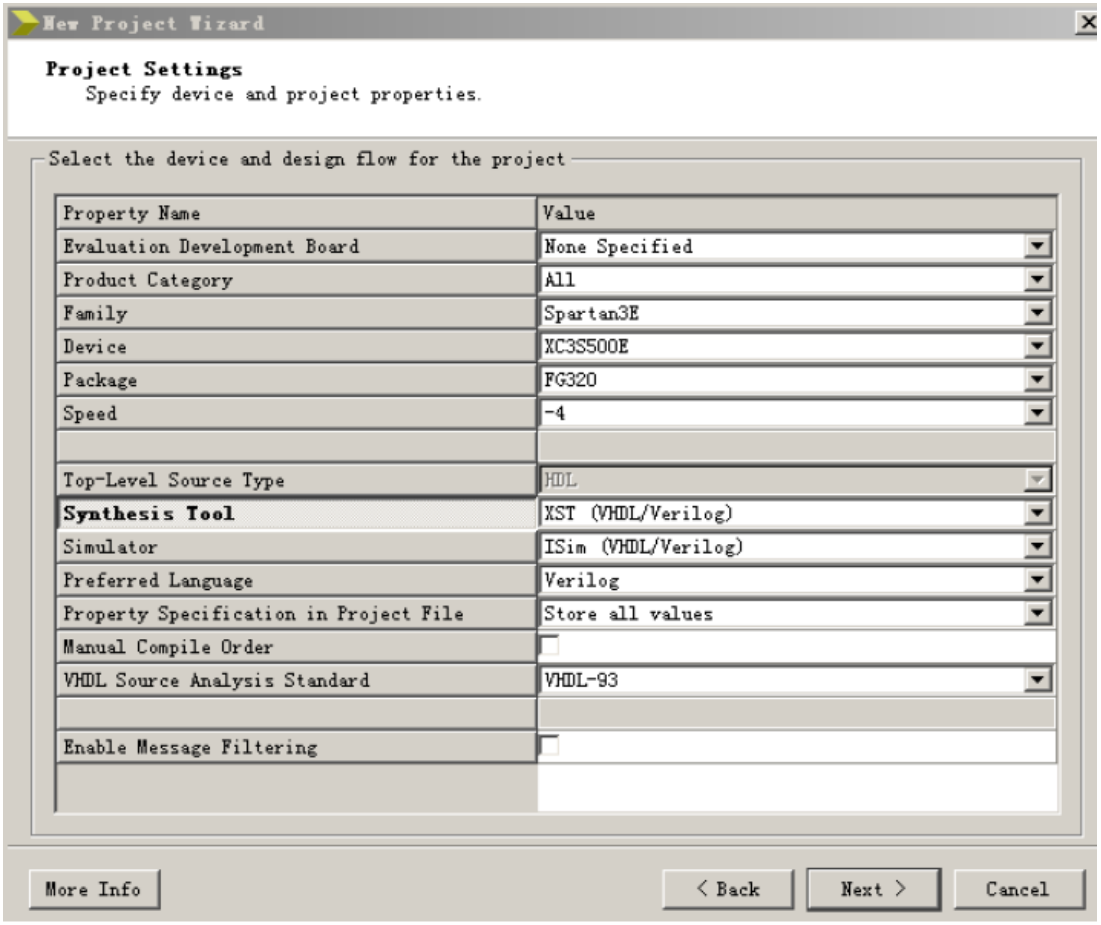
实验三实验报告

姓名：申继媛
学号：5130309194

【实验环境】



【主控制单元模块 Ctr】

模块描述——

主控制模块输入为指令的 opcode（操作码）字段，即 instructiion[31-26]。操作码经过主控制模块的译码，产生 regDst、aluSrc、memToReg、regWrite、memRead、memWrite、branch、jump 以及 aluOp 控制信号。

实验过程——

（1）按照实验指导书上如下给出的操作码与控制信号之间的对应关系，将代码写在了新建的 Ctr.v 中。这是刚接触 xilinx 软件之后的第一段自己上手的代码，主要掌握的是对 reg 型变量的定义，always@语句、begin-end 语句、case 语句的运用。

3.1.3 编写译码功能

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

主控制模块真值表（OpCode 与控制输出的编码关系）

注：Jump 指令编码是 000010，Jump 信号输出 1，其余输出 0

指令	opCode
R 型：add, sub, and, or, slt	000000
I 型：lw	100011
I 型：sw	101011
I 型：beq	000100
J 型：J	000010

指令操作码

（2）Ctr 代码截图

```
module Ctr(
    input [5:0] opCode,
    output regDst,
    output aluSrc,
    output memToReg,
    output regWrite,
    output memRead,
    output memWrite,
    output branch,
    output [1:0] aluOp,
    output jump
    );

    reg regDst;
    reg aluSrc;
    reg memToReg;
    reg regWrite;
    reg memRead;
    reg memWrite;
    reg branch;
    reg [1:0] aluOp;
    reg jump;
```

```
always @ (opCode)
begin
    case (opCode)
    6'b000010:  //jump
    begin
        regDst = 0;
        aluSrc = 0;
        memToReg = 0;
        regWrite = 0;
        memRead = 0;
        memWrite = 0;
        branch = 0;
        aluOp = 2'b00;
        jump = 1;
    end
    6'b000000:  //R type
    begin
        regDst = 1;
        aluSrc = 0;
        memToReg = 0;
        regWrite = 1;
        memRead = 0;
        memWrite = 0;
        branch = 0;
        aluOp = 2'b10;
        jump = 0;
    end
```

```
    6'b100011:  //lw
    begin
        regDst = 0;
        aluSrc = 1;
        memToReg = 1;
        regWrite = 1;
        memRead = 1;
        memWrite = 0;
        branch = 0;
        aluOp = 2'b00;
        jump = 0;
    end
    6'b101011:  //sw
    begin
        aluSrc = 1;
        regWrite = 0;
        memRead = 0;
        memWrite = 1;
        branch = 0;
        aluOp = 2'b00;
        jump = 0;
    end
```

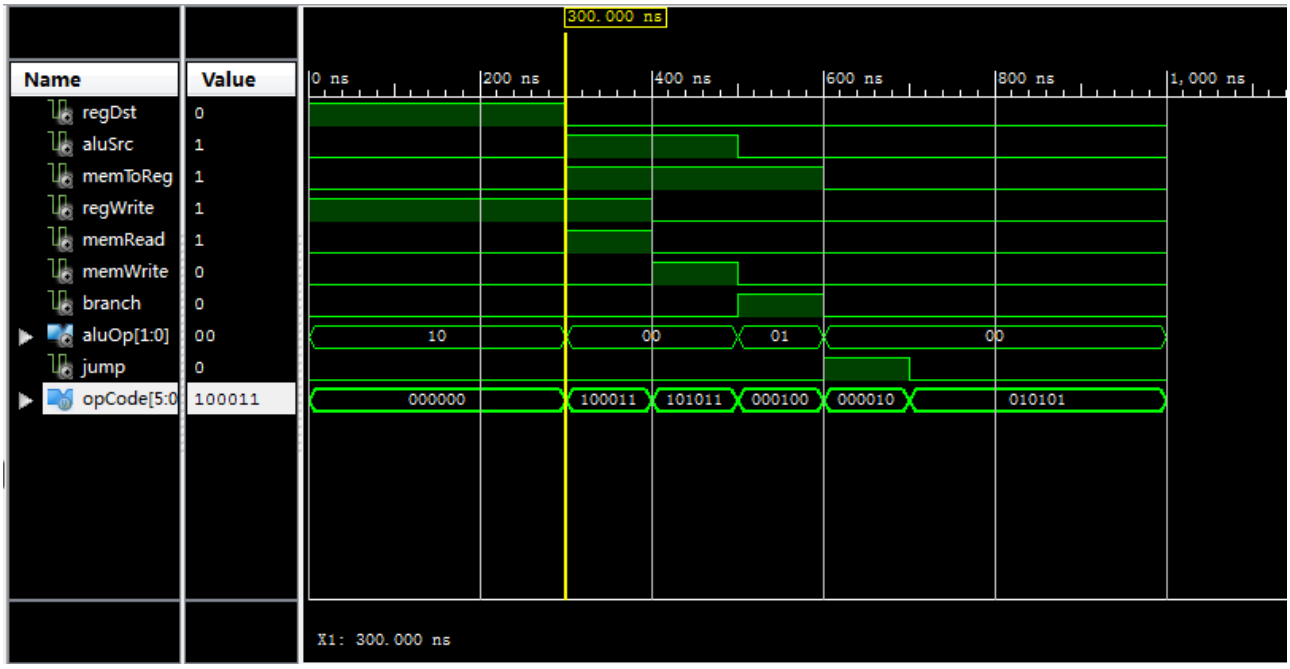
```
    6'b000100:  //beq
    begin
        aluSrc = 0;
        regWrite = 0;
        memRead = 0;
        memWrite = 0;
        branch = 1;
        aluOp = 2'b01;
        jump = 0;
    end
    default:
    begin
        regDst = 0;
        aluSrc = 0;
        memToReg = 0;
        regWrite = 0;
        memRead = 0;
        memWrite = 0;
        branch = 0;
        aluOp = 2'b00;
        jump = 0;
    end
  endcase
  end

endmodule
```

（3）test_for_Ctr 代码截图

（分别测试 R 型、lw、sw、beq、jump）

```
initial begin
    // Initialize Inputs
    opCode = 0;

    // Wait 100 ns for global reset to finish
    #100;

    // Add other stimulus here
    #100 opCode = 6'b000000;
    #100 opCode = 6'b100011;
    #100 opCode = 6'b101011;
    #100 opCode = 6'b000100;
    #100 opCode = 6'b000010;
    #100 opCode = 6'b010101;

end
```

（4）在 simulation 的 hierarchy 窗口中选中 test_for_Ctr 文件，并双击 Simulate Behavioral Model，得到仿真波形如下截图：



实验感想——

这是第一个指导书没有把所有步骤都详细说明的实验，让我更加熟悉 xilinx 软件，熟悉 verilog 语言，回顾计组知识，这个简单的小实验激发了我对接下来其它计组实验的好奇与兴趣，这才是最重要的。

【ALU 单元控制模块 AluCtr】

实验描述——

AluCtr 根据主控制模块 Ctr 输出的 AluOp 来判断指令的类型。进一步根据指令的后 6 位，即 instruction[5:0]来区分是哪一种 R 型指令。因此，AluCtr 控制模块有两个输入：2bit—AluOp 和 6bit—instructon[5:0]，一个输出 4bit—aluCtr[3:0]。

实验过程——

（1）按照实验指导书给出的如下输入输出真值表，将代码写在新建的 AluCtr.v 中，主要掌握 casex 语句的运用。

ALUOp		Funct field						
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	Operation
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

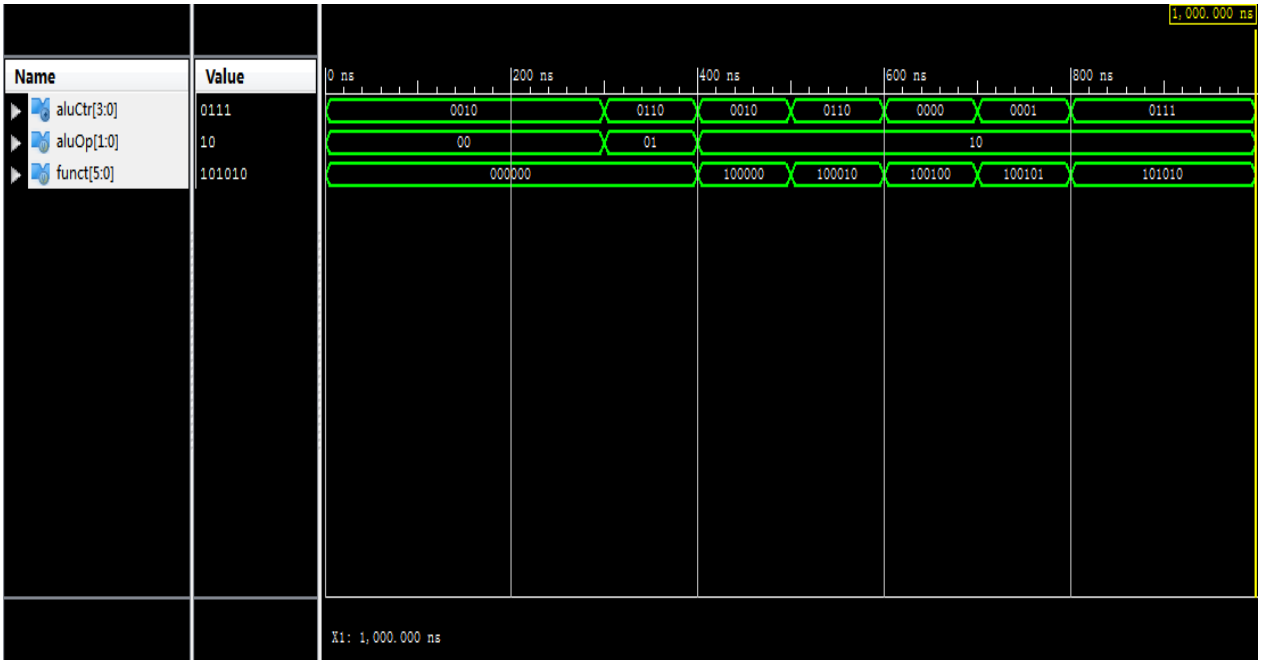
（2）AluCtr 代码截图

```
module AluCtr(
    input [1:0] aluOp,
    input [5:0] funct,
    output [3:0] aluCtr
    );

    reg [3:0] aluCtr;

    always @(aluOp or funct)
    casex({aluOp, funct})
        8'b00xxxxxx: aluCtr = 4'b0010;
        8'b01xxxxxx: aluCtr = 4'b0110;
        8'b10100000: aluCtr = 4'b0010;
        8'b10100010: aluCtr = 4'b0110;
        8'b10100100: aluCtr = 4'b0000;
        8'b10100101: aluCtr = 4'b0001;
        8'b10101010: aluCtr = 4'b0111;
        default: aluCtr = 4'b0000;
    endcase

endmodule
```

（3）test_for_AluCtr 代码截图

```
initial begin
    // Initialize Inputs
    aluOp = 0;
    funct = 0;

    // Wait 100 ns for global reset to finish
    #100;

    // Add stimulus here
    #100 aluOp = 2'b00;
    #100 aluOp = 2'b01;
    #100 aluOp = 2'b10;funct = 6'b100000;
    #100 aluOp = 2'b10;funct = 6'b100010;
    #100 aluOp = 2'b10;funct = 6'b100100;
    #100 aluOp = 2'b10;funct = 6'b100101;
    #100 aluOp = 2'b10;funct = 6'b101010;

end
```

（4）在 simulation 的 hierarchy 窗口中选中 test_for_AluCtr 文件，并双击 Simulate Behavioral Model，得到仿真波形如下截图：



实验感想——

AluCtr 模块的代码比主控制 Ctr 模块还要简单一些，跟着实验指导书一步步做就能做出来，主要是巩固一下计组的知识。

【ALU 单元模块 Alu】

实验描述——

根据控制信号 aluCtr，对两个输入做对应的操作。输出结果为 aluRes。当为减法操作且 aluRes 为零时将 zero 置为零。即 Alu 有三个输入：控制信号 aluCtr 和两个输入数据（input1、input2），有两个输出：运算结果 aluRes 和零信号 zero。

实验过程——

（1） 按照实验指导书上给出如下输入 4bit—控制信号 aluCtr 和 alu 操作的对应关系，可实现 Alu 功能在新建的 Alu.v 中。

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

（2） Alu 代码截图：

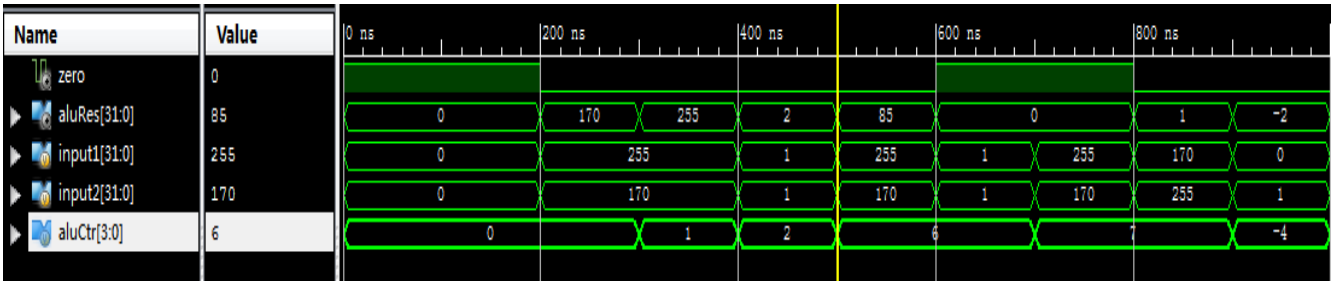
```
module Alu(
    input [31:0] input1,
    input [31:0] input2,
    input [3:0] aluCtr,
    output zero,
    output [31:0] aluRes
    );

    reg zero;
    reg [31:0] aluRes;

    always @ (input1 or input2 or aluCtr)
    begin
        if (aluCtr == 4'b0010)//add
            aluRes = input1 + input2;
        else if (aluCtr == 4'b0110)//sub
        begin
            aluRes = input1 - input2;
        end
        else if (aluCtr == 4'b0000)//and
            aluRes = input1 & input2;
        else if (aluCtr == 4'b0001)//or
            aluRes = input1 | input2;
        else if (aluCtr == 4'b0111)//slt
        begin
            if(input1 < input2)
                aluRes = 1;
            else
                aluRes = 0;
        end
        else if (aluCtr == 4'b1100)//nor
            aluRes = ~(input1 | input2);
            if(aluRes == 0)
                zero = 1;
            else
                zero = 0;
end

endmodule
```

（3） test_for_Alu 代码截图：

```
initial begin
    // Initialize Inputs
    input1 = 0;
    input2 = 0;
    aluCtr = 0;

    // Wait 100 ns for global reset to finish
    #100;

    // Add stimulus here
    #100 aluCtr = 4'b0000; input1 = 255; input2 = 170;
    #100 aluCtr = 4'b0001; input1 = 255; input2 = 170;
    #100 aluCtr = 4'b0010; input1 = 1; input2 = 1;
    #100 aluCtr = 4'b0110; input1 = 255; input2 = 170;
    #100 aluCtr = 4'b0110; input1 = 1; input2 = 1;
    #100 aluCtr = 4'b0111; input1 = 255; input2 = 170;
    #100 aluCtr = 4'b0111; input1 = 170; input2 = 255;
    #100 aluCtr = 4'b1100; input1 = 0; input2 = 1;

end
```

代码解释：以上测试代码分别是 255 AND 170； 255 OR 170；1 ADD 1；255 SUB 170；1 SUB 1； 255 slt 170； 170 slt 255； 0 NOR 1；

（4）在 simulation 的 hierarchy 窗口中选中 test_for_Alu 文件，并双击 Simulate Behavioral Model，得到仿真波形如下截图：



根据（3）中的代码解释，预计的仿真输出有：

从 200ns 处开始，第一个 100ns 内有 aluCtr 为 4’b0000，即 0；有 255 AND 170，即 8’b11111111 AND 8’b10101010；输出 aluRes 为 8’b10101010，即 170；输出 zero 为 0。

第二个 100ns 内有 aluCtr 为 4’b0001，即 1；有 255 OR 170，即 8’b11111111 OR 8’b10101010；输出 aluRes 为 8’b11111111，即 255；输出 zero 为 0。

第三个 100ns 内有 aluCtr 为 4’b0010，即 2；有 1 ADD 1，即 1+1；输出 aluRes 为 2。

第四个 100ns 内有 aluCtr 为 4’b0110，即 6；有 255 SUB 170，即 8’b11111111 - 8’b10101010；输出 aluRes 为 8’b01010101，即 85；输出 zero 为 0。

第五个 100ns 内有 aluCtr 为 4’b0110，即 6；有 1 SUB 1，即 1-1；输出 aluRes 为 0；输出 zero 为 1；

第六个 100ns 内有 aluCtr 为 4’b0111，即 7；有 255 slt 170，即 aluRes=1 when 255<170；输出 aluRes 为 0；输出 zero 为 1；

第七个 100s 内有 aluCtr 为 4’b0111，即 7；有 170 slt 255，即 aluRes=1 when 170<255；输出 aluRes 为 1；输出 zero 为 0。

第八个 100ns 内有 aluCtr 为 4’b1100，即-4（signed number）；有 0 NOR 1；输出 aluRes=-2；输出 zero=0。