哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

课程报告

开课学期：2023 夏季

课程名称：计算机设计与实践

项目名称：基于 miniRV 的 SoC 设计

项目类型：综合设计型

课程学时：56 地点：T2615

学生班级：

学生学号：

学生姓名：

评阅教师：

报告成绩：

实验与创新实践教育中心制

2023 年 7 月

注：本设计报告中各个部分如果页数不够，请同学们自行扩页。原则上一定要把报告写详细，能说明设计的成果、特色和过程。报告应该详细叙述整体设计，以及设计中的每个模块。设计报告将是评定每个人成绩的重要组成部分（**设计内容及报告写作**都作为评分依据）。

设计概述（罗列出所有实现的指令，以及单周期/流水线 CPU 频率）
共实现 24 条 miniRV 指令： 移位指令：sll, sllli, srl, srli, sra, srai 算数运算指令：add, addi, sub, lui 逻辑运算指令：xor, xori, or, ori, and, andi 跳转链接指令：jal, jalr 加载指令：lw 存储指令：sw 分支指令：beq, bne, blt, bge 单周期 CPU 频率：50 MHz 流水线 CPU 频率：50 MHz
设计的主要特色（除基本要求以外的设计）
单周期 CPU 实现了 50 MHz 的频率

资源使用、功耗数据截图（Post Implementation；含单周期、流水线 2 个截图）

单周期：

DRC Violations

Summary: 7 warnings

Implemented DRC Report

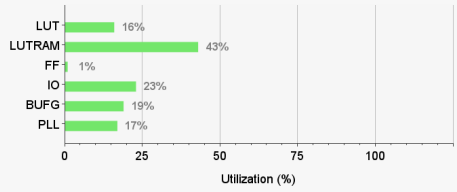
Timing Setup | Hold | Pulse Width

Worst Negative Slack (WNS):	3.2 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	10291

Implemented Timing Report

Utilization Post-Synthesis | **Post-Implementation**

Graph | Table



Power Summary | On-Chip

Total On-Chip Power:	**0.209 W**
Junction Temperature:	**25.6 °C**
Thermal Margin:	59.4 °C (22.1 W)
Effective ϑJA:	2.7 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

流水线：

DRC Violations

Summary: 4 warnings

Implemented DRC Report

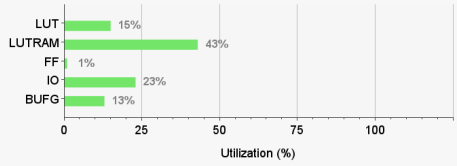
Timing Setup | Hold | Pulse Width

Worst Negative Slack (WNS):	0.506 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	82891

Implemented Timing Report

Utilization Post-Synthesis | **Post-Implementation**

Graph | Table



Power Summary | On-Chip

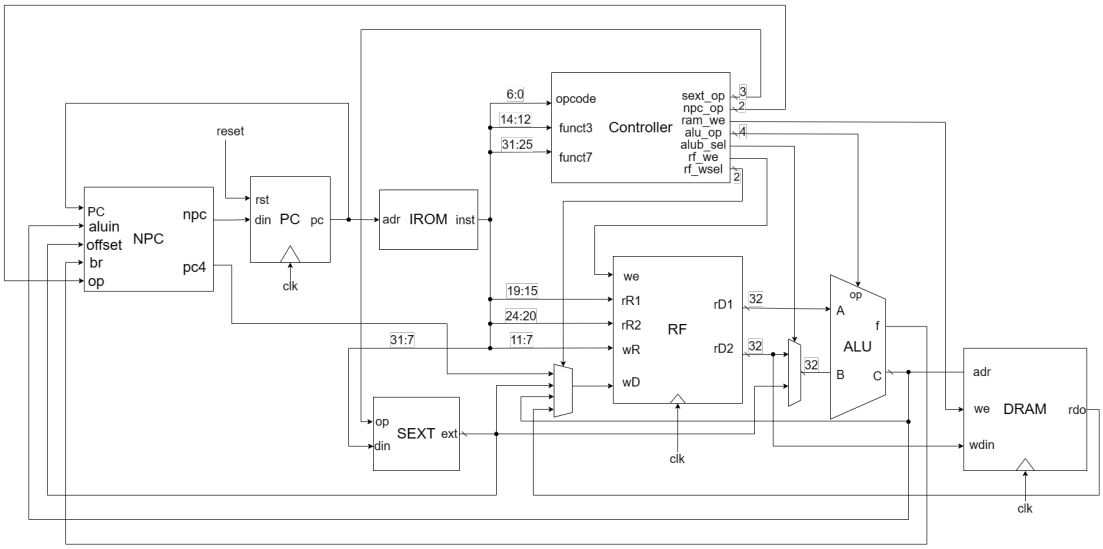
Total On-Chip Power:	**0.118 W**
Junction Temperature:	**25.3 °C**
Thermal Margin:	59.7 °C (22.2 W)
Effective ϑJA:	2.7 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

1 单周期 CPU 设计与实现

1.1 单周期 CPU 数据通路设计

要求：贴出完整的单周期数据通路图，无需画出模块内的具体逻辑，但要标出模块的接口信号名、模块之间信号线的信号名和位宽，并用文字阐述各模块的功能。



NPC 模块：

- 根据当前 PC 值、ALU 运算值、偏移量、跳转标识、控制信号决定下一条指令的地址；
- 产生 pc4 信号（pc4 = pc + 4），用于写回寄存器。

PC 模块：

- 产生新一条指令地址。

IROM 模块：

- 根据 PC 模块的地址获取指令。

Controller 模块：

- 根据指令产生控制信号，包括立即数扩展控制信号 sext_op、下一条指令控制信号 npc_op，数据存储器写入控制信号 ram_we，ALU 控制信号 alu_op，ALU 输入选择信号 alub_sel，寄存器堆写入控制信号 rf_we，寄存器堆写入选择信号 rf_wsel。

SEXT 模块：

- 按照 Controller 模块产生的立即数扩展控制信号 sext_op 将操作数扩展至 32 位。

RF 模块：

- 存取寄存器的值。

ALU 模块：

- 根据 Controller 模块产生的 ALU 控制信号 alu_op，进行对应的运算。

DRAM 模块：

- 根据数据存储器写入控制信号 ram_we，读/写数据存储器。

1.2 单周期 CPU 模块详细设计

要求：以表格的形式列出各个部件的接口信号、位宽、功能描述等，并结合图、表、核心代码等形象化工具和手段，详细描述各个部件的关键实现。

NPC 模块：

部件	in/out	信号名	位宽	功能描述
NPC	in	PC	32	获取当前 PC 值
NPC	in	aluin	32	获取 ALU 计算值
NPC	in	offset	32	获取跳转偏移量
NPC	in	br	1	判断是否跳转
NPC	in	op	2	选择 NPC 所执行的操作
NPC	out	npc	32	下一条指令的地址
NPC	out	pc4	32	PC+4，用于写回

```
/* NPC */
assign NPC_pc4 = NPC_pc + 4;
always @(*) begin
    case (NPC_op)
        // NPC_PC4
        2'b00: NPC_npc = NPC_pc4;
        // NPC_B
        2'b01: NPC_npc = NPC_br ? NPC_pc + NPC_offset : NPC_pc4;
        // NPC_JMP
        2'b10: NPC_npc = NPC_pc + NPC_offset;
        // NPC_JALR
        default: NPC_npc = NPC_aluin;
    endcase
end
```

NPC 关键：获取下一条指令的地址

PC 模块：

部件	in/out	信号名	位宽	功能描述
PC	in	rst	1	复位
PC	in	clk	1	时钟
PC	in	din	32	获取 npc 的值
PC	out	pc	32	当前 PC 地址

```
/* PC */
always @(posedge clk or posedge rst) begin
    if (rst)
        PC_pc <= 32'hFFFF_FFFC;
    else
        PC_pc <= PC_din;
end
```

PC 关键：产生指令的地址

IROM 模块：

部件	in/out	信号名	位宽	功能描述
IROM	in	adr	32	指令的地址
IROM	out	inst	32	实际指令

IROM 关键：获取实际指令

Controller 模块：

部件	in/out	信号名	位宽	功能描述
Controller	in	opcode	7	获取指令的 opcode
Controller	in	funct3	3	获取指令的 funct3
Controller	in	funct7	7	获取指令的 funct7
Controller	out	sext_op	3	产生 SEXT 部件的控制信号
Controller	out	npc_op	2	产生 NPC 部件的控制信号
Controller	out	ram_we	1	控制 DRAM 部件的写入使能
Controller	out	alu_op	4	产生 ALU 部件的控制信号
Controller	out	alub_sel	1	产生 ALU 部件的输入选择信号
Controller	out	rf_we	1	控制 RF 部件的写入使能
Controller	out	rf_wsel	2	产生 RF 部件的输入选择信号

```
always @(*) begin
    case (CU_opcode)
        // R-type
        7'b0110011:略
        // I-type
        7'b0010011: 略
        // lw
        7'b0000011: 略
        // jalr
        7'b1100111: 略
        // S-type
        7'b0100011: 略
        // B-type
        7'b1100011: 略
        // U-type
        7'b0110111: 略
        // J-type
        default: 略
    endcase
end
```

Controller 关键：产生各个控制信号

SEXT 部件：

部件	in/out	信号名	位宽	功能描述
SEXT	in	op	3	获取控制信号
SEXT	in	din	25	获取未扩展的立即数
SEXT	out	ext	32	输出已扩展的立即数

```
/* SEXT */
always @(*) begin
    case (SEXT_op)
        // EXT_I
        3'b000:  SEXT_ext = $signed({SEXT_din[24:13]});
        // EXT_S
        3'b001:  SEXT_ext = $signed({SEXT_din[24:18],SEXT_din[4:0]});
        // EXT_B
        3'b010: SEXT_ext = $signed({SEXT_din[24],SEXT_din[0],
                            SEXT_din[23:18],SEXT_din[4:1], 1'b0});
        // EXT_U
        3'b011:  SEXT_ext = {SEXT_din[24:5], 12'b0};
        // EXT_J
        default: SEXT_ext = $signed({SEXT_din[24], SEXT_din[12:5],
                             SEXT_din[13], SEXT_din[23:14], 1'b0});
    endcase
end
```

SEXT 关键：产生立即数

RF 模块：

部件	in/out	信号名	位宽	功能描述
RF	in	clk	1	时钟信号
RF	in	we	1	获取写入使能信号
RF	in	rR1	5	获取输入信号 1
RF	in	rR2	5	获取输入信号 2
RF	in	wR	5	获取待写入寄存器号
RF	in	wD	32	获取待写入数
RF	out	rD1	32	输出读取的数据 1
RF	out	rD2	32	输出读取的数据 2

```
/* RF */
// 同步写
always @(posedge clk) begin
    if (RF_we) begin
        for (i = 1; i < 32; i = i + 1) begin
            if (RF_wR == i)
                RF_reg[i] <= RF_wD;
        end
    end
end
// 异步读
always @(*) begin
    for (ii = 1; ii < 32; ii = ii + 1) begin
            if (RF_rR1 == ii)
                RF_rD1 = RF_reg[ii];
        end
    for (jj = 1; jj < 32; jj = jj + 1) begin
            if (RF_rR2 == jj)
                RF_rD2 = RF_reg[jj];
        end
    if (RF_rR1 == 32'b0) RF_rD1 = 32'b0;
    if (RF_rR2 == 32'b0) RF_rD2 = 32'b0;
end
```

RF 关键：正确读写寄存器堆

ALU 模块：

部件	in/out	信号名	位宽	功能描述
ALU	in	op	4	获取控制信号
ALU	in	A	32	获取待操作数 1
ALU	in	B	32	获取待操作数 2
ALU	out	C	32	输出计算结果
ALU	out	f	1	判断是否跳转

```
/* ALU */
always @(*) begin
    case (ALU_op)
        // ALU_ADD
        4'b0001: 略
        // ALU_SUB
        4'b0010: 略
        // ALU_AND
        4'b0011: 略
        // ALU_OR
        4'b0100: 略
        // ALU_XOR
        4'b0101: 略
        // ALU_SLL
        4'b1000: 略
        // ALU_SRL
        4'b1001: 略

        ……

    endcase
end
```

ALU 关键：进行逻辑运算

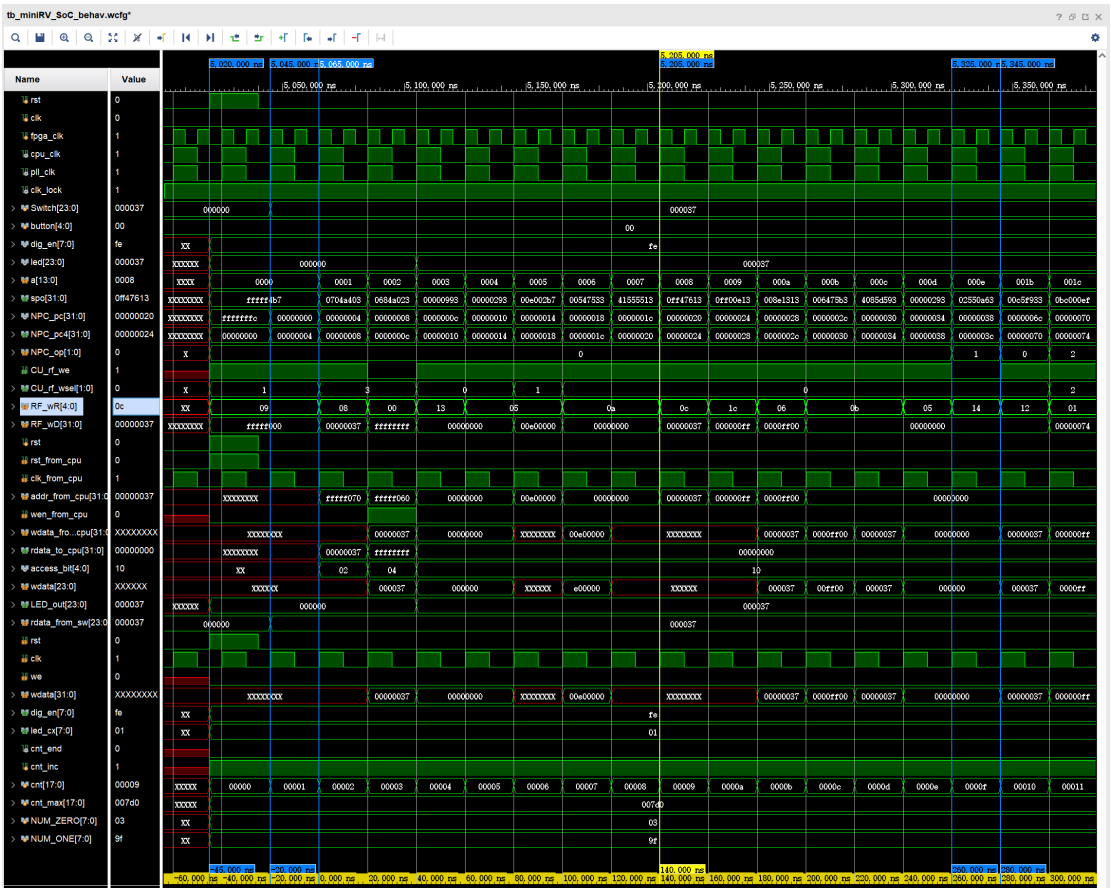
DRAM 模块：

部件	in/out	信号名	位宽	功能描述
DRAM	in	clk	1	时钟信号
DRAM	in	adr	32	获取待操作地址
DRAM	in	we	1	获取写入使能信号
DRAM	in	wdin	32	获取写入数据
DRAM	out	rdo	32	输出访存结果

DRAM 关键：正确读写数据存储器

1.3 单周期 CPU 仿真及结果分析

要求：包含逻辑运算、访存、分支跳转三类指令的仿真截图及波形分析；每类指令的截图和分析中，至少包含 1 条具体指令；截图需包含信号名和关键信号。



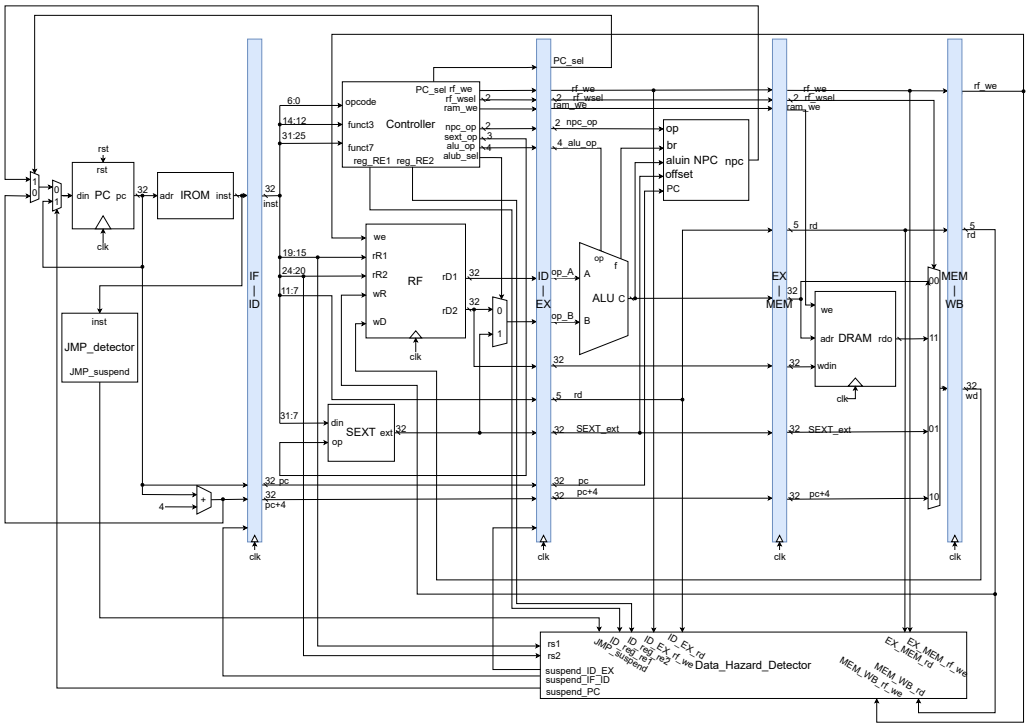
1. 5020ns 时，置 rst 信号为 1，各信号成功初始化，符合预期；
2. 5045ns 时，进入第一个周期，设置输入为 000037。pc 值（NPC_pc）为 0x0000_0000，符合预期；
3. 【访存指令 lw】5065ns 时，指令地址为 0x0000_0004，指令为 0x0704a403，即`lw x8,70(x9)`，由图可知，RF 写回的数据 RF_wD 为 0x0000_0037，符合预期；
4. 【逻辑运算指令 andi】5205ns 时，指令地址为 0x0000_0020，指令为 0x0ff4_7613，即`andi x12,x8,0xff`，由图可知，RF 写回的数据 RF_wD 为 0x0000_0037，符合预期；
5. 【分支跳转指令 beq】5325ns 时，指令地址为 0x0000_0038，指令为 0x255_0a63，即`beq x10,x5,0x34`，而 x10 与 x5 均为 0，应该跳转。由图可知，在 5345ns 时，指令地址为 0x0000_006c，成功跳转，符合预期。

综上所述，由仿真结果可知，本程序正确实现了单周期 CPU 的功能。

2 流水线 CPU 设计与实现

2.1 流水线 CPU 数据通路

要求：贴出完整的流水线数据通路图，无需画出模块内的具体逻辑，但要标出模块的接口信号名、模块之间信号线的信号名和位宽，并用文字阐述各模块的功能。此外，数据通路图应当能体现出流水线是如何划分的，并用文字阐述每个流水级具备什么功能、需要完成哪些操作。



注：数据通路图画了暂停的部分，实际代码中暂停功能尚未实现。

CPU 分为 5 个流水级，分别为 IF、ID、EX、MEM、WB。

IF 流水级：取址，判断是否有跳转指令（是否需要暂停）。

PC 模块：

- 产生新一条指令地址。

IROM 模块：

- 根据 PC 模块的地址获取指令。

JMP_detector 模块：

- 判断是否有跳转指令。

ID 流水级：译码，产生控制信号，生成立即数。

Controller 模块：

- 根据指令产生控制信号，包括立即数扩展控制信号 sext_op、下一条指

令控制信号 npc_op，数据存储器写入控制信号 ram_we，ALU 控制信号 alu_op，ALU 输入选择信号 alub_sel，寄存器堆写入控制信号 rf_we，寄存器堆写入选择信号 rf_wsel，PC 选择信号 PC_sel，寄存器读信号 reg_RE1 和 reg_RE2。

SEXT 模块：

- 按照 Controller 模块产生的立即数扩展控制信号 sext_op 将操作数扩展至 32 位。

RF 模块：

- 存储寄存器的值。

EX 流水级：逻辑运算，计算下一条指令的地址。

ALU 模块：

- 根据 Controller 模块产生的 ALU 控制信号 alu_op，进行对应的运算。

NPC 模块：

- 根据 PC 值、ALU 运算值、偏移量、跳转标识、控制信号决定下一条指令的地址；

MEM 流水级：读写数据存储器。

DRAM 模块：

- 根据数据存储器写入控制信号 ram_we，读/写数据存储器。

WB 流水级：写回寄存器。

数据冒险检测模块 Data_Hazard_Detector 用于检测数据冒险：

- 检测将要写入寄存器堆的数据是否存在冒险，若存在，输出暂停的周期数。

流水线寄存器 IF_ID、ID_EX、EX_MEM、MEM_WB：

- 用于暂存各流水级之间的信息；输入流水线暂停信号，控制流水线的通断。

2.2 流水线 CPU 模块详细设计

要求：以表格的形式列出所有与单周期不同的部件的接口信号、位宽、功能描述等，并结合图、表、核心代码等形象化工具和手段，详细描述这些部件的关键实现。此外，如果实现了冒险控制，必须结合数据通路图，详细说明数据冒险、控制冒险的解决方法。

Controller 模块：

部件	in/out	信号名	位宽	功能描述
Controller	out	reg_RE1	1	RF 数据 1 的读信号
Controller	out	reg_RE2	1	RF 数据 2 的读信号
Controller	out	PC_sel	1	PC 选择信号

// 表格中为新增的

以下模块已设计，但未跑通 Trace。

JMP_detector 模块：

部件	in/out	信号名	位宽	功能描述
JMP_detector	in	inst	32	获取信号
JMP_detector	out	JMP_suspend	1	是否需要暂停

Data_Hazard_Detector 模块：

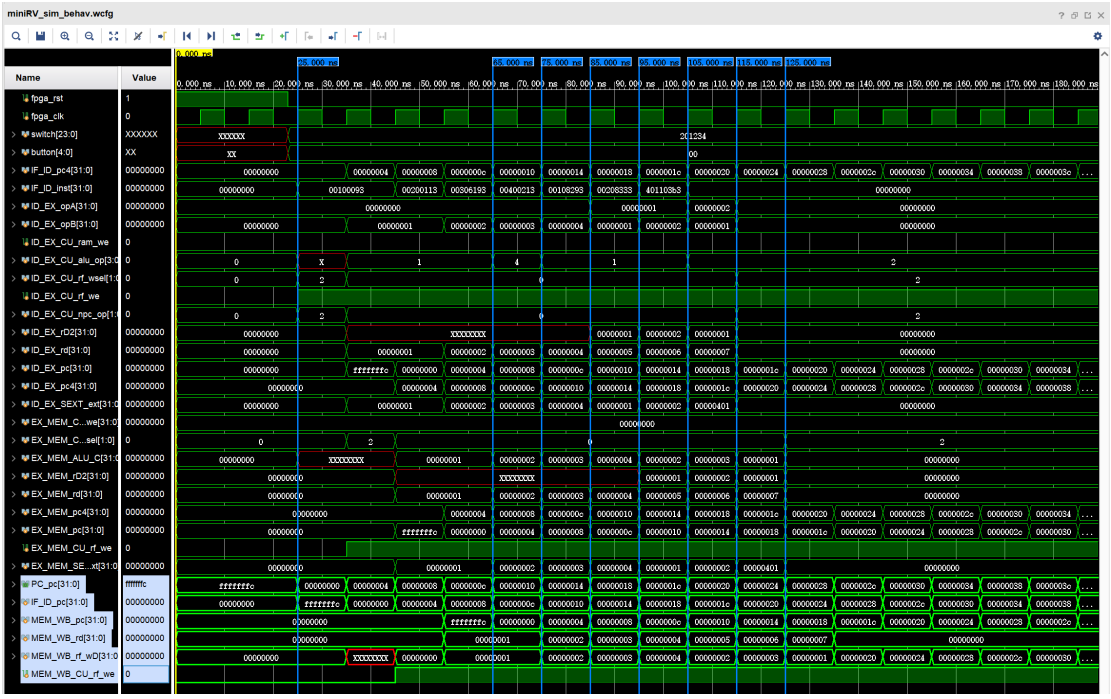
部件	in/out	信号名	位宽	功能描述
DHD	in	rs1	5	写入数据 1 的寄存器号
DHD	in	rs2	5	写入数据 2 的寄存器号
DHD	in	JMP_suspend	1	是否因为跳转指令而停止
DHD	in	ID_reg_re1	1	RF 数据 1 的读信号（ID 阶段）
DHD	in	ID_reg_re2	1	RF 数据 2 的读信号（ID 阶段）
DHD	in	ID_EX_rf_we	1	RF 的写使能（EX 阶段）
DHD	in	ID_EX_rd	5	RF 的写回寄存器号（EX 阶段）
DHD	in	EX_MEM_rd	5	RF 的写回寄存器号（MEM 阶段）
DHD	in	EX_MEM_rf_we	1	RF 的写使能（MEM 阶段）
DHD	in	MEM_WB_rd	5	RF 的写回寄存器号（WB 阶段）
DHD	in	MEM_WB_rf_we	1	RF 的写使能（WB 阶段）

解决数据冒险：检测写回的寄存器是否和要读取的寄存器存在冲突，若冲突，则暂停对应情况的周期数。

2.3 流水线 CPU 仿真及结果分析

要求：包含控制冒险和数据冒险三种情形的仿真截图，以及波形分析。若仅实现了理想流水，则此处贴上理想流水的仿真截图及详细的波形分析。

代码部分数据冒险暂未实现，以下是理想流水线的仿真分析：



1. 25ns 时，开始复位后的第一个时钟周期，PC 的值（PC_pc）为 0x0000_0000，符合预期
2. 65ns 时，即开始复位后的第五个时钟周期，此时处于第一条指令的写回阶段，第一条指令为`addi x1, x0, 1`。图中，MEM_WB_pc 为 0x0000_0000，符合预期；寄存器写使能（MEM_WB_CU_rf_we）为 1，符合预期；写回的寄存器（MEM_WB_rd）为 1，写回的数据（MEM_WB_rf_wD）为 1，符合预期；
3. 75ns 时，即开始复位后的第六个时钟周期，此时处于第二条指令的写回阶段，第二条指令为`addi x2, x0, 2`图中，MEM_WB_pc 为 0x0000_0004，符合预期；寄存器写使能（MEM_WB_CU_rf_we）为 2，符合预期；写回的寄存器（MEM_WB_rd）为 2，写回的数据（MEM_WB_rf_wD）为 1，符合预期；
4. 125ns 时，此时处于第七条指令的写回阶段，第七条指令为`sub x7, x2, x1`。图中，MEM_WB_pc 为 0x0000_0018，符合预期；寄存器写使能（MEM_WB_CU_rf_we）为 1，符合预期；写回的寄存器（MEM_WB_rd）为 7，写回的数据（MEM_WB_rf_wD）为 1，符合预期；

综上所述，由仿真结果可知，本程序正确实现了理想流水线 CPU 的功能。

附：理想汇编指令：

```
addi x1, x0, 1
addi x2, x0, 2
ori  x3, x0, 3
addi x4, x0, 4
addi x5, x1, 1
add  x6, x1, x2
sub  x7, x2, x1
```

3 设计过程中遇到的问题及解决方法

要求：包括设计过程中遇到的有价值的错误，或测试过程中遇到的有价值的问题。所谓有价值，指的是解决该错误或问题后，能够学到新的知识和技巧，或加深对已有知识的理解和运用。

汇编程序：

问题：不知道存储指令与外设如何交互。
解决：运行示例代码，知道不同外设使用不同的基地址。

单周期：

问题：BLT 和 BGE 指令两数大小比较时负数比正数大。
解决：使用有符号数比较： `if ($signed(ALU_A) < $signed(ALU_B))`

问题：不知道数据存储器如何使用。
解决：阅读总线桥代码（Bridge.v），了解具体内容。

问题：IROM 导入机器码存在问题。
解决：删除中文注释。

问题：Trace 第一条指令报错。
解决：PC 从 FFFF_FFFC 开始。

问题：Trace 时经常在寄存器堆处报错。
解决：不给 x0 赋值，取 x0 值时直接取出 0。

问题：上板时无复位键。
解决：将一个 button 设置为复位键。

问题：七段数码管显示出错。
解决：多次修改刷新频率，使其合理。

流水线：

问题：将单周期代码改为流水线时无从下手。
解决：先修改数据通路图

问题：NPC 不能放在 IF 阶段。
解决：将 NPC 放在 EX 阶段。

4 总结

要求：谈谈学完本课程后的个人收获以及对本课程的建议和意见。请在认真总结和思考后填写总结。

个人收获：

通过夏季学期的课程，我完成了 RISC-V 汇编语言的程序设计，并用 verilog 完成了单周期和理想流水线的 CPU 设计。本次课程让我收获颇丰，我熟悉了 RARS、Logisim 等汇编和模拟仿真工具的使用；熟练掌握了 RISC-V 汇编语言，熟悉并理解了 RISC-V 指令系统；了解程序在单周期 RISC-V CPU 搭建的 SoC 中的运行过程；理解了单周期 CPU 工作过程；理解指令存储器和数据存储器的哈佛结构存储；熟悉 miniRV 指令集；掌握单周期 CPU 设计与实现方法；理解流水线 CPU 工作过程；理解流水线冲突产生的原因及其解决方法；掌握流水线 CPU 设计与实现方法。

建议：

可以多整理一些易错点，列成共享文档，防止在实验过程中被一些小问题卡住。

总结：

在实验一中，通过实现 RISCV 指令的汇编程序，我理解了指令集和汇编语言的运作方式。这帮助我更好地理解计算机程序的执行过程。实验二和实验三中，通过使用 Verilog 完成单周期和流水线的 CPU 设计，我提高了自己的 CPU 设计能力。在完成 CPU 设计的过程中，我也学会了问题解决和调试的技巧。硬件设计中常常会遇到各种问题，如逻辑错误、时序问题等，通过仿真等方法不断调试和排查，我学会了如何更快地找到问题并解决它们。