

深圳大学实验报告

课程名称: 计算机系统(3)

实验项目名称: 新增指令实验

学院: 计算机与软件学院

专业: 计算机与软件学院所有专业

指导教师: 刘刚

报告人: ___ 学号: ___ 班级: ___

实验时间: 2025年12月17日

实验报告提交时间: 2025年12月26日

一、实验目标：

了解 RISC-V mini 处理器架构，在其基础之上新增一个指令，完成设计并观察指令执行。

二、实验内容

- 1) 修改数据通路，新增指令 comb rs1,rs2,rd 采用 R 型指令格式，实现将 rs1 高 16 位和 rs2 低 16 位拼接成 32 位整数，并且保存到 rd 寄存器。
- 2) 在处理器上执行该指令，观察仿真波形，验证功能是否正确。
- 3) 自行设计其他功能指令，并验证设计是否正确

三、实验环境

硬件：桌面 PC

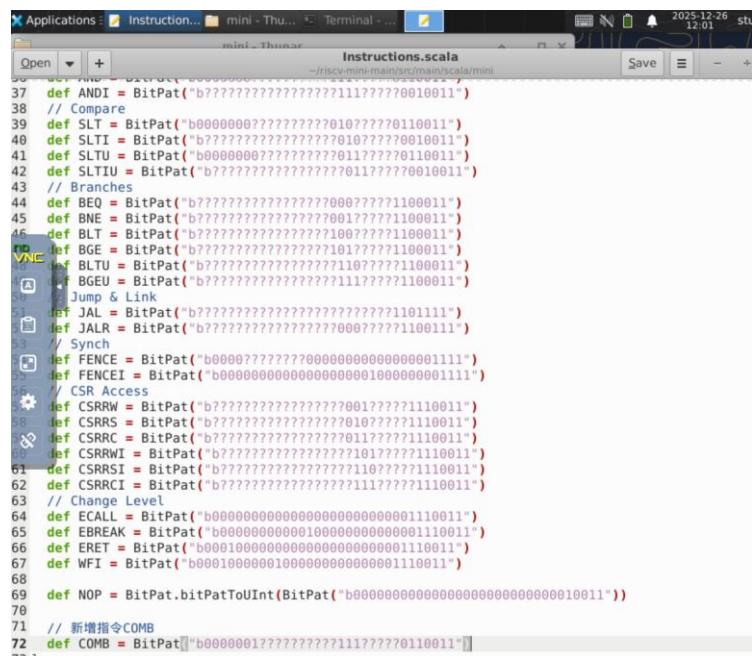
软件：Chisel 开发环境

四、实验步骤及说明

学习 Chisel 数据通路的 Chisel 描述，特别是指令译码部分和 core 核心代码。然后按照下面操作完成指令译码器的修改，以及数据通路的修改，

具体操作如下：**按照参考文档完成 comb 指令的实现，自行设计新指令实现其功能并验证。**

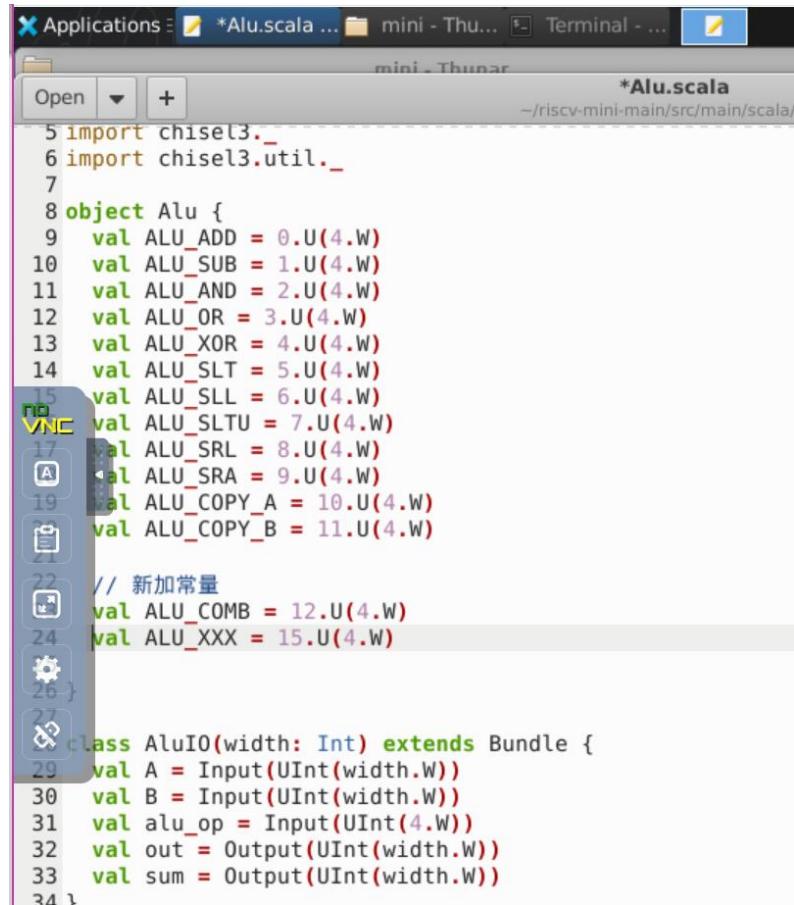
1. 在 Instrucions.scala 文件中添加 comb 指令比特模式串



```
def ANDI = BitPat("b?????????????????1111????00100111")
// Compare
def SLT = BitPat("b00000007????????010?????01100111")
def SLTI = BitPat("b????????????????0101?????01100111")
def SLTU = BitPat("b00000007????????0111?????01100111")
def SLTIU = BitPat("b????????????????0111?????00100111")
// Branches
def BEQ = BitPat("b?????????????????0000?????11000111")
def BNE = BitPat("b?????????????????001?????11000111")
def BLT = BitPat("b?????????????????100?????11000111")
VNC def BGE = BitPat("b?????????????????101?????11000111")
def BLTU = BitPat("b?????????????????1101?????11000111")
def BGEU = BitPat("b?????????????????1111?????11000111")
// Jump & Link
def JAL = BitPat("b?????????????????0001?????11011111")
def JALR = BitPat("b?????????????????0000?????11001111")
// Sync
def FENCE = BitPat("b0000?????????00000000000000001111")
def FENCEI = BitPat("b0000000000000000000000001111")
// CSR Access
def CSRRW = BitPat("b?????????????????0011?????11100111")
def CSRRS = BitPat("b?????????????????0101?????11100111")
& def CSRRC = BitPat("b?????????????????0111?????11100111")
def CSRRWI = BitPat("b?????????????????1011?????11100111")
61 def CSRRSI = BitPat("b?????????????????1101?????11100111")
def CSRRCI = BitPat("b?????????????????1111?????11100111")
// Change Level
def ECALL = BitPat("b000000000000000000000000000011100111")
def EBREAK = BitPat("b000000000001000000000000000011100111")
def ERET = BitPat("b001000000000000000000000000011100111")
def WFI = BitPat("b000100000100000000000000000011100111")
def NOP = BitPat.toInt(BitPat("b0000000000000000000000000000000010011"))
// 新增指令COMB
72 def COMB = BitPat["b0000001?????????1111?????01100111"]
```

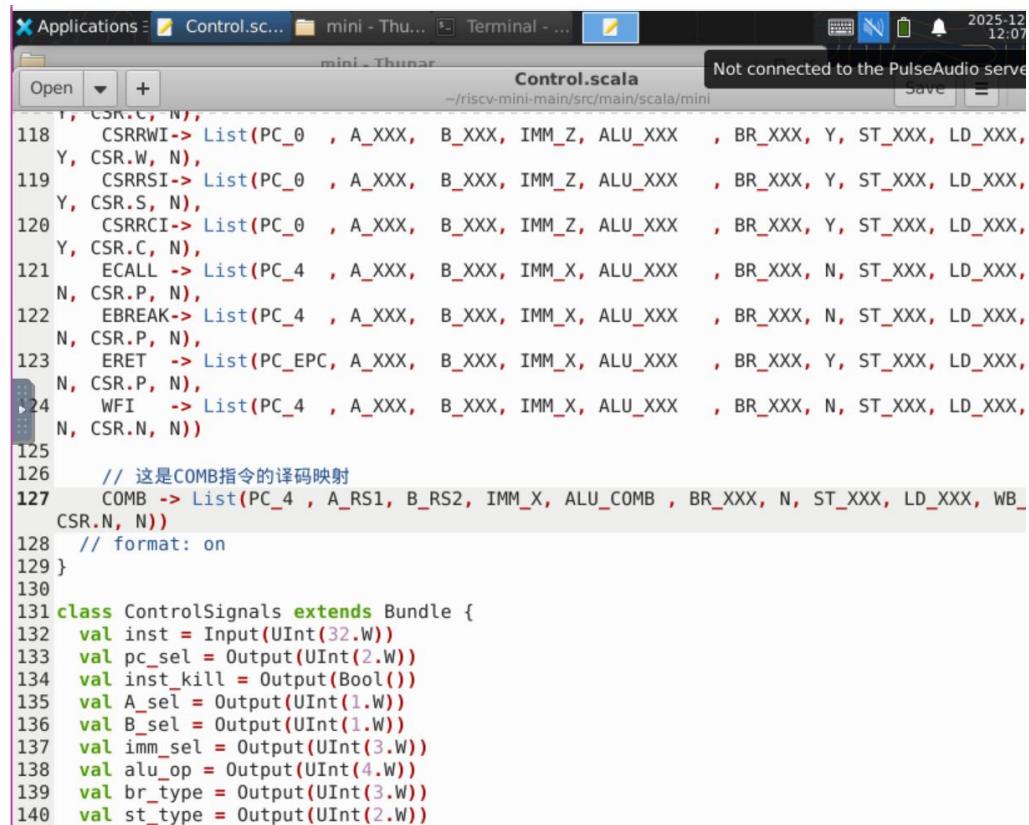
2. 添加 comb 指令的译码

在 Alu.scala 文件中添加常量 ALU_COMB，让译码器可以译码出正确的信号



```
5 import chisel3._  
6 import chisel3.util._  
7  
8 object Alu {  
9   val ALU_ADD = 0.U(4.W)  
10  val ALU_SUB = 1.U(4.W)  
11  val ALU_AND = 2.U(4.W)  
12  val ALU_OR = 3.U(4.W)  
13  val ALU_XOR = 4.U(4.W)  
14  val ALU_SLT = 5.U(4.W)  
15  val ALU_SLL = 6.U(4.W)  
16  val ALU_SLTU = 7.U(4.W)  
17  val ALU_SRL = 8.U(4.W)  
18  val ALU_SRA = 9.U(4.W)  
19  val ALU_COPY_A = 10.U(4.W)  
20  val ALU_COPY_B = 11.U(4.W)  
  
21 // 新加常量  
22  val ALU_COMB = 12.U(4.W)  
23  val ALU_XXX = 15.U(4.W)  
24 }  
25  
26 class AluIO(width: Int) extends Bundle {  
27   val A = Input(UInt(width.W))  
28   val B = Input(UInt(width.W))  
29   val alu_op = Input(UInt(4.W))  
30   val out = Output(UInt(width.W))  
31   val sum = Output(UInt(width.W))  
32 }
```

在 Control.scala 文件中添加代码



```
1, CSR.C, N),  
118  CSRWR-> List(PC_0 , A_XXX, B_XXX, IMM_Z, ALU_XXX , BR_XXX, Y, ST_XXX, LD_XXX,  
Y, CSR.W, N),  
119  CSRSI-> List(PC_0 , A_XXX, B_XXX, IMM_Z, ALU_XXX , BR_XXX, Y, ST_XXX, LD_XXX,  
Y, CSR.S, N),  
120  CSRRCI-> List(PC_0 , A_XXX, B_XXX, IMM_Z, ALU_XXX , BR_XXX, Y, ST_XXX, LD_XXX,  
Y, CSR.C, N),  
121  ECALL -> List(PC_4 , A_XXX, B_XXX, IMM_X, ALU_XXX , BR_XXX, N, ST_XXX, LD_XXX,  
N, CSR.P, N),  
122  EBREAK-> List(PC_4 , A_XXX, B_XXX, IMM_X, ALU_XXX , BR_XXX, N, ST_XXX, LD_XXX,  
N, CSR.P, N),  
123  ERET -> List(PC_EPC, A_XXX, B_XXX, IMM_X, ALU_XXX , BR_XXX, Y, ST_XXX, LD_XXX,  
N, CSR.P, N),  
124  WFI -> List(PC_4 , A_XXX, B_XXX, IMM_X, ALU_XXX , BR_XXX, N, ST_XXX, LD_XXX,  
N, CSR.N, N))  
125  
126 // 这是COMB指令的译码映射  
127  COMB -> List(PC_4 , A_RS1, B_RS2, IMM_X, ALU_COMB , BR_XXX, N, ST_XXX, LD_XXX, WB_  
CSR.N, N))  
128 // format: on  
129 }  
130  
131 class ControlSignals extends Bundle {  
132   val inst = Input(UInt(32.W))  
133   val pc_sel = Output(UInt(2.W))  
134   val inst_kill = Output(Bool())  
135   val A_sel = Output(UInt(1.W))  
136   val B_sel = Output(UInt(1.W))  
137   val imm_sel = Output(UInt(3.W))  
138   val alu_op = Output(UInt(4.W))  
139   val br_type = Output(UInt(3.W))  
140   val st_type = Output(UInt(2.W))
```

3. 实现 comb 指令的执行操作

在 Alu.scala 文件添加将 rs1 高 16 位和 rs2 低 16 位拼接成 32 位整数的操作

```
Click to start dragging "Alu.scala (~/.riscv-mini-main/src/main/scal...)" -> riscv-mini-main/src/main/scala/min...
39  def width: Int
40  val io: AluIO
41 }
42
43 class AluSimple(val width: Int) extends Alu {
44   val io = IO(new AluIO(width))
45
46   val shamt = io.B(4, 0).asUInt
47
48   io.out := MuxLookup(io.alu_op, io.B)()
Seq(
  ALU_ADD -> (io.A + io.B),
  ALU_SUB -> (io.A - io.B),
  ALU_SRA -> (io.A.asInt >> shamt).asUInt,
  ALU_SR -> (io.A >> shamt),
  ALU_SLL -> (io.A << shamt),
  ALU_SLT -> (io.A.asInt < io.B.asInt),
  ALU_SLTU -> (io.A < io.B),
  ALU_AND -> (io.A & io.B),
  ALU_OR -> (io.A | io.B),
  ALU_XOR -> (io.A ^ io.B),
  ALU_COPY_A -> io.A
  // COMB 指令执行
  ALU_COMB -> Cat[io.A(31,16), io.B(15,0)]
```

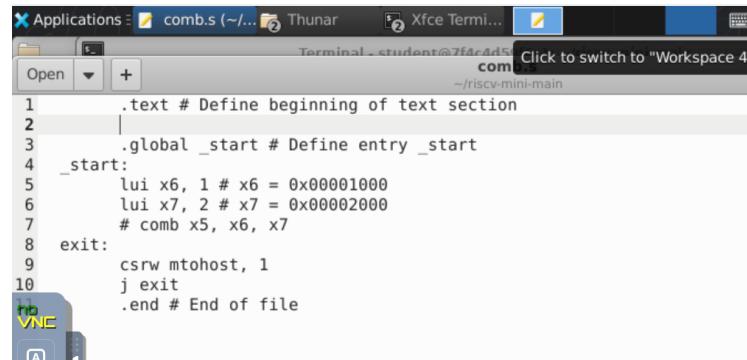
```
Applications : *Alu.scala ... mini - Thu... Terminal - ...
Current workspace: "Workspace 1"
Open + ~/riscv-mini/main/src/main/scala/mini

61    // COMB指令执行
62    ALU_COMB -> Cat(io.A(31,16), io.B(15,0))
63
64  )
65
66
67  io.sum := io.A + Mux(io.alu_op(0), -io.B, io.B)
68 }
69
70 class AluArea(val width: Int) extends Alu {
71   val io = IO(new AluIO(width))
72   val sum = io.A + Mux(io.alu_op(0), -io.B, io.B)
73   val cmp =
74     Mux(io.A(width - 1) === io.B(width - 1), sum(width - 1), Mux(io.alu_op(1), io.A(width - 1)))
75   val shamt = io.B(4, 0).asUInt
76   val shin = Mux(io.alu_op(3), io.A, Reverse(io.A))
77   val shiftr = (Cat(io.alu_op(0) && shin(width - 1), shin).asSInt >> shamt).asUInt
78   val shiftl = Reverse(shiftr)
79
80   // 将A(rs1)的高16位与B(rs2)的低16位拼接
81   val comb = Cat(io.A(31,16), io.B(15,0))
82
83
84   val out =
85   Mux(
86     io.alu_op === ALU_ADD || io.alu_op === ALU_SUB,
87     sum,
88     Mux(
89       io.alu_op === ALU_SLT || io.alu_op === ALU_SLTU,
90       cmp,
```

```
Applications : * "alu.scala" mini - Thu, ... Terminal ... mini_Thunar Click to switch to "Workspace 3"
Open + mini_Thunar ~ /trace/min/main/src/main/scala/main
Save ...
2025-12-26 12:12
77 val shiftfr = (Cat(io.alu_op & 0x0f, 0) == 0x0f) ? 0 : shin(width - 1), shin).asInt >> shamt)(width - 1, 0)
78 val shiftl = Reverse(shiftfr)
79
80 // 将A(rs1)的高16位与B(rs2)的低16位拼接
81 val comb = Cat(io.A(31,16), io.B(15,0))
82
83
84 val out =
85 Mux(
86   io.alu_op === ALU_ADD || io.alu_op === ALU_SUB,
87   sum,
88   Mux(
89     io.alu_op === ALU_SLT || io.alu_op === ALU_SLTU,
90     cmp,
91     Mux(
92       io.alu_op === ALU_SRA || io.alu_op === ALU_SRL,
93       shiftfr,
94       Mux(
95         io.alu_op === ALU_SLL,
96         shiftl,
97         Mux(
98           io.alu_op === ALU_AND,
99           io.A & io.B,
100          Mux(
101            io.alu_op === ALU_OR,
102            io.A | io.B,
103            Mux(io.alu_op === ALU_XOR, io.A ^ io.B,
104                 // COMB指令执行
105                 Mux(io.alu_op === ALU_COMB, comb, Mux(io.alu_op === ALU_COPY_A, io.A,
106                   io.B)))
107               )
108             )
109           )
110         )
111       )
112     )
113   )
114 )
```

4. 对 comb 指令进行测试

创建 comb.s 文件

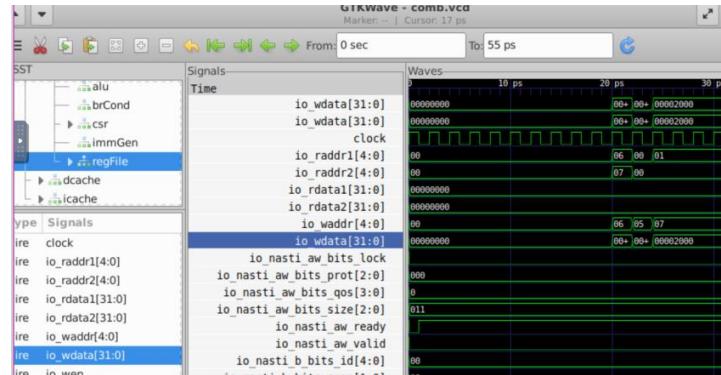


```
1 .text # Define beginning of text section
2 |
3 .global _start # Define entry _start
4 _start:
5    lui x6, 1 # x6 = 0x00001000
6    lui x7, 2 # x7 = 0x00002000
7    # comb x5, x6, x7
8    exit:
9       csrw mtohost, 1
10      j exit
11 .end # End of file
```

编译并转换 hex 文件，按要求进行修改后进行 make，产生 vcd 文件

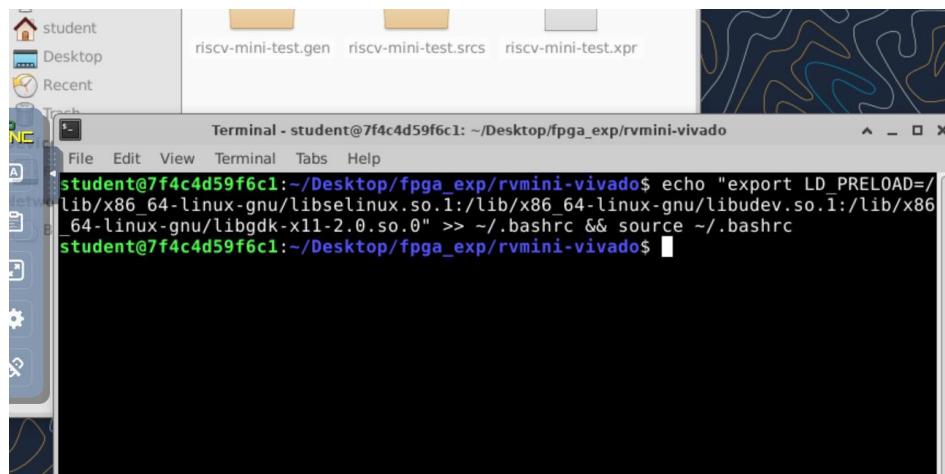
```
a -pthread -lpthread -latomic -o /home/student/riscv-mini-main/V
rm VTile_ALL.verilator_deplist.tmp
make[1]: Leaving directory '/home/student/riscv-mini-main/generated-s
rc'
student@7f4c4d59f6c1:~/riscv-mini-main$ ./VTile comb.hex comb.vcd
Enabling waves...
Starting simulation!
Simulation completed at time 56 (cycle 5)
Finishing simulation!
student@7f4c4d59f6c1:~/riscv-mini-main$
```

查看波形文件

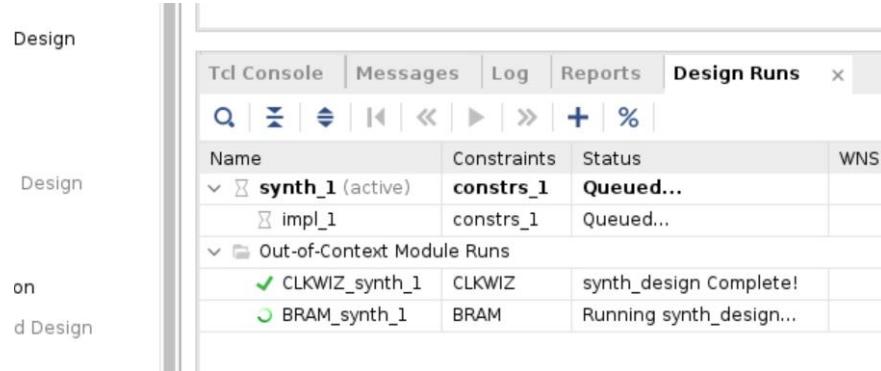


从波形图中可以看出， comb 指令将拼接后的结果 0x00002000 写回到了 5 号寄存器中，故该指令执行正常。

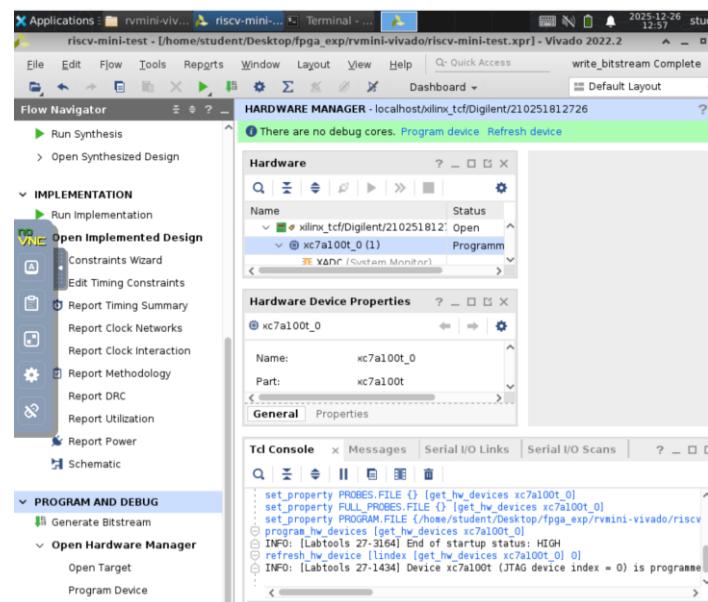
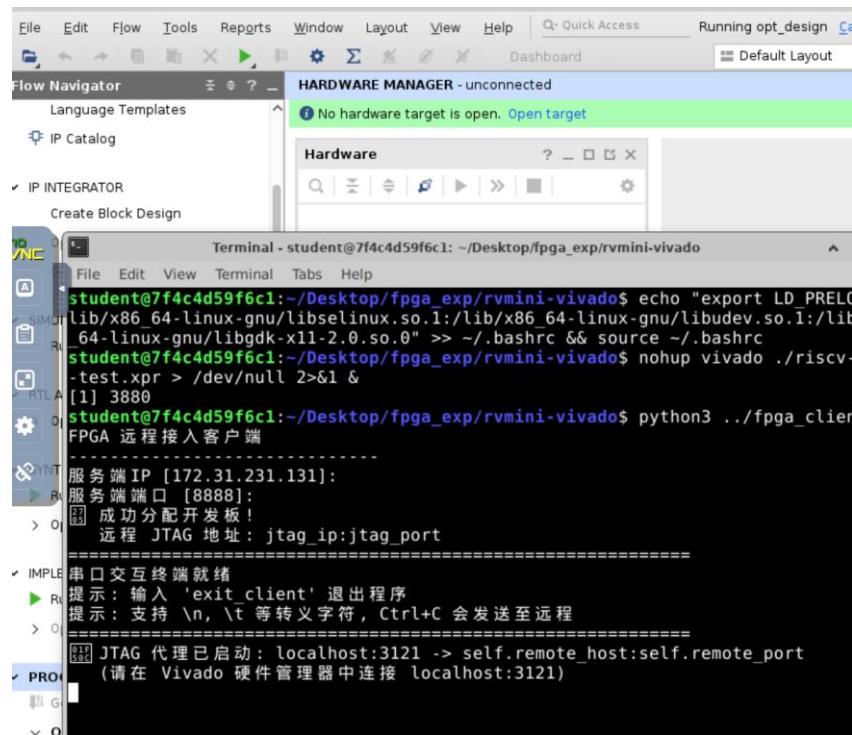
串口实验环境设置



启动 vivado 打开项目，进行比特流生成



申请并连接开发板

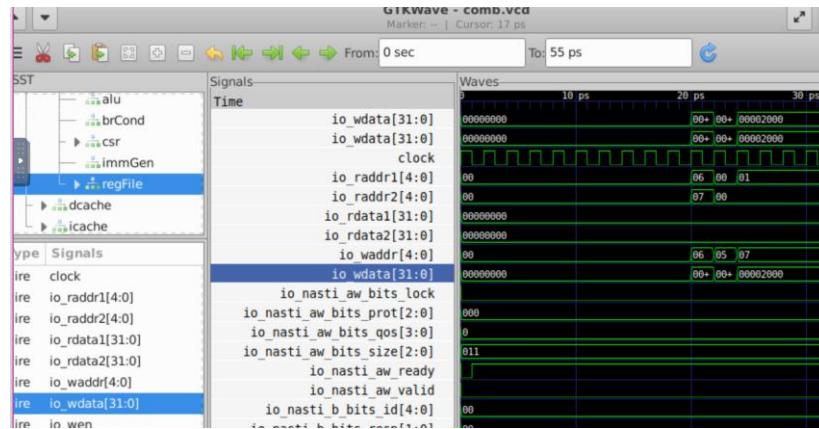


成功连接并可使用

```
Terminal - student@7f4c4d59f6c1: ~/Desktop/fpga_exp/rvmini-vivado
File Edit View Terminal Tabs Help
64-linux-gnu/libgdk-x11-2.0.so" >> ~/.bashrc && source ~/.bashrc
student@7f4c4d59f6c1:~/Desktop/fpga_exp/rvmini-vivado$ nohup vivado ./ris
-test.xpr > /dev/null 2>&1 &
[1] 3880
student@7f4c4d59f6c1:~/Desktop/fpga_exp/rvmini-vivado$ python3 ../fpga_cl
FPGA 远程接入客户端

服务端 IP [172.31.231.131]:
服务端端口 [8888]:
[?] 成功分配开发板!
远程 JTAG 地址: jtag_ip:jtag_port
=====
串口交互终端就绪
提示: 输入 'exit_client' 退出程序
提示: 支持 '\n, \t 等转义字符, Ctrl+C 会发送至远程
=====
[?] JTAG 代理已启动: localhost:3121 -> self.remote_host:self.remote_port
(请在 Vivado 硬件管理器中连接 localhost:3121)
Hello from RISC-V Mini UART!
Type something and I will echo it back: hello
echo: hello
hello,FPGA
echo: hello,FPGA
```

五、实验结果



从波形图中可以看出，comb 指令将拼接后的结果 0x00002000 写回到了 5 号寄存器中，故该指令执行正常。

```
Terminal - student@7f4c4d59f6c1: ~/Desktop/fpga_exp/rvmini-vivado
File Edit View Terminal Tabs Help
-test.xpr > /dev/null 2>&1 &
[1] 3880
student@7f4c4d59f6c1:~/Desktop/fpga_exp/rvmini-vivado$ python3 ../fpga_cl
FPGA 远程接入客户端

服务端 IP [172.31.231.131]:
服务端端口 [8888]:
[?] 成功分配开发板!
远程 JTAG 地址: jtag_ip:jtag_port
=====
串口交互终端就绪
提示: 输入 'exit_client' 退出程序
提示: 支持 '\n, \t 等转义字符, Ctrl+C 会发送至远程
=====
[?] JTAG 代理已启动: localhost:3121 -> self.remote_host:self.remote_port
(请在 Vivado 硬件管理器中连接 localhost:3121)
Hello from RISC-V Mini UART!
Type something and I will echo it back: hello
echo: hello
hello,FPGA
echo: hello,FPGA
exit_client
客户端已退出
```

可成功使用开发板

六、实验总结与体会

通过本次实验，我深入了解了 RISC-V mini 处理器的架构和数据通路的设计，学会了如何新增指令并修改处理器的实现。通过对 comb 指令的设计与测试，掌握了 Chisel 中指令译码和数据通路的修改方法。实验中，成功地将两个寄存器的不同部分拼接成一个 32 位值，并验证了指令功能的正确性。通过调试和仿真波形的分析，进一步加深了对硬件设计的理解和掌握。

指导教师批阅意见:

成绩评定:

指导教师签字:

年 月 日

备注:

注: 1、报告内的项目或内容设置, 可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。