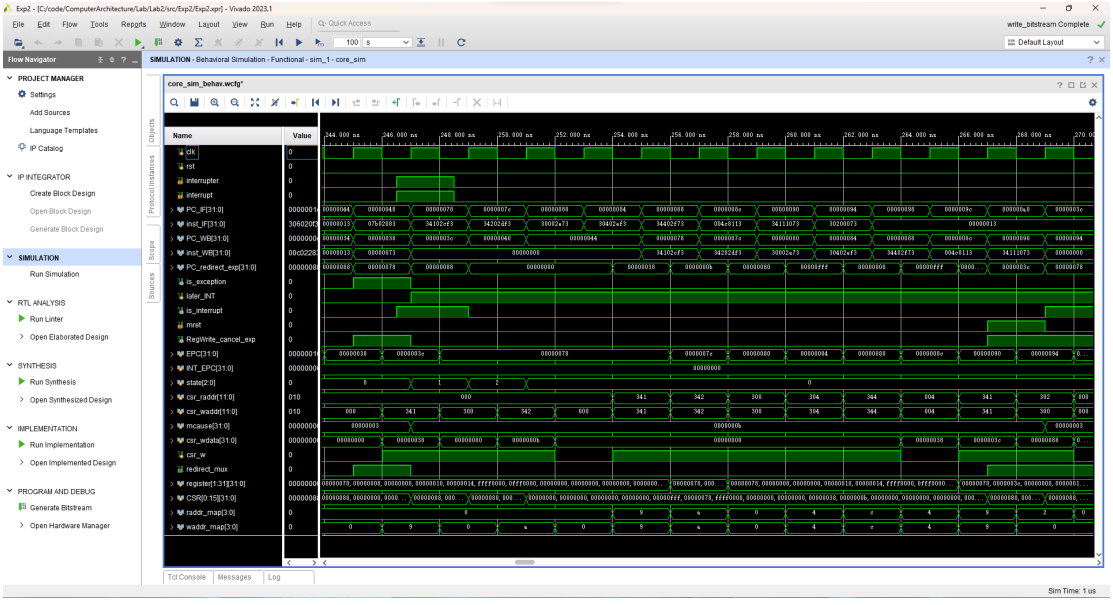

Exp2 - [C:/code/ComputerArchitecture/Lab/Lab2/src/Exp2/Exp2.xpr] - Vivado 2023.1
File Edit Flow Tools Reports Window Layout View Run Help
write_bitstream Complete
Default Layout
Flow Navigator
SIMULATION - Behavioral Simulation - Functional - sim_1 - core_sim
core_sim_behav.wcfg*
Name
Value
clk
rst
interrupter
interrupt
PC_IF[31:0]
inst_IF[31:0]
PC_WB[31:0]
inst_WB[31:0]
PC_redirect_exp[31:0]
is_exception
later_INT
is_interrupt
mret
RegWrite_cancel_exp
EPC[31:0]
INT_EPC[31:0]
state[2:0]
csr_radd[11:0]
csr_wadd[11:0]
mcause[31:0]
csr_wdata[31:0]
csr_w
redirect_mux
register[1:31][31:0]
CSR[0:15][31:0]
raddr_map[3:0]
waddr_map[3:0]
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation
RTL ANALYSIS
Run Linter
Open Elaborated Design
SYNTHESIS
Run Synthesis
Open Synthesized Design
IMPLEMENTATION
Run Implementation
Open Implemented Design
PROGRAM AND DEBUG
Generate Bitstream
Open Hardware Manager
Tcl Console
Messages
Log
Sim Time: 1 us

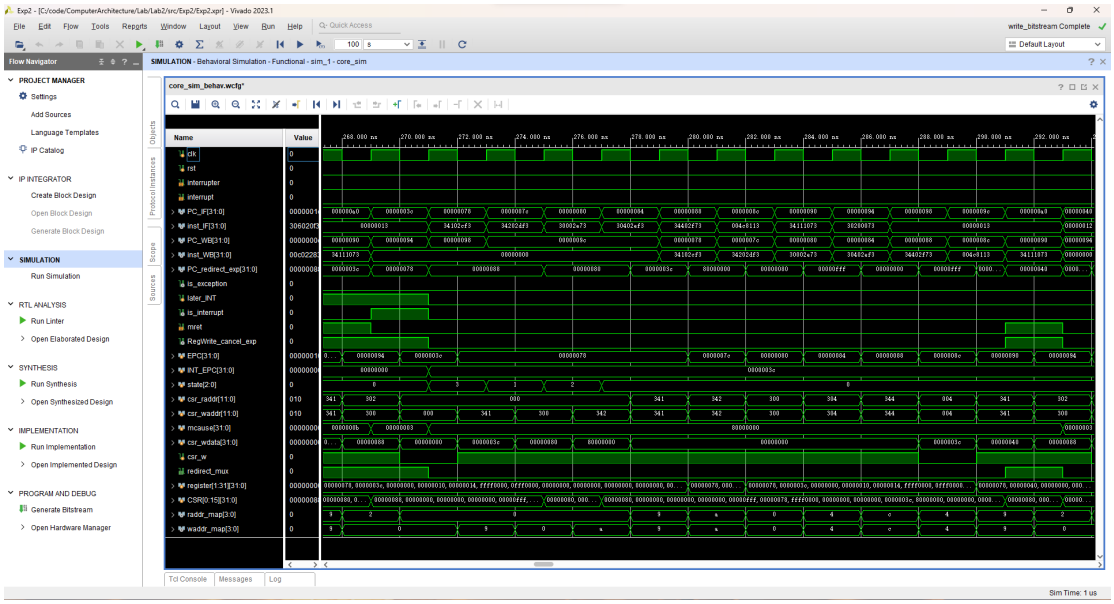

Exp2 - [C:/code/ComputerArchitecture/Lab/Lab2/src/Exp2/Exp2.xpr] - Vivado 2023.1
File Edit Flow Tools Reports Window Layout View Run Help
write_bitstream Complete
Default Layout
Flow Navigator
SIMULATION - Behavioral Simulation - Functional - sim_1 - core_sim
core_sim_behav.wcfg*
Name
Value
clk
rst
interrupter
interrupt
PC_IF[31:0]
inst_IF[31:0]
PC_WB[31:0]
inst_WB[31:0]
PC_redirect_exp[31:0]
is_exception
later_INT
is_interrupt
mret
RegWrite_cancel_exp
EPC[31:0]
INT_EPC[31:0]
state[2:0]
csr_radd[11:0]
csr_wadd[11:0]
mcause[31:0]
csr_wdata[31:0]
csr_w
redirect_mux
register[1:31][31:0]
CSR[0:15][31:0]
raddr_map[3:0]
waddr_map[3:0]
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation
RTL ANALYSIS
Run Linter
Open Elaborated Design
SYNTHESIS
Run Synthesis
Open Synthesized Design
IMPLEMENTATION
Run Implementation
Open Implemented Design
PROGRAM AND DEBUG
Generate Bitstream
Open Hardware Manager
Tcl Console
Messages
Log
Sim Time: 1 us


Exp2 - [C:/code/ComputerArchitecture/Lab/Lab2/src/Exp2/Exp2.xpr] - Vivado 2023.1
File Edit Flow Tools Reports Window Layout View Run Help
write_bitstream Complete
Default Layout
Flow Navigator
SIMULATION - Behavioral Simulation - Functional - sim_1 - core_sim
core_sim_behav.wcfg*
Name
Value
clk
rst
interrupter
interrupt
PC_IF[31:0]
inst_IF[31:0]
PC_WB[31:0]
inst_WB[31:0]
PC_redirect_exp[31:0]
is_exception
later_INT
is_interrupt
mret
RegWrite_cancel_exp
EPC[31:0]
INT_EPC[31:0]
state[2:0]
csr_radd[11:0]
csr_wadd[11:0]
mcause[31:0]
csr_wdata[31:0]
csr_w
redirect_mux
register[1:31][31:0]
CSR[0:15][31:0]
raddr_map[3:0]
waddr_map[3:0]
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation
RTL ANALYSIS
Run Linter
Open Elaborated Design
SYNTHESIS
Run Synthesis
Open Synthesized Design
IMPLEMENTATION
Run Implementation
Open Implemented Design
PROGRAM AND DEBUG
Generate Bitstream
Open Hardware Manager
Tcl Console
Messages
Log
Sim Time: 1 us



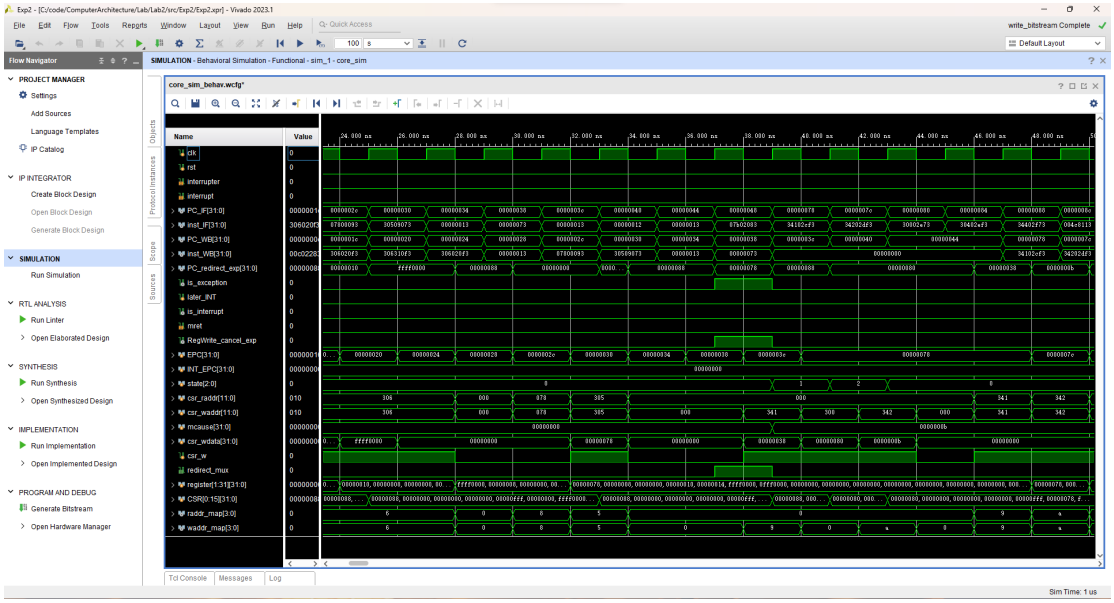
以下是我自行添加的仿真，用于测试异常和外部中断同时发生的情形：





Task 9: 请给出一个实现 ecall 功能部分仿真的高清图片，并对涉及到的信号加以详细解释 (10 points)

ecall 指令进入 WB 阶段，进入异常处理程序：

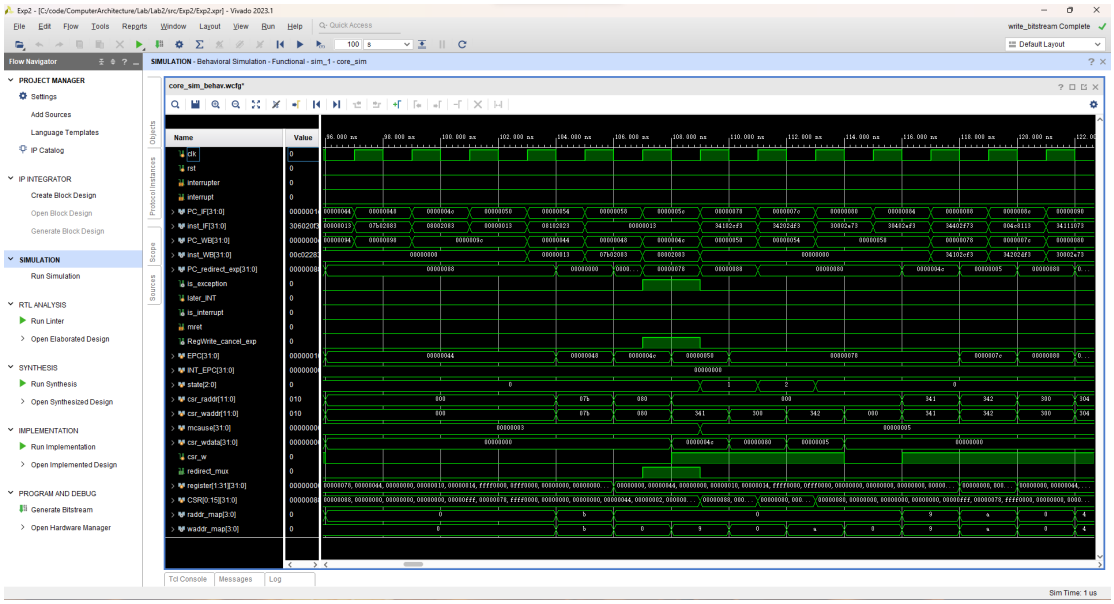


在 PC_IF = 0x48（37000ns）时，可以看到，此时的 PC_WB = 0x38, inst_WB = 00000073(ecall),即此时 ecall 指令进入 WB 阶段,此时 ExceptionUnit 接收到 ecall_m 的高电平信号，根据此信号更新异常处理模块中 mcause 寄存器的值为 0xb，等待后续写入；此时状态机处于 IDLE 状态（state=0），且 ecall_m 信号为高，说明发生异常（is_exception=1）,需要进行跳转，在将状态机转到 MSTATUS 状态（state=1，39000ns）的过程中，对当前流水线进行清空，所有 flush 信号置 1，RegWrite_cancel 置 1，PC 重定向使能（redirect_mux=1）并将 PC 重定向地址（PC_redirect）设置为 mtvec 寄存器

中读取到的值（由于我修改了 CSR 寄存器模块，引出了 mtvec 寄存器的信号，这里可以通过该信号获取 mtvec 寄存器的值），同时，将当前的 PC 保存进 mepc（可以看到 csr_waddr 为 341，csr_wdata 为 0x38）。接下来进入 MSTATUS 状态(state=1)，该阶段写 mstatus 寄存器(可以看到 csr_waddr 为 300，csr_wdata 为 0x80，mie 位为 0)，保存 mstatus[mie]至 mstatus[mpie]，并将 mstatus[mie]置 0，禁止外部中断的发生，然后转换到 MCAUSE 状态（state=2）。在 MCAUSE 状态，写入之前已经保存好的 mcause 值（csr_addr=342，csr_wdata=mcause=0xb），随后状态回到 IDLE，继续执行中断处理程序。

Task 10: 请给出一个触发 load/store access fault 部分仿真的高清图片，并对涉及到的信号加以详细解释 (10 points)

L_access fault:

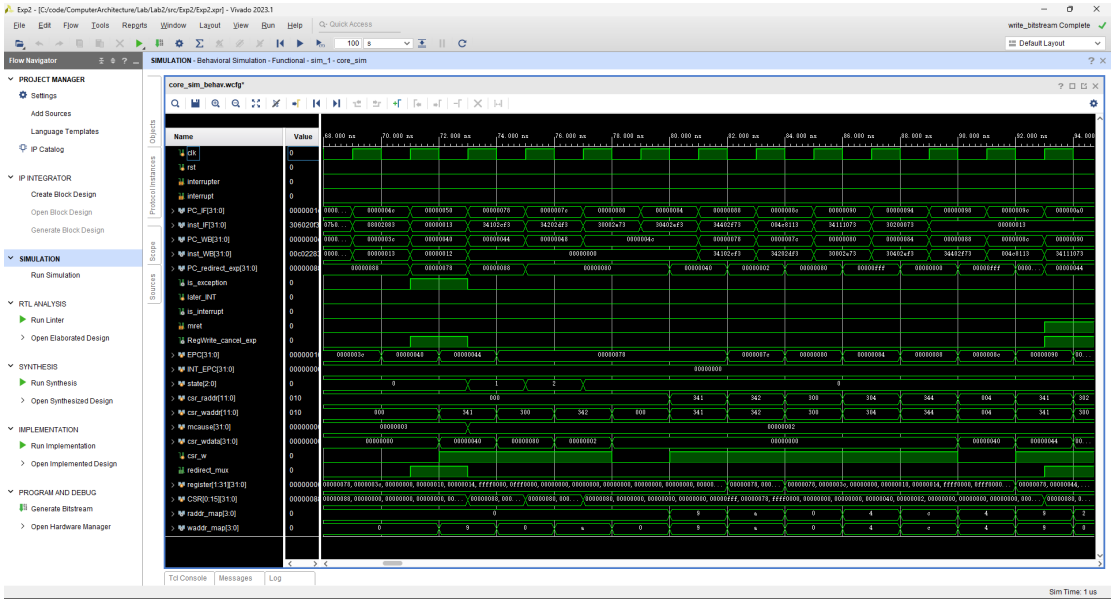


在 PC_IF = 0x5c（107000ns）时，可以看到，此时的 PC_WB = 0x4c，inst_WB = 08002083(lw 地址 128 处的值)，即此时该指令进入 WB 阶段，此时 ExceptionUnit 接收到 l_access_fault 的高电平信号，根据此信号更新异常处理模块中 mcause 寄存器的值为 0x5，等待后续写入；此时状态机处于 IDLE 状态（state=0），且 l_access_fault 信号为高，说明发生异常（is_exception=1），需要进行跳转，在将状态机转到 MSTATUS 状态（state=1）的过程中，对当前流水线进行清空，所有 flush 信号置 1，RegWrite_cancel 置 1，PC 重定向使能（redirect_mux=1）并将 PC 重定向地址（PC_redirect）设置为 mtvec 寄存器中读取到的值（由于我修改了 CSR 寄存器模块，引出了 mtvec 寄存器的信号，这里可以通过该信号获取 mtvec 寄存器的值），同时，将当前的 PC 保存进 mepc（可以看到

csr_waddr 为 341，csr_wdata 为 0x4c）。接下来进入 MSTATUS 状态(state=1，109000ns)，该阶段写 mstatus 寄存器（可以看到 csr_waddr 为 300，csr_wdata 为 0x80，mie 位为 0），保存 mstatus[mie]至 mstatus[mpie]，并将 mstatus[mie]置 0，禁止外部中断的发生，然后转换到 MCAUSE 状态（state=2）。在 MCAUSE 状态，写入之前已经保存好的 mcause 值（csr_addr=342，csr_wdata=mcause=0x5），随后状态回到 IDLE，继续执行中断处理程序。

S_access_fault 部分的 mcause 不同，为 0x7。

Task 11: 请给出一个触发 Illegal instruction 部分仿真的高清图片，并对涉及到的信号加以详细解释 (10 points)

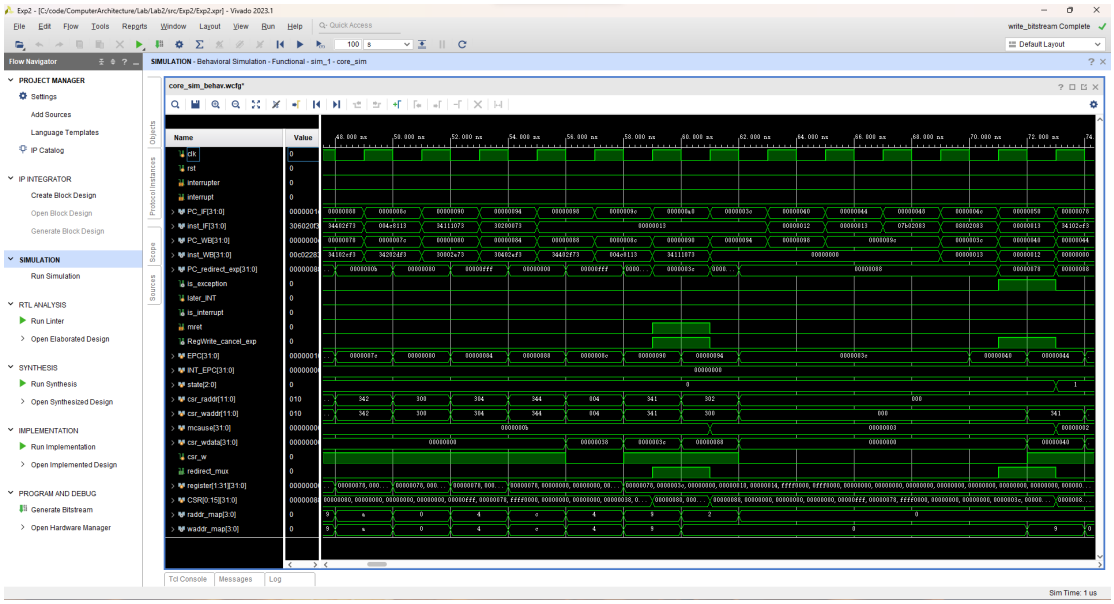


在 PC_IF = 0x50（71000ns）时，可以看到，此时的 PC_WB = 0x40, inst_WB = 00000012(illegal addi x0,x0,0)，即此时非法指令进入 WB 阶段，此时 ExceptionUnit 接收到 illegal_inst 的高电平信号，根据此信号更新异常处理模块中 mcause 寄存器的值为 0x2，等待后续写入；此时状态机处于 IDLE 状态（state=0），且 illegal_inst 信号为高，说明发生异常（is_exception=1），需要进行跳转，在将状态机转到 MSTATUS 状态（state=1）的过程中，对当前流水线进行清空，所有 flush 信号置 1，RegWrite_cancel 置 1，PC 重定向使能（redirect_mux=1）并将 PC 重定向地址（PC_redirect）设置为 mtvec 寄存器中读取到的值（由于我修改了 CSR 寄存器模块，引出了 mtvec 寄存器的信号，这里可以通过该信号获取 mtvec 寄存器的值），同时，将当前的 PC 保存进 mepc（可以看到 csr_waddr 为 341，csr_wdata 为 0x40）。接下来进入 MSTATUS 状态(state=1，73000ns)，该阶段写 mstatus 寄存器（可以看到 csr_waddr 为 300，csr_wdata 为 0x80，mie 位为 0），

保存 mstatus[mie]至 mstatus[mpie]，并将 mstatus[mie]置 0，禁止外部中断的发生，然后转换到 MCAUSE 状态（state=2）。在 MCAUSE 状态，写入之前已经保存好的 mcause 值（csr_addr=342，csr_wdata=mcause=0x2），随后状态回到 IDLE，继续执行中断处理程序。

Task 12: 请给出一个实现 mret 功能成功部分仿真的高清图片，并对涉及到的信号加以详细解释 (5 points)

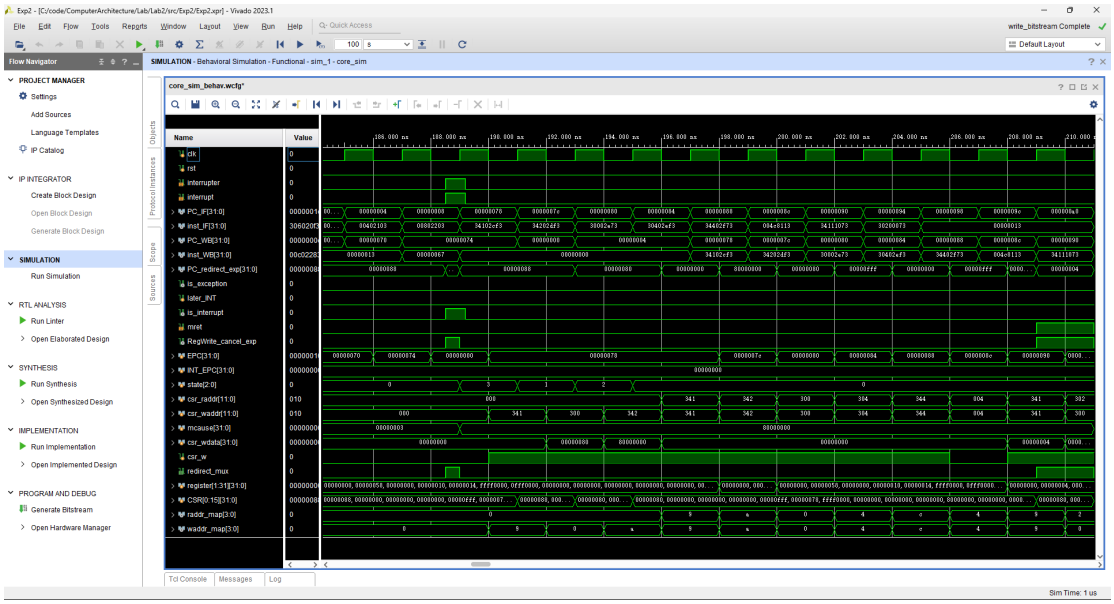
异常处理程序执行完毕，执行 mret（对应第一个 ecall 的 mret）：



在 59000ns 时，异常处理程序执行完毕，mret 指令进入 MEM 阶段（RV32core 中 mret 接入 ExceptionUnit 的信号为 mret_MEM，故 mret 是在 MEM 阶段检测的），mret 信号拉高，此时状态机处于 IDLE 状态(state=0),开始处理 mret:PC 重定向使能(redirect_mux=1)，取 PC 重定向地址为 mpec 中保存的值（同样由于我给 CSR 寄存器模块引出了 mepc 读信号，这里直接让 PC_redirect 获取到 mepc 值），同时也需要 flush 当前流水线（虽然后边几条都是 nop），同时从 mstatus[mpie]恢复 mstatus[mie]，可以看到 csr_waddr=300，csr_wdata=0x88。由于以上操作一个周期就能完成，故没有进行状态转换。

Task 13: 请给出一个触发外部中断部分仿真的高清图片，并对涉及到的信号加以详细解释 (5 points)

通过更改 core_sim.v 中的 interrupter 输入，手动在 189000ns 处给出了一个持续 700ns 的外部中断信号，可以看到 189000ns 处 interrupter 为高：



由于我设计的外部中断也为时钟上升沿触发，故在 189000ns 处，状态才开始转换，通过对于此时的仿真信号进行分析，可以看到 mcause 和 csr_waddr 均未准备好，并非所需的 mcause=0x80000000 和 mepc 的地址，这是由于外部中断是突然产生的，而不像异常实质上是推迟到 WB 阶段处理，所以异常可以在 WB 阶段一开始就准备好所需的地址以及对应的 mcause，但是外部中断不行，于是我将其增加了一个状态（INT_CACHE，state=3），用于准备 mepc 地址，可以看到在 189000ns 时，状态转换到 INT_CACHE(state=3)，更新 mcause 为本实验中规定的外部中断的 mcause(0x80000000),并在下降沿准备好了 csr_waddr=0x341(mepc 地址)以及 csr_wdata(将下一条需要执行的指令存储至 mepc)。后续中断处理与异常相同。

四、 讨论与心得

Task 14: 请写出对本次实验内容的深入讨论，或者本次实验的心得体会。例如遇到的难题等等。请认真填写本模块，若不填写或胡乱填写将酌情扣分，写明白真实情况即可。 (+10 points)

这次实验遇到了许多问题：

1. 寄存器写入的延迟问题，如果都设置为上升沿触发，将会导致传入写地址和读地址时比实际需要地址的时间慢一整个时钟周期，最后我选择将状态机处理过程设置为下降沿触发，将其缩短为半个时钟周期。
2. 异常与外部中断的不同处理方式，一开始我将这两种中断都用同一种方式处理，导致外部中断不能获取正确的写地址，最后发现是因为异常是

在进入 WB 阶段阶段就可准备好写地址的，而外部中断是无法提前准备好地址的，故将其空置一个状态解决。

3. 上板过程中遇到的时序问题：一开始上板触发不同种类异常与外部中断的 mcause 都一直不变，可能是因为我将异常处理模块中 mcause 的更新放在了 always @(*)块中，后来将其更改为上升沿触发，并通过增加一条 mcause<=mcause 消除 latch，上板后可以正确显示 mcause（因为异常处理程序中执行了 csrr x27,0x342 指令，可以在显示器中的 x27 寄存器查看）。
4. 一开始希望以异步的方式检测外部中断，但是欠缺考虑，忽略了信号抖动带来的问题，最后将其触发方式改为同步。