实验五 数据通路与控制单元设计

**一、实验目的**

1 掌握Verilog语言和Vivado、Logisim开发平台的使用；

2 掌握数据通路与控制单元的设计和测试方法。

**二、实验内容**

1 数据通路的设计；

2 控制单元的设计。

**三、实验要求**

1 掌握Vivado或Logisim开发工具的使用，掌握以上电路的设计和测试方法；

2 记录设计和调试过程（Verilog代码/电路图/表达式/真值表，Vivado仿真结果，Logisim验证结果等）；

3 分析Vivado仿真波形/Logism验证结果，注重输入输出之间的对应关系。

**四、实验过程及分析**

1、MainDec（主译码器）

设计文件：  
 module MainDec(

input [5:0] Op,

output MemToReg, MemWrite,

output Branch, ALUSrc,

output RegDst, RegWrite,

output Jump,

output [1:0] ALUOp);

reg[8:0] Controls;

assign{RegWrite,RegDst,ALUSrc,Branch,MemWrite,MemToReg,Jump,ALUOp}=Controls;

always@(\*)

case(Op)

6'b000000: Controls <=9'b110000010;// RTYPE

6'b100011: Controls<=9'b101001000;//LW

6'b101011: Controls <=9'b001010000;//SW

6'b000100: Controls <=9'b000100001;//EEQ

6'b001000: Controls<=9'b101000000;//ADDI

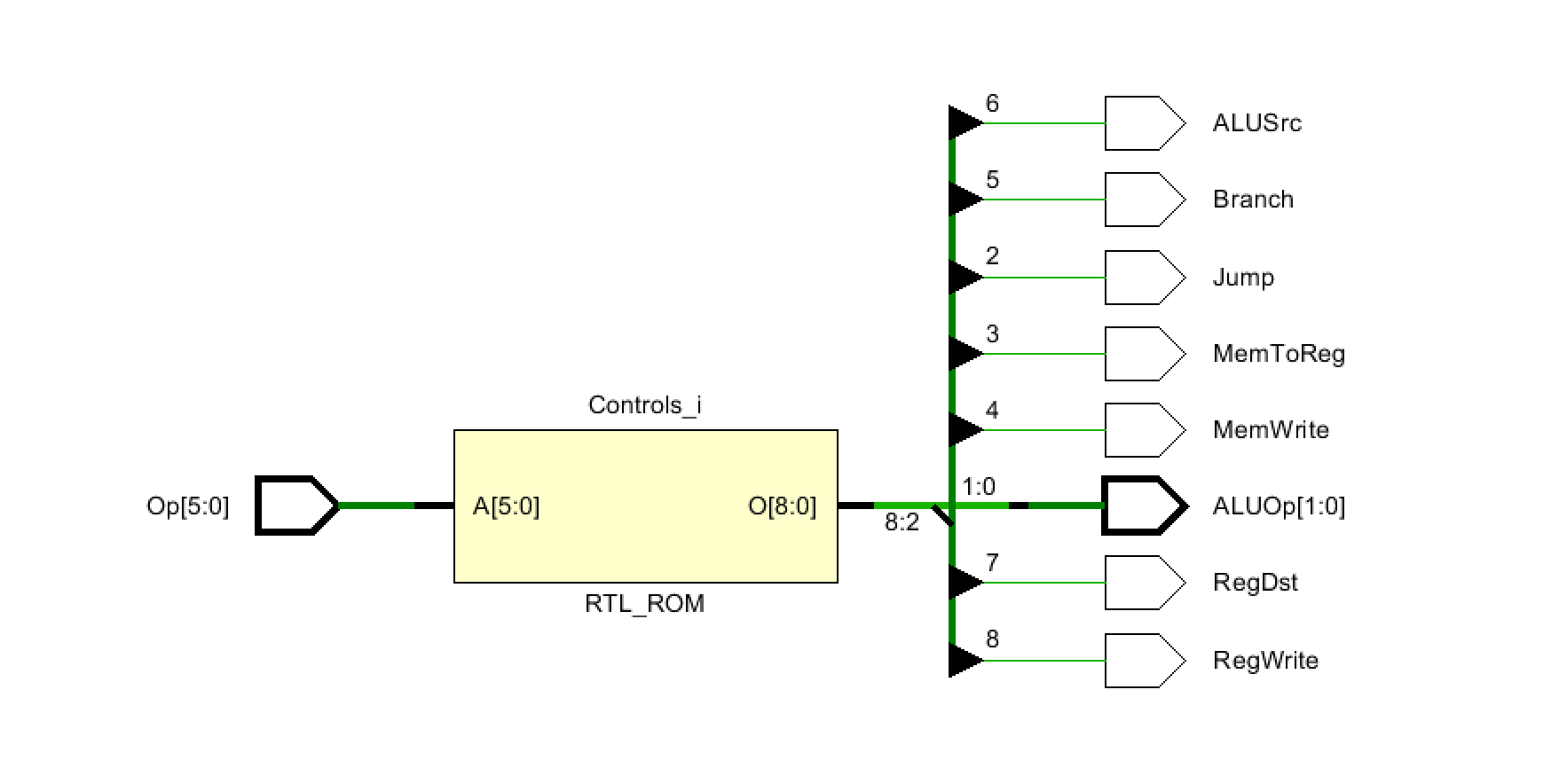
6'b000010: Controls <=9'b000000100;//J

default: Controls<=9'bxxxxxxxxx;//illegal Op

endcase

endmodule

逻辑图：

仿真文件：

module sim\_MainDec();

reg [5:0] Op;

wire MemToReg, MemWrite;

wire Branch, ALUSrc;

wire RegDst, RegWrite;

wire Jump;

wire [1:0] ALUOp;

MainDec main(Op, MemToReg, MemWrite, Branch, ALUSrc, RegDst, RegWrite, Jump, ALUOp);

initial begin

Op = 0;

repeat(10) begin #10 Op = 6'b100011;

#10 Op = 6'b101011;

#10 Op = 6'b000100;

#10 Op = 6'b001000;

#10 Op = 6'b000010;

#10 Op = 6'b000000;

end

end

endmodule

代码分析：

遍历设计文件中OP可能的所有情况，以观察译码结果是正确。

时序图：

1. ALUDec

设计模块：

module ALUDec(//ALU

input [5:0] Funct,

input [1:0] ALUOp,

output reg[2:0] ALUControl);

always@(\*)

case(ALUOp)

2'b00: ALUControl<=3'b010;// add(for lw/sw/addi)

2'b01: ALUControl <=3'b110;//sub(for beq)

default: case(Funct) //R-type Instructions

6'b100000: ALUControl<=3'b010; //add

6'b100010: ALUControl <=3'b110;//sub

6'b100100: ALUControl <=3'b000;//and

6'b100101: ALUControl<=3'b001;//or

6'b101010: ALUControl<=3'b111;//slt

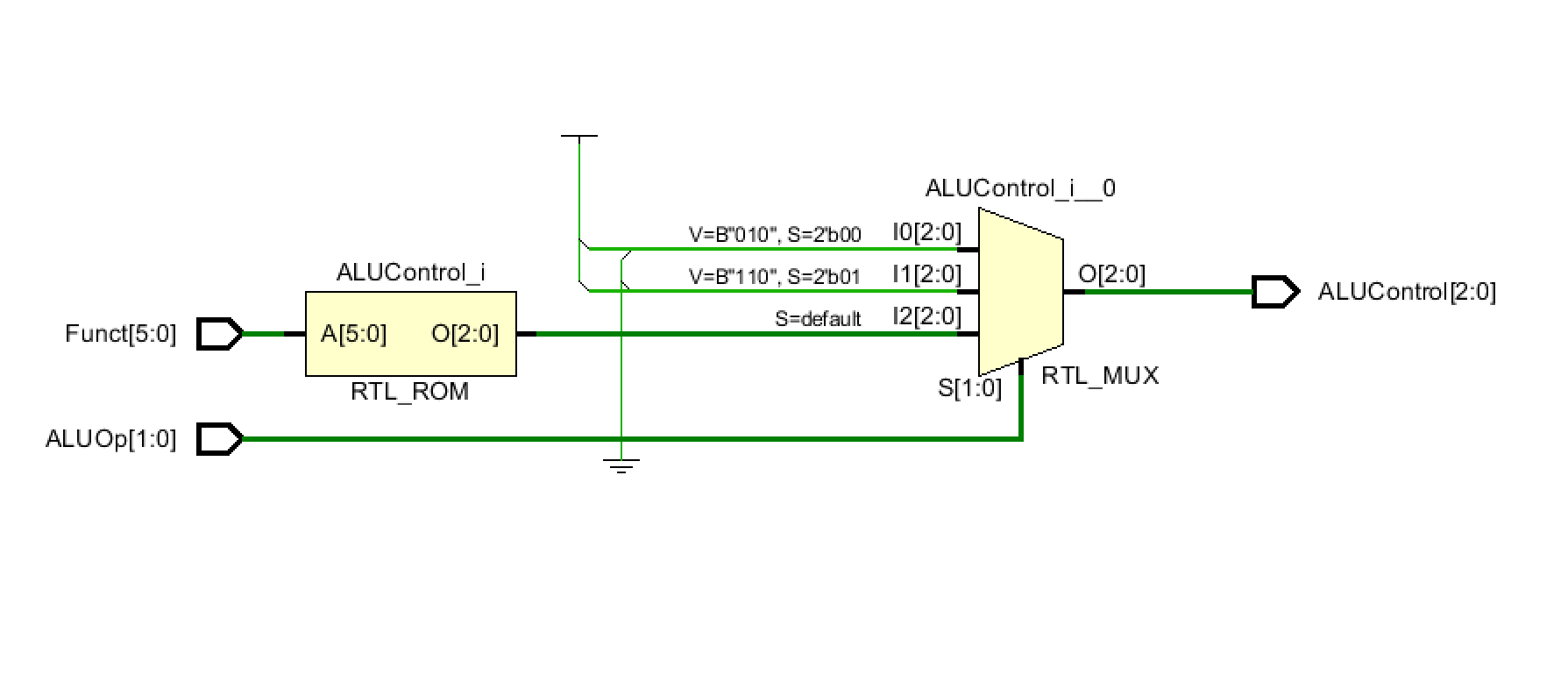
default: ALUControl<=3'bxxx;//???

endcase

endcase

endmodule

逻辑图：



仿真文件：

module sim\_ALUDec();

reg [5:0]Funct;

reg [1:0]ALUOp;

wire [2:0]ALUControl;

ALUDec alu(Funct, ALUOp, ALUControl);

initial begin

ALUOp = 0; Funct = 6'b100000;

fork

repeat(20) #30 ALUOp = ALUOp + 1;

repeat(20) begin

#10 Funct = 6'b100010;

#10 Funct = 6'b100100;

#10 Funct = 6'b100101;

#10 Funct = 6'b101010;

#10 Funct = 6'b100000;

end

join

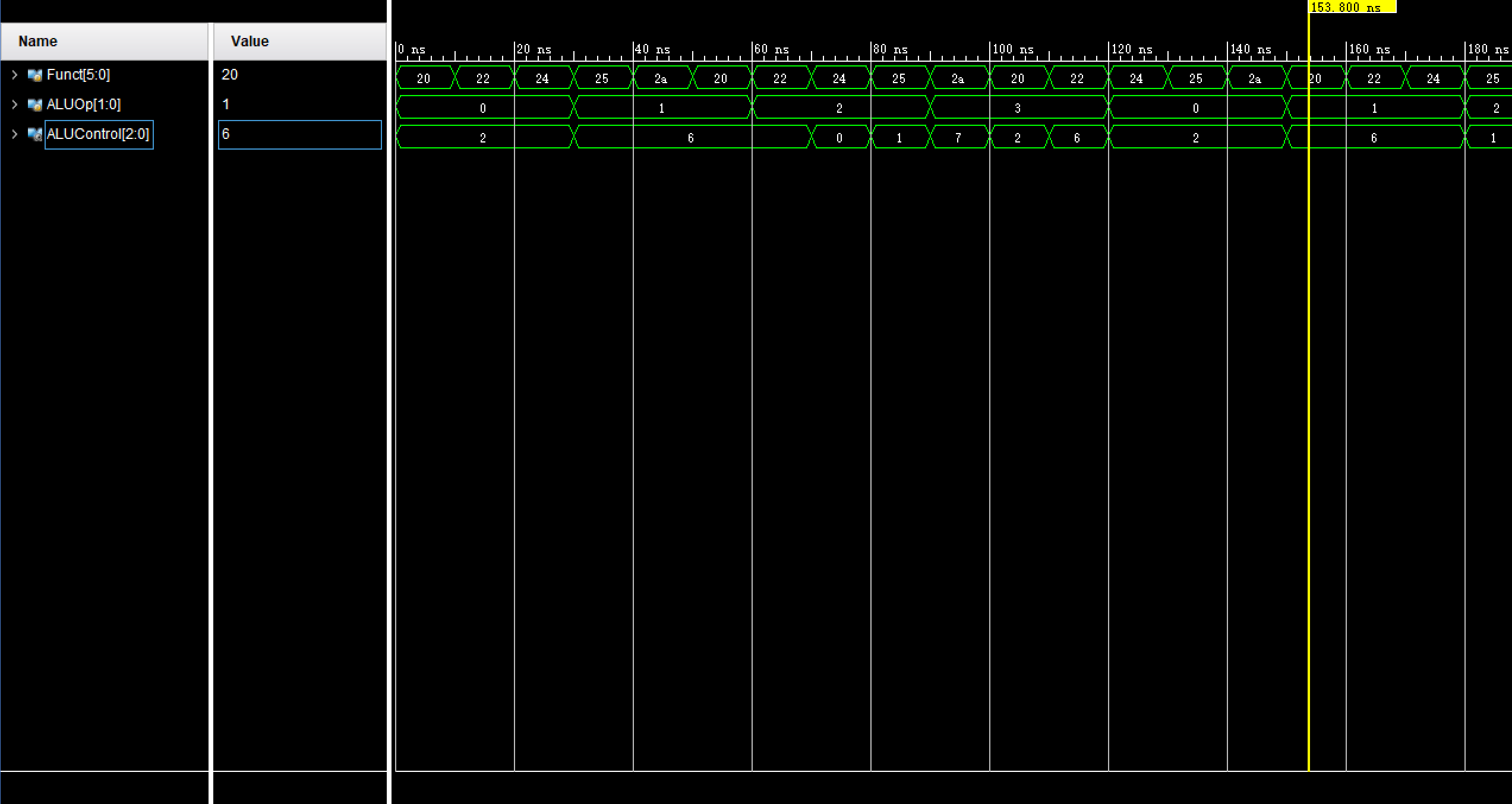
end

endmodule

代码分析：

该仿真文件的设计思路与sim\_ALUDec文件相同，即遍历设计文件中所有ALUOp和Funct的情况，观察译码是否正常。

时序逻辑图：

分析：

观察时序图可得，成功达到设计目标。当ALUOp值为00和01时，不受Funct取值影响；当ALUOp不为00和01时(default)，最终的ALUControl还受Funct取值影响。

1. Controller

设计文件

module Controller(

input [5:0] Op,Funct,

input Zero,

output MemToReg, MemWrite,

output PCSrc,ALUSrc,

output RegDst,RegWrite,

output Jump,

output [2:0] ALUControl);

wire[1:0] ALUOp;

wire Branch;

MainDec MainDec\_1(Op,MemToReg,MemWrite,Branch,ALUSrc,RegDst,RegWrite,Jump,ALUOp);

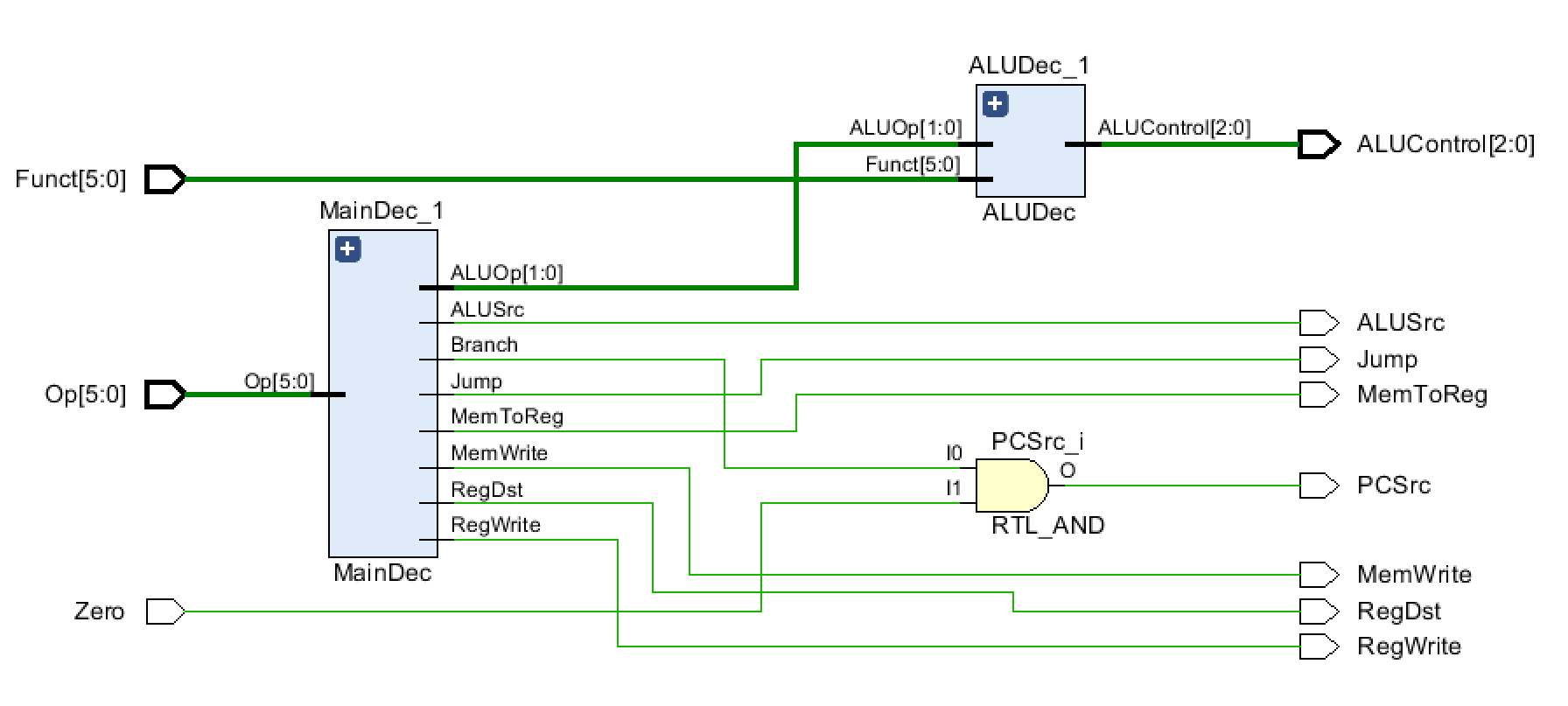
ALUDec ALUDec\_1(Funct,ALUOp,ALUControl);

assign PCSrc=Branch & Zero;

endmodule

代码分析：

实例化一个MainDec模块MainDec\_1和一个ALUDec模块ALUDec\_1，把MainDec\_1对输入Op的译码ALUOp作为输入信号对ALUDec\_1进行处理。



逻辑图：

仿真文件：

module sim\_Controller();

reg [5:0] Op;

reg [5:0] Funct;

reg Zero;

wire MemToReg, MemWrite;

wire PCSrc, ALUSrc;

wire RegDst, RegWrite;

wire Jump;

wire [2:0] ALUControl;

Controller con(Op, Funct, Zero, MemToReg, MemWrite, PCSrc, ALUSrc, RegDst, RegWrite, Jump, ALUControl);

initial begin

Op = 0; Funct = 6'b100000; Zero = 0;

fork

repeat(10) begin

#50 Op = 6'b100011;

#50 Op = 6'b101011;

#50 Op = 6'b000100;

#50 Op = 6'b001000;

#50 Op = 6'b000010;

#50 Op = 6'b000000;

end

repeat(10) begin

#10 Funct = 6'b100010;

#10 Funct = 6'b100100;

#10 Funct = 6'b100101;

#10 Funct = 6'b101010;

#10 Funct = 6'b100000;

end

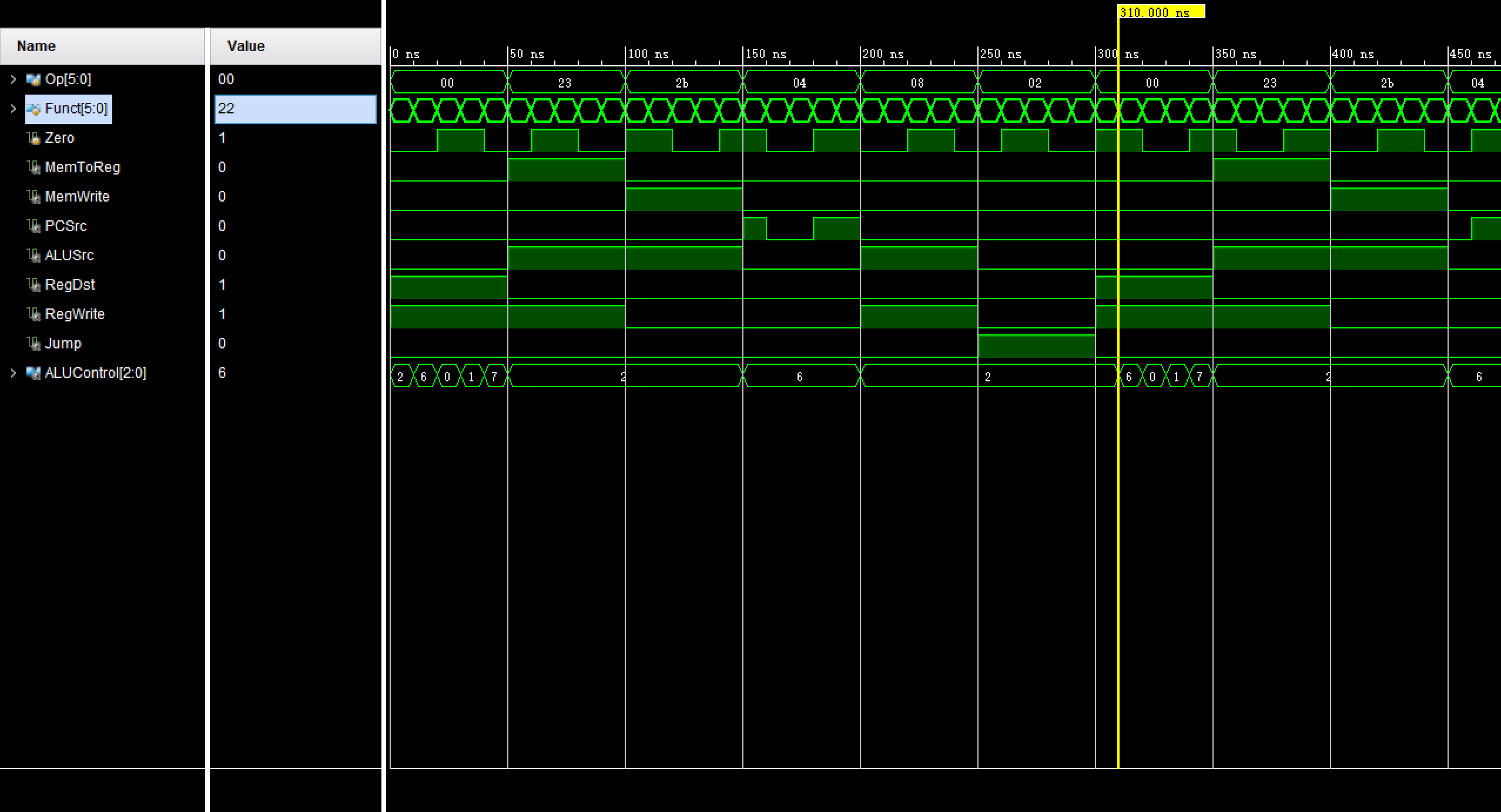
repeat(25) #20 Zero <= ~Zero;

join

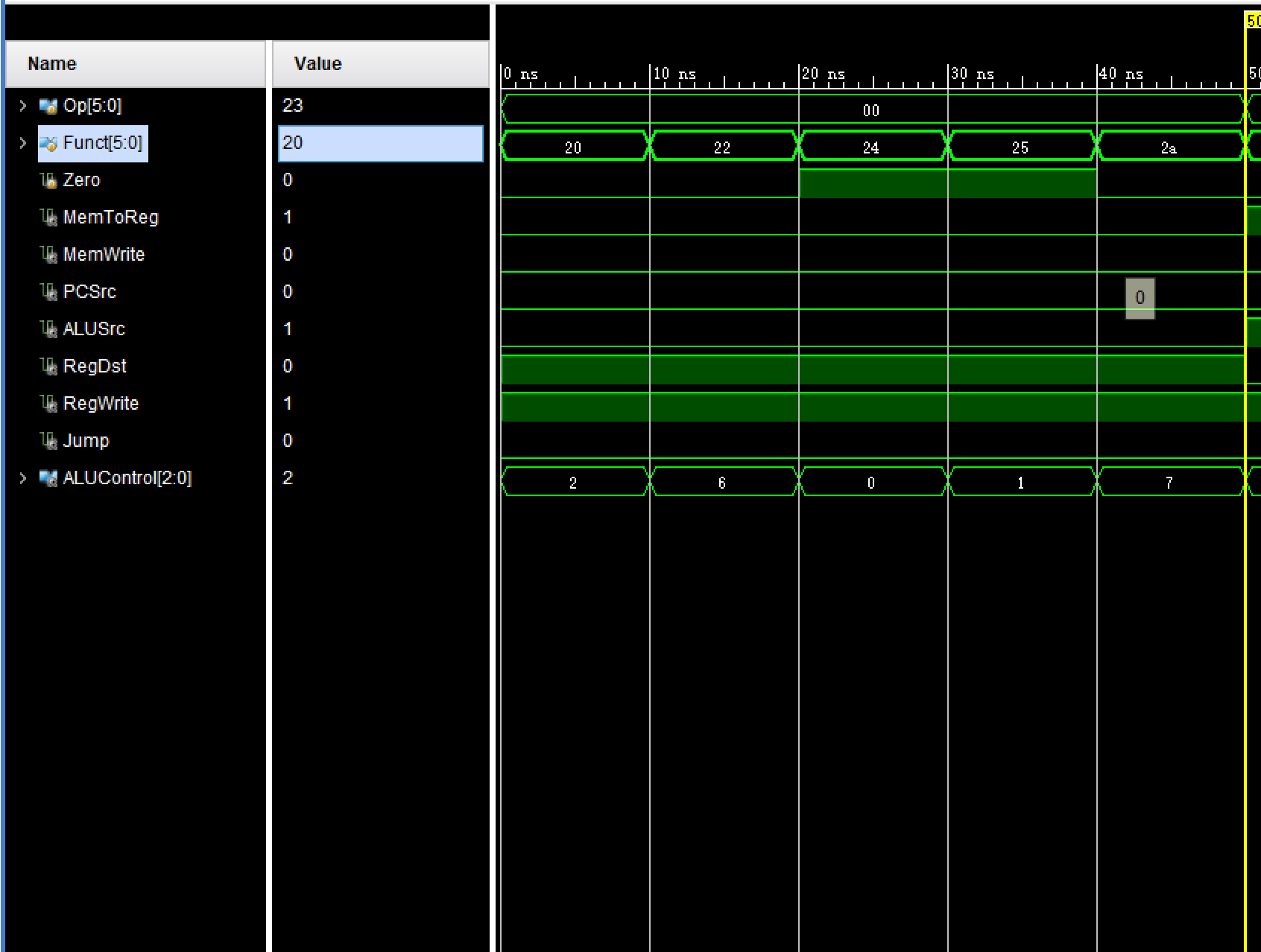
end

endmodule

时序逻辑图：



在一个OP值里Func变化：



代码分析：

思路与两个子模块的仿真文件思路基本相同，即遍历所有的合法输入值检查是否输出信号的值与所期望的一样。每50ns更新一次Op值，每10ns更新一次Funct值，由于一共有5个合法的Funct输入，所有可以遍历到所有合法输入情况。根据时序逻辑图观察可得设计文件达到目标效果。

1. **调试和心得体会**

本次实验的主要任务为为设计模块写仿真文件检验模块是否达到目标要求，由于输入输出信号繁多，且一个子模块的输出信号需要作为另一个的输入信号，这是与之前的实验不一样的地方。难点主要在在于合理设计仿真文件，遍历到所有合法情况，以观察是否能达到特定的功能。