计算机组成原理 实验报告

姓名：魏钊　学号：PB18111699　实验日期：2020-7-5

一、实验题目：

实验六 综合设计

二、实验目的：

1. 理解计算机系统的组成结构和工作原理；
2. 理解计算机总线和接口的结构和功能；
3. 掌握软硬件综合系统的设计和调试方法。

三、实验平台：

ISE / Vivado（暂不支持其他 Verilog HDL 开发环境的检查）

四、实验过程：

利用单周期 CPU 实现斐波那契数列

```
5  module ALU
6      #(parameter N = 32)
7      (
8      input [N-1:0] a,b,
9      input [2:0] ALUOP,
10
11     output reg [N-1:0] y,
12     output reg cf,of,zero,sf
13     );
14
15 always@(*)
16     begin
17         case(ALUOP)
18             3'b000:  {cf,y} = a+b;
19             3'b001:  {cf,y} = a-b;
20             3'b010:  {cf,y} = a&b;
21             3'b011:  {cf,y} = a|b;
22             3'b100:  {cf,y} = a^b;
23             default: y = 4'bz;
24         endcase
25         zero = (y == 0) ? 1 : 0;
26     end
27 endmodule
```

```v
module RegFile(
input clk,                          //时钟（上升沿有效）
input [4:0] ra0,                    //读端口0地址
input [4:0] ra1,                    //读端口1地址
input [4:0] ra2,                   //FIB
output [31:0] rd0,   //读端口0数据
output [31:0] rd1,   //读端口1数据
output [31:0] rd2, //FIB
input [4:0] wa,                     //写端口地址
input we,                         //写使能，高电平有效
input [31:0] wd          //写端口数据
);
reg [31:0]REG[0:31];     //定义寄存器堆的寄存器
integer i;//初始化寄存器堆数据为0
initial
    for(i = 0;i < 32;i = i + 1)
        REG[i] <= 0;//同步写操作，需时钟控制
always@(negedge clk)
    begin
    if(wa > 0)
    begin
        if(we)
            REG[wa] <= wd;
    end
    end
//异步读操作，不需时钟控制，组合逻辑
assign rd0 = REG[ra0];
assign rd1 = REG[ra1];
assign rd2 = REG[ra2];
```

elrbombre

```v
module PC(
        input clk,rst,
        input PCWre,                //是否接受新的地址。0-不更改；1-可以更改
        //input [1:0] PCSrc,                //数据选择器输入
        input [31:0] nextPC,   //新指令地址
        output reg [31:0] curPC //当前指令的地址
    );
    initial begin
        curPC <= 32'b0;
    end
    always@(posedge clk)
    begin
        if(rst)
            begin
                curPC <= 32'b0;
            end
        else
            begin
                if(PCWre) // PCWre == 1
                    begin
                        curPC <= nextPC;
                    end
                else    // PCWre == 0, halt
                    begin
                        curPC <= curPC;
                    end
            end
    end
endmodule
```

```verilog
module pcADD(
        input clk,rst,                  //时钟
        input Branch,zero,Jump,                 //数据选择器输入
        input [31:0] Ex_Imm,   //偏移量
        input [25:0] addr,
        input [31:0] curPC,
        output reg[31:0] nextPC  //新指令地址
    );

    initial begin
        nextPC <= 32'b0;
    end
    reg [31:0] pc;
    always@(negedge clk)
    begin
        if(rst) begin
            nextPC <= 32'b0;
        end
        else begin
            pc <= curPC + 32'b0100;
            case({Branch&zero,Jump})
                2'b00: nextPC <= curPC + 32'b0100;
                2'b10: nextPC <= curPC + 32'b0100 + (Ex_Imm<<2);
                2'b01: nextPC <= {pc[31:28],addr,2'b00};
                default: ;
            endcase
        end
    end
endmodule
```

```verilog
module ControlUnit(
        input zero,          //ALU运算结果是否为0，为0时候为1
        input [5:0] op,      //指令的操作码
        //output reg PCWre,        //PC是否更改的信号量，为0时候不更改，否则可以更改
        //output reg ExtSel,       //立即数扩展的信号量，为0时候为0扩展，否则为符号扩展
        //output reg InsMemRW,     //指令寄存器的状态操作符，为0的时候写指令寄存器，否则为读指令寄存器
        output  RegDst,       //写寄存器组寄存器的地址，为0的时候地址来自rt，为1的时候地址来自rd
        output  RegWrite,       //寄存器组写使能，为1的时候可写
        output  ALUSrc,      //控制ALU数据B的选择端的输入，为0的时候，来自寄存器堆data2输出，为1时候来自扩展过的立即数
        output  Jump,Branch,   //获取下一个pc的地址的数据选择器的选择端输入
        output  [2:0]ALUOp,   //ALU 8种运算功能选择(000-111)
        //output reg mRD,          //数据存储器读控制信号，为0读
        output  MemWrite,          //数据存储器写控制信号，为1写
        output  MemtoReg    //数据保存的选择端，为0来自ALU运算结果的输出，为1来自数据寄存器（Data MEM）的输出
    );
    reg [9:0] code;


    always@(*)
    begin
        //PCWre = (op == 6'b111111) ? 0 : 1;   //halt
        //InsMemRW = (op == 6'b111111) ? 0 : 1;
        case(op)
            //add
            6'b000000: code = 10'b1_1_0_0_000_0_0;
            //addi
            6'b001000: code = 10'b0_1_1_0_000_0_0;
            //lw
            6'b100011: code = 10'b0_1_1_0_000_0_1;
```

```
            //sw
            6'b101011: code = 10'b0_0_1_0_0_000_1_0;
            //beq
            6'b000100: code = 10'b0_0_0_0_1_001_0_0;
            //j
            6'b000010: code = 10'b0_0_0_1_0_000_0_0;
            default:
            begin
                code = 10'b0000000000;
            end
        endcase
    end
    assign {RegDst,RegWrite,ALUSrc,Jump,Branch,ALUOp,MemWrite,MemtoReg} = code;
endmodule
```

```
module InsCut(
        input [31:0] instruction,
        output reg[5:0] op,
        output reg[4:0] rs,
        output reg[4:0] rt,
        output reg[4:0] rd,
        //output reg[4:0] sa,
        output reg[15:0] immediate,
        output reg[25:0] addr
    );
    initial begin
        op = 5'b00000;
        rs = 5'b00000;
        rt = 5'b00000;
        rd = 5'b00000;
    end
    always@(*)
    begin
        op = instruction[31:26];
        rs = instruction[25:21];
        rt = instruction[20:16];
        rd = instruction[15:11];
        //sa = instruction[10:6];
        immediate = instruction[15:0];
        addr = instruction[25:0];
    end
endmodule
```

```
module EDG(
    input clk,
    input button,
    output button_edg
    );
    reg button_r1,button_r2;
    always@(posedge clk)
        button_r1<=button;
    always@(posedge clk)
        button_r2<=button_r1;
    assign button_edg=button_r1&(~button_r2);
endmodule
```

```
module SingleCPU(
        input clk,rst,
        input [31:0] F,F_Addr,   //斐波那契数据和写入地址
        input Run,Step,F_Write   //控制CPU运行和F0，F1写入Mem的信号
         );
    //PC相关变量
    wire [31:0] curPC,nextPC;

    //指令相关变量
    wire [31:0] instruction;
    wire [5:0] op;
    wire [4:0] rs,rt,rd;
    wire [15:0] immediate;
    wire [25:0] addr;

    //各个存储器，多选器，ALU接口
    wire [4:0] WriteReg;                    //寄存器堆写回地址
    wire [31:0] ALUSrcB;                    //ALU第二输入
    wire [31:0] RF_ReadData1,RF_ReadData2,ALUout;//寄存器堆输出以及ALU输出
    wire [31:0] RF_WriteData,WriteData,ReadData;          //数据存储器输入输出
    wire [31:0] Ex_Imm;                     //位扩展后的Imm
    //控制器相关变量
    wire zero,PCWre,RegDst,RegWrite,ALUSrc,Jump,Branch,MemWrite,MemtoReg;
    wire [2:0] ALUOp;

    //斐波那契数列相关变量
    wire Step_Edg;

    wire [31:0] MemData,MemAddr,F_RegData;//DataMem的写入数据和地址
    wire [4:0] F_RegAddr;

    EDG edg(clk, Step, Step_Edg);
    assign F_RegAddr = 5'b01010;
    assign MemData = F_Write ? F : RF_ReadData2;
    assign MemAddr = F_Write ? F_Addr : ALUout;
    assign PCWre = Run ? 1 : (Step_Edg);

    assign Ex_Imm = immediate[15] ? {16'hffff,immediate} : {16'h0000,immediate};
    assign WriteReg = RegDst ? rd : rt;
    assign ALUSrcB = ALUSrc ? Ex_Imm : RF_ReadData2;
    assign RF_WriteData = MemtoReg ? ReadData : ALUout;
    assign WriteData = RF_ReadData2;

    InsCut IC(.instruction(instruction), .op(op), .rs(rs), .rt(rt), .rd(rd),
    .addr(addr), .immediate(immediate));

    ControlUnit CU(.op(op), /*.PCWre(PCWre),*/ .RegDst(RegDst), .RegWrite(RegWrite), .ALUSrc(ALUSrc),
    .Jump(Jump), .Branch(Branch), .ALUOp(ALUOp), .MemWrite(MemWrite), .MemtoReg(MemtoReg));

    pcADD pcadd(.clk(clk), .rst(rst), .Branch(Branch), .zero(zero), .Jump(Jump),
    .Ex_Imm(Ex_Imm), .addr(addr), .curPC(curPC), .nextPC(nextPC));

    PC PC(.clk(clk),.rst(rst), .PCWre(PCWre), .nextPC(nextPC), .curPC(curPC));

    RegFile RF(.clk(clk), .ra0(rs), .ra1(rt), .ra2(F_RegAddr), .rd0(RF_ReadData1), .rd1(RF_ReadData2), .rd2(F_RegData),
    .wa(WriteReg), .we(RegWrite), .wd(RF_WriteData));

    ALU alu(.a(RF_ReadData1), .b(ALUSrcB), .ALUOP(ALUOp), .y(ALUout), .zero(zero));
```

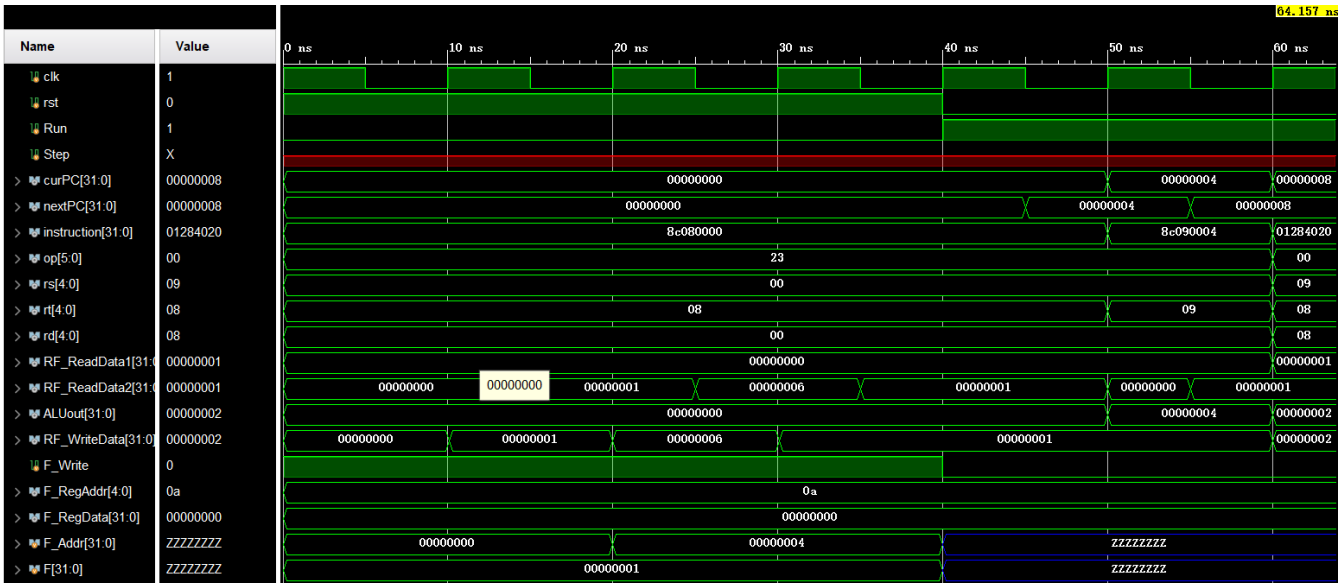
```
62
63      InsMem IM(.a(curPC[9:2]), .spo(instruction));
64
65      DataMem DM(.we(MemWrite || F_Write), .a(MemAddr[9:2]/*ALUout[9:2]*/), .clk(clk), .d(MemData/*RF_ReadData2*/), .spo(ReadData));
66
67
68  endmodule
```

```
23      module testbench(
24          );
25          reg clk,rst;
26          reg Run,Step,F_Write;
27          reg [31:0] F_Addr,F;
28          SingleCPU CPU(.clk(clk), .rst(rst), .F(F), .F_Addr(F_Addr), .Run(Run), .Step(Step), .F_Write(F_Write));
29
30      initial clk = 1;
31      always #5 clk = ~clk;
32      initial
33          begin
34              rst = 1;
35              #40 rst = 0;
36          end
37
38      initial
39      begin
40          Run = 0;
41          #40
42          Run = 1;
43      end
44      /*
45      initial
46      begin
47          Step = 0;
48          #40
49          Step = 1;
50          #20
51          Step = 0;
```

```
54      initial
55      begin
56          F_Write = 1;
57          #40 F_Write = 0;
58      end
59
60      initial
61      begin
62          F_Write = 1;
63          #40 F_Write = 0;
64      end
65
66      initial
67      begin
68          F_Addr = 32'h0000;
69          #20
70          F_Addr = 32'h0004;
71          #20
72          F_Addr = 32'hzzzz;
73      end
74
75      initial
76      begin
77          F = 32'h0001;
78          #20
79          F = 32'h0001;
80          #20
81          F = 32'hzzzz;
82      end
```

```
83      endmodule
```

五、实验结果：



六、心得体会：

通过本次实验理解了计算机系统的组成结构和工作原理、计算机总线和接口的结构和功能，掌握了软硬件综合系统的设计和调试方法。