计算机组成原理实验报告

实验题目：流水线CPU设计
学生姓名：刘恒远
学生学号：PB20111642
完成日期：2022.5.19

实验目的

•理解流水线CPU的结构和工作原理

•掌握流水线CPU的设计和调试方法，特别是流水线中数据相关和控制相关的处理

•熟练掌握数据通路和控制器的设计和描述方法

实验环境

vivado，FPGAOL

实验过程

题目一：

满足写优先的寄存器堆：

```
module Registers//寄存器堆
(
    input we,
    input clk,
    input [4:0] wa,
    input [31:0] wd,
    input [4:0] ra1,ra2,ra3,
    output [31:0] rd1,rd2,rd3
);
    reg [31:0] regfile[0:31];
    integer i;
    initial begin
        for(i=0;i<32;i=i+1) regfile[i]=0;
    end

    always @(posedge clk)
    begin
        if(we && wa!=0 ) regfile[wa]<=wd;
        else regfile[wa]<=regfile[wa];
    end

    assign rd1 = (we && wa == ra1) ? wd : regfile[ra1];
    assign rd2 = (we && wa == ra2) ? wd : regfile[ra2];
    assign rd3 = (we && wa == ra3) ? wd : regfile[ra3];
```

```
endmodule
```

题目二：

设计流水线CPU，模块如下：

CPU 模块：

```
module Mux2(
input [31:0]in0,in1,
input sel,
output reg[31:0]out
    );
always@(*)
begin
    if(sel)
        out=in1;
    else
        out=in0;
end
endmodule

module Add32(
input [31:0]a,b,
output [31:0] result
    );
assign result=a+b;
endmodule

module Control(
input [6:0]ins,
output reg [1:0]RegScr,//wd段写回数据的来源
output reg jal,branch,imm_gen,ALUop,MemWrite,RegWrite
    );

always@(*)
begin
    branch=0;
    jal=0;
    imm_gen=0;
    RegScr=0;
    ALUop=0;
    MemWrite=0;
    RegWrite=0;

    case(ins[6:0])
    7'b1100011: //beq
    begin
        branch=1;
        jal=0;
        imm_gen=1;//是否产生立即数
        RegScr=0;
        ALUop=1;//需要用到ALU
        MemWrite=0;
        RegWrite=0;
    end
```

```
    7'b1101111: //jal
    begin
        branch=1;//
        jal=1;
        imm_gen=1;
        RegScr=2'd2;//source : pc+4
        ALUop=0;//no use
        MemWrite=0;
        RegWrite=1;
    end
    7'b0000011: //lw
    begin
        branch=0;
        jal=0;
        imm_gen=1;
        RegScr=1;
        ALUop=1;
        MemWrite=0;
        RegWrite=1;
    end
    7'b0100011: //sw
    begin
        branch=0;
        jal=0;
        imm_gen=1;
        RegScr=0;
        ALUop=1;
        MemWrite=1;
        RegWrite=0;
    end
    7'b0010011: //addi
    begin
        branch=0;
        jal=0;
        imm_gen=1;
        RegScr=0;
        ALUop=1;
        MemWrite=0;
        RegWrite=1;
    end
    7'b0110011: //add
    begin
        branch=0;
        jal=0;
        imm_gen=0;
        RegScr=0;
        ALUop=1;
        MemWrite=0;
        RegWrite=1;
    end
    default:
    begin
        branch=0;
        jal=0;
        imm_gen=0;
        RegScr=0;
        ALUop=0;
        MemWrite=0;
```

```
        RegWrite=0;
    end
endcase
end
endmodule

module Registers//寄存器堆
(
    input we,
    input clk,
    input [4:0] wa,
    input [31:0] wd,
    input [4:0] ra1,ra2,ra3,
    output [31:0] rd1,rd2,rd3
);
    reg [31:0] regfile[0:31];
    integer i;
    initial begin
        for(i=0;i<32;i=i+1) regfile[i]=0;
    end

    always @(posedge clk)
    begin
        if(we && wa!=0 ) regfile[wa]<=wd;
        else regfile[wa]<=regfile[wa];
    end

    assign rd1 = (we && wa == ra1) ? wd : regfile[ra1];
    assign rd2 = (we && wa == ra2) ? wd : regfile[ra2];
    assign rd3 = (we && wa == ra3) ? wd : regfile[ra3];

endmodule

module Imm_gen_control
(
input imm_gen,//no-use
input [31:0]ins,
output reg [31:0]imm
);
//already shift left
always@(*)
begin
    case(ins[6:0])
        7'b0000011: imm={ {21{ins[31]}}, ins[30:20] };
        7'b0100011: imm={ {21{ins[31]}}, ins[30:25], ins[11:7]};
        7'b0010011: imm={ {21{ins[31]}}, ins[30:20] };
        7'b1100011: imm={ {20{ins[31]}}, ins[7], ins[30:25], ins[11:8],
1'b0};//beq
        7'b1101111: imm={ {12{ins[31]}}, ins[19:12], ins[20], ins[30:21], 1'b0};
        7'b0110011: imm=32'd0;//add
        default:imm=32'd0;
    endcase
end

endmodule

module ALU_control(
input ALUop,
```

```
input [31:0]ins,
output reg[2:0]f
    );
always@(*)
if(ALUop)
begin
    case(ins[6:0])
    7'b1100011: //beq
    begin
        f=1;
    end
    7'b0000011: //lw
    begin
        f=0;
    end
    7'b0100011: //sw
    begin
        f=0;
    end
    7'b0010011: //addi
    begin
        f=0;
    end
    7'b0110011: //add
    begin
        f=0;
    end
    default:
    begin
        f=0;
    end
endcase
end
endmodule

module Forwarding_Unit//前递
(
input [4:0]rs1,rs2,rdm,rdw,
input wb1,wb2,imm_gen,
output reg[1:0]ALUSrc1,ALUSrc2,BSrc,
input [31:0] ire
);
wire [6:0] iii;//op
assign iii = ire[6:0];
always@(*)
begin
    if(rs1==rdm&&wb1)
        ALUSrc1<=1;
    else
        if(rs1==rdw&&wb2)
            ALUSrc1<=2;
        else
            ALUSrc1<=0;
    if(imm_gen&&iii!=7'h63)//////////
        ALUSrc2<=3;
    else
        if(rs2==rdm&&wb1)
            ALUSrc2<=1;
```

```
        else
            if(rs2==rdw&&wb2)
                ALUSrc2<=2;
            else
                ALUSrc2<=0;

    //BSrc 选择信号 for MEM Write--sw 指令
    if(rs2==rdm&&wb1)
        BSrc<=1;
    else
        if(rs2==rdw&&wb2)
            BSrc<=2;
        else
            BSrc<=0;
end
endmodule

module Mux3(
input [31:0]in0,in1,in2,
input [1:0]sel,
output reg[31:0]out
    );
always@(*)
begin
    case(sel)
    0:out=in0;
    1:out=in1;
    2:out=in2;
    default:out=0;
    endcase
end
endmodule

module Mux4(
input [31:0]in0,in1,in2,in3,
input [1:0]sel,
output reg[31:0]out
    );
always@(*)
begin
    case(sel)
    0:out=in0;
    1:out=in1;
    2:out=in2;
    3:out=in3;
    default:out=0;
    endcase
end
endmodule

module ALU#(parameter WIDTH = 32)
(
input [WIDTH-1:0] a, b, //两操作数
input [2:0] f,          //操作功能
output reg [WIDTH-1:0] y,   //运行结果
output z            //是否为0
);
assign z=~|y;
```

```
always@(*)
case (f)
    3'b000: y=a+b;
    3'b001: y=a-b;
    3'b010: y=a&b;
    3'b011: y=a|b;
    3'b100: y=a^b;
    default:
        y=0;
endcase
endmodule

module Hazard_Detection_Unit(
input branch,//== PC_Src (1: beq | jal)
input [6:0]opcode_ex,opcode_id,//for ALU
input [4:0]rs1,rs2,rd,
output reg PC_flush,PC_EN,IF_ID_flush,ID_EX_flush,IF_ID_EN,ID_EX_EN
    );
always@(*)
begin
    if(branch)//stall
        begin
            PC_flush=0;
            PC_EN=1;
            IF_ID_flush=1;
            ID_EX_flush=1;
            IF_ID_EN=1;
            ID_EX_EN=1;
        end
    else
        begin
            if(opcode_ex==7'b0000011)//load-use(beq,add,sw)
                if(((opcode_id==7'b1100011)||(opcode_id==7'b0110011)||
(opcode_id==7'b0100011))&&((rs1==rd)||(rs2==rd)))
                begin
                    PC_flush=0;
                    PC_EN=0;
                    IF_ID_flush=0;
                    ID_EX_flush=1;
                    IF_ID_EN=0;
                    ID_EX_EN=0;
                end
                else//(addi,lw)
                    if(((opcode_id==7'b0000011)||(opcode_id==7'b0010011))&&
(rs1==rd))
                    begin
                        PC_flush=0;
                        PC_EN=0;
                        IF_ID_flush=0;
                        ID_EX_flush=1;
                        IF_ID_EN=0;
                        ID_EX_EN=0;
                    end
                    else
                    begin
                        PC_flush=0;
                        PC_EN=1;
                        IF_ID_flush=0;
```

```
                        ID_EX_flush=0;
                        IF_ID_EN=1;
                        ID_EX_EN=1;
                    end
            else
            begin
                PC_flush=0;
                PC_EN=1;
                IF_ID_flush=0;
                ID_EX_flush=0;
                IF_ID_EN=1;
                ID_EX_EN=1;
            end
        end
end
endmodule
```

调用CPU模块

```
module CPU(
  input clk,
  input rst,

  //IO_BUS
  output [7:0] io_addr,      //led和seg的地址
  output [31:0] io_dout,     //输出led和seg的数据
  output io_we,                 //输出led和seg数据时的使能信号
  input [31:0] io_din,          //来自sw的输入数据

  //Debug_BUS
  input [7:0] m_rf_addr,   //存储器(MEM)或寄存器(RF)的调试读口地址
  output [31:0] rf_data,    //从RF读取的数据
  output [31:0] m_data,    //从MEM读取的数据

  //PC/IF/ID 流水段寄存器
  output reg [31:0] pc,             //PC
  output reg [31:0] pcd,            //ori_pc
  output reg [31:0] ir,             //inst_id
  output reg [31:0] pcin,           //nextpc

  //ID/EX 流水段寄存器
  output reg [31:0] pce,            //pc_ex
  output reg [31:0] a,              //rf_rd data1_ex
  output reg [31:0] b,              //rf_rd data2_ex
  output reg [31:0] imm,            //imm_gen_out_ex
  output reg [4:0] rd,              //imm_wr addr_ex
  output reg [31:0] ctrl,

  //EX/MEM 流水段寄存器
  output reg [31:0] y,              //alu_out_mem
  output reg [31:0] bm,             //reg_b_mem
  output reg [4:0] rdm,             //rf_wr addr_mem
  output reg [31:0] ctrlm,

  //MEM/WB 流水段寄存器
  output reg [31:0] yw,             //alu_out_wr data_wb
  output reg [31:0] mdr,            //mem_wr data_wb
```

```
  output reg [4:0] rdw,             //rf_wr addr_wb
  output reg [31:0] ctrlw
);

wire PC_Src,PC_flush,PC_EN,IF_ID_flush,ID_EX_flush,IF_ID_EN,ID_EX_EN;
reg RegWrite_mem,RegWrite_ex,RegWrite_wb;
wire RegWrite_id;



//IF
wire [31:0]pc_plus4,pcj,pc_next,inst;//+4 or jump

Mux2 Mux_PC(
    .in0(pc_plus4),
    .in1(pcj),
    .out(pc_next),
    .sel(PC_Src)
);
always@(posedge clk,posedge rst)//rst
begin
    if(rst)   pc<=32'h3000;
    else
    begin
    if(PC_flush)    pc<=32'h3000;
    else if(PC_EN)    pc<=pc_next;
    end
end

Add32 add_pc_plus4(
    .a(pc),
    .b(32'd4),
    .result(pc_plus4)
);
Instruction_Memory instruction_memory(
  .a((pc-32'h3000)/4),      // input wire [7 : 0] a   表示pc/4（pc从0开始，不是
3000）
  .spo(inst)  // output wire [31 : 0] spo
);

//IF/ID
always@(posedge clk)
begin
    if(IF_ID_flush)
    begin
        pcin<=32'h3000;
        pcd<=32'h3000;
        ir<=0;
    end
    else
    if(IF_ID_EN)
    begin
        pcin<=pc_next+4;//npc
        pcd<=pc;//original pc
        ir<=inst;//instruction
    end
end
```

```
//ID
wire jal_id,branch_id,imm_gen_id,ALUop_id,MemWrite_id;
wire [1:0]RegSrc_id;
wire [31:0]wd,rd1,rd2,imm_id;


Control control(
.ins(ir[6:0]),      //opcode
.RegScr(RegSrc_id),//判断写入寄存器的数据来源 0：alu_out eg:beq算出新的pc  1：mem_out
eg:lw,r[x]=mem_out   2：pc+4 eg：jal算出pc+4，r[x]=pc+4
.jal(jal_id),       //jal一定跳转
.branch(branch_id),//bran需要判断是否需要跳转
.imm_gen(imm_gen_id),//根据指令类型  指出是否需要产生立即数
.ALUop(ALUop_id),     //是否用到ALU
.MemWrite(MemWrite_id),//写存储器？
.RegWrite(RegWrite_id)//写寄存器？
);


Registers registers(
    .we(RegWrite_wb),
    .clk(clk),
    .wa(rdw),//addr  由wd阶段传回
    .wd(wd),//三选一
    .ra1(ir[19:15]),
    .ra2(ir[24:20]),
    .ra3(m_rf_addr[4:0]),
    .rd1(rd1),
    .rd2(rd2),
    .rd3(rf_data)
);


Imm_gen_control imm_gen_control(
.imm_gen(imm_gen_id),//control  no-use
.ins(ir),
.imm(imm_id)//往后传
);



//ID/EX
reg ALUop_ex,branch_ex,jal_ex,MemWrite_ex,imm_gen_ex;
reg [1:0]RegSrc_ex;
reg [4:0]rs1_ex,rs2_ex;
reg [31:0]ire,npce;

always@(posedge clk)
begin
    //initialize
    ALUop_ex<=0;
    RegWrite_ex<=0;
    branch_ex<=0;
    jal_ex<=0;
    MemWrite_ex<=0;
    imm_gen_ex<=0;
    RegSrc_ex<=0;
    rs1_ex<=0;
```

```
        rs2_ex<=0;
        ire<=0;
        npce<=32'h3000;
        pce<=32'h3000;
        a<=0;
        b<=0;
        imm<=0;
        rd<=0;
        ctrl<=0;

        if(ID_EX_flush)
        begin
            ALUop_ex<=0;
            RegSrc_ex<=0;
            branch_ex<=0;
            jal_ex<=0;
            MemWrite_ex<=0;
            RegWrite_ex<=0;
            imm_gen_ex<=0;
            npce<=32'h3000;
            pce<=32'h3000;
            a<=0;
            b<=0;
            imm<=0;
            rs1_ex<=0;
            rs2_ex<=0;
            rd<=0;
            ire<=0;
            ctrl<=0;
        end
        else
        if(ID_EX_EN)
        begin
            //EX
            ALUop_ex<=ALUop_id;
            branch_ex<=branch_id;
            jal_ex<=jal_id;
            imm_gen_ex<=imm_gen_id;//似乎没用
            npce<=pcin;//plus 4
            pce<=pcd;//original
            a<=rd1;
            b<=rd2;
            imm<=imm_id;
            //for Forwarding Unit用于前递
            rs1_ex<=ir[19:15];
            rs2_ex<=ir[24:20];

            //MEM
            MemWrite_ex<=MemWrite_id;

            //WB
            RegSrc_ex<=RegSrc_id;
            RegWrite_ex<=RegWrite_id;
            rd<=ir[11:7];//写入寄存器的数据的地址

            //EX阶段ir的名字
            ire<=ir;
```

```
        //Ctrl
        ctrl[31:19]<=0;
        ctrl[18]<=RegWrite_id;
        ctrl[17:16]<=RegSrc_id;
        ctrl[15:14]<=0;
        ctrl[13]<=1;
        ctrl[12]<=MemWrite_id;
        ctrl[11:10]<=0;
        ctrl[9]<=jal_id;
        ctrl[8]<=branch_id;
        ctrl[7:1]<=0;
        ctrl[0]<=ALUop_id;
    end
end


//EX
wire [2:0]f;
wire [1:0]ALUSrc1,ALUSrc2,BSrc;
wire [31:0]ALU_a,ALU_b,b_out,ALU_result;
wire zero;
Add32 add_pc_imm(
    .a(pce),
    .b(imm),
    .result(pcj)//pc + offset
);

ALU_control Alu_control(
    .ALUop(ALUop_ex),
    .ins(ire),
    .f(f)//ALU_operator 0:加   1：减 只有beq
);

Forwarding_Unit forwarding_unit(
.rs1(rs1_ex),
.rs2(rs2_ex),
.rdm(rdm),
.rdw(rdw),
.wb1(RegWrite_mem),
.wb2(RegWrite_wb),
.imm_gen(imm_gen_ex),
.ALUSrc1(ALUSrc1),
.ALUSrc2(ALUSrc2),
.BSrc(BSrc),//SEL信号
.ire(ire)
);

//ALU_operand
Mux3 mux_a(
.in0(a),
.in1(y),//刚传给MEM的ALU_Result结果  eg:add x2 x1 x1   add x3 x2 x2 ，第二句x2用前一个alu的值
.in2(wd),//刚传给WD的从MEM中取出的结果 ed：lw x2 0(x0)  add x3 x2 x2,  第二句x2用前一个从mem取出的值
.sel(ALUSrc1),
.out(ALU_a)
);
Mux4 mux_b(
```

```
.in0(b),
.in1(y),
.in2(wd),
.in3(imm),//imm and rs2/////////////
.sel(ALUSrc2),
.out(ALU_b)
);

//for MEM  sw因为sw语句的ALU_b = imm
Mux3 mux_bb(
    .in0(b),//instruction从Register中取出的第二个操作数
    .in1(y),//刚传给MEM的ALU_Result结果
    .in2(wd),//刚传给WD的从MEM中取出的结果
    .sel(BSrc),//BSrc  选择信号
    .out(b_out)//for mem to sw
);

ALU alu(
.a(ALU_a),
.b(ALU_b),
.f(f),//ALU_operator
.y(ALU_result),
.z(zero)
);
assign PC_Src=jal_ex|(branch_ex&zero);//计算出来立马需要用到


//EX/MEM
reg MemWrite_mem;
reg [1:0]RegSrc_mem;
reg [31:0]npcm;         //next_pc_mem
always@(posedge clk)
begin
    //MEM
    bm<=b_out;//for mem to sw
    MemWrite_mem<=MemWrite_ex;
    y<=ALU_result;//mem 阶段获得的ALU_Result   for mem to locate address

    //WB
    RegSrc_mem<=RegSrc_ex;
    npcm<=npce;//for jal wirte back
    RegWrite_mem<=RegWrite_ex;
    rdm<=rd;//for WB to write back address


    //ctrl
    ctrlm[31:19]<=0;
    ctrlm[18]<=RegWrite_ex;
    ctrlm[17]<=0;
    ctrlm[16]<=RegSrc_ex;
    ctrlm[15:14]<=0;
    ctrlm[13]<=1;
    ctrlm[12]<=MemWrite_ex;
    ctrlm[11:10]<=0;
    ctrlm[9]<=jal_ex;
    ctrlm[8]<=branch_ex;
    ctrlm[7:1]<=0;
    ctrlm[0]<=ALUop_ex;
```

```
end

//MEM
wire [31:0]ReadData,MemReadData;
wire MemWrite;
assign io_addr=y[7:0];
assign io_dout=bm;
assign MemWrite=(~y[10])&MemWrite_mem; //若y[10]=1, y/4得到的值开头是400，该地址超出定义的mem大小
assign io_we=MemWrite_mem&y[10];        //若y[10]=1,对应out，io_we=1
Data_Memory Data_memory(
  .a(y[9:2]),        // input wire [7 : 0] a    y是地址，应除以4
  .d(bm),        // input wire [31 : 0] d
  .dpra(m_rf_addr[4:0]),  // input wire [7 : 0] dpra
  .clk(clk),    // input wire clk
  .we(MemWrite),      // input wire we
  .spo(ReadData),    // output wire [31 : 0] spo
  .dpo(m_data)    // output wire [31 : 0] dpo
);
Mux2 mux_mem_io(
    .in0(ReadData),
    .in1(io_din),
    .sel(y[10]),
    .out(MemReadData)    //y[10]=1,对应开头40C in，读出值由io_din决定
);

//MEM/WB
reg [1:0]RegSrc_wb;
reg [31:0]npcw;
always@(posedge clk)
begin
    //WB
    RegSrc_wb<=RegSrc_mem;
    RegWrite_wb<=RegWrite_mem;
    npcw<=npcm;
    yw<=y;
    mdr<=MemReadData;
    rdw<=rdm;

    ctrlw[31:19]<=0;
    ctrlw[18]<=RegWrite_mem;
    ctrlw[17:16]<=RegSrc_mem;
    ctrlw[15:0]<=0;
end

//WB
Mux3 mux_write_data(
    .in0(yw),           //返回值是alu eg: add
    .in1(mdr),          //返回值mem  eg: lw
    .in2(npcw),         //返回值npc  eg: jal
    .sel(RegSrc_wb),
    .out(wd)//for ld to write back
);

//Hazard detection
Hazard_Detection_Unit Hazard_detection_unit(
    .branch(PC_Src),
    .PC_flush(PC_flush),
```

```
    .PC_EN(PC_EN),
    .opcode_ex(ire[6:0]),
    .opcode_id(ir[6:0]),
    .rs1(ir[19:15]),//ir , instruction for ID
    .rs2(ir[24:20]),
    .rd(rd),//for register to wite back
    .IF_ID_flush(IF_ID_flush),
    .ID_EX_flush(ID_EX_flush),
    .IF_ID_EN(IF_ID_EN),
    .ID_EX_EN(ID_EX_EN)
);

endmodule
```

PDU模块

```
module  pdu(
  input clk,
  input rst,

  //选择CPU工作方式;
  input run,
  input step,
  output clk_cpu,

  //输入switch的端口
  input valid,
  input [4:0] in,

  //输出led和seg的端口
  output [1:0] check,  //led6-5:查看类型
  output [4:0] out0,   //led4-0
  output [2:0] an,     //8个数码管
  output [3:0] seg,
  output ready,        //led7

  //IO_BUS
  input [7:0] io_addr,
  input [31:0] io_dout,
  input io_we,
  output [31:0] io_din,

  //Debug_BUS
  output reg[7:0] m_rf_addr,
  input [31:0] rf_data,
  input [31:0] m_data,

  //增加流水线寄存器调试接口
  input [31:0] pcin, pc, pcd, pce,
  input [31:0] ir, imm, mdr,
  input [31:0] a, b, y, bm, yw,
  input [4:0]  rd, rdm, rdw,
  input [31:0] ctrl, ctrlm, ctrlw
);

reg [4:0] in_r, in_2r;    //同步外部输入用，为信号in增加一级寄存器
reg run_r, step_r, step_2r, valid_r, valid_2r;
```

```
wire step_p, valid_pn;     //取边沿信号
wire pre_pn,next_pn;       //增加取边沿信号

reg clk_cpu_r;       //寄存器输出CPU时钟
reg [4:0] out0_r;    //输出外设端口
reg [31:0] out1_r;
reg ready_r;
reg [19:0] cnt;      //刷新计数器，刷新频率约为95Hz
reg [1:0] check_r;   //查看信息类型，00-运行结果，01-寄存器堆，10-存储器，11-plr

reg [7:0] io_din_a; //_a表示为满足组合always描述要求定义的，下同
reg [4:0] out0_a;
reg [31:0] out1_a;
reg [3:0] seg_a;

//增加pre,next取边沿计数器
reg [4:0] cnt_m_rf;      //寄存器堆和存储器地址计数器
reg [1:0] cnt_ah_plr;    //流水线寄存器高两位地址计数器
reg [2:0] cnt_al_plr;    //流水线寄存器低三位地址计数器

//增加流水线寄存器地址和数据选择输入
wire [4:0] addr_plr ;
reg [31:0] plr_data;

assign clk_cpu = clk_cpu_r;
assign io_din = io_din_a;
assign check = check_r;
assign out0 = out0_a;
assign ready = ready_r;
assign seg = seg_a;
assign an = cnt[19:17];
assign step_p = step_r & ~step_2r;      //取上升沿
assign valid_pn = valid_r ^ valid_2r;  //取上升沿或下降沿
assign pre_pn =in_r[1] ^in_2r[1];       //增加pre取上升或下降沿信号
assign next_pn =in_r[0] ^in_2r[0];      //增加next取上升或下降沿信号

//同步输入信号
always @(posedge clk) begin
  run_r <= run;
  step_r <= step;
  step_2r <= step_r;
  valid_r <= valid;
  valid_2r <= valid_r;
  in_r <= in;
  in_2r <= in_r;         //为信号in增加一级寄存器
end

//CPU工作方式
always @(posedge clk, posedge rst) begin
  if(rst)
    clk_cpu_r <= 0;
  else if (run_r)
    clk_cpu_r <= ~clk_cpu_r;
  else
    clk_cpu_r <= step_p;
end

//读外设端口
```

```
always @* begin
  case (io_addr)
    8'h0c: io_din_a = {{27{1'b0}}, in_r};
    8'h10: io_din_a = {{31{1'b0}}, valid_r};
    default: io_din_a = 32'h0000_0000;
  endcase
end

//写外设端口
always @(posedge clk, posedge rst) begin
if (rst) begin
  out0_r <= 5'h1f;
  out1_r <= 32'h1234_5678;
  ready_r <= 1'b1;
end
else if (io_we)
  case (io_addr)
    8'h00: out0_r <= io_dout[4:0];
    8'h04: ready_r <= io_dout[0];
    8'h08: out1_r <= io_dout;
    default: ;
  endcase
end

//增加寄存器堆和存储器地址计数：依靠pre,next边沿计数使能
always @(posedge clk, posedge rst) begin
  if (rst) cnt_m_rf <= 5'b0_0000;
  else if (step_p)
    cnt_m_rf <= 5'b0_0000;
  else if (next_pn)
    cnt_m_rf <= cnt_m_rf + 5'b0_0001;
  else if (pre_pn)
    cnt_m_rf <= cnt_m_rf - 5'b0_0001;
end

//增加流水寄存器地址计数，流水线寄存器高两位地址依靠pre边沿计数，低三位地址依靠next边沿计数
always @(posedge clk, posedge rst) begin
  if (rst) cnt_ah_plr <= 2'b00;
  else if (step_p)
    cnt_ah_plr <= 2'b00;
  else if (pre_pn)
    cnt_ah_plr <= cnt_ah_plr + 2'b01;
end

always @(posedge clk, posedge rst) begin
  if (rst) cnt_al_plr <= 3'b000;
  else if (step_p)
    cnt_al_plr <= 3'b000;
  else if (next_pn)
    if (cnt_ah_plr==2'b01)
        if (cnt_al_plr == 3'b101)
            cnt_al_plr <= 3'b000;
        else cnt_al_plr <= cnt_al_plr + 3'b001;
    else begin
        cnt_al_plr [2] <= 1'b0;
        cnt_al_plr [1:0] <= cnt_al_plr[1:0] + 2'b01;
    end
end
```

```
assign  addr_plr = {cnt_ah_plr,cnt_al_plr};  //增加流水线寄存器地址

//寄存器堆和存储器地址输出选择
//下面的always块也可以用assign m_rf_addr = {in_r[4:2],cnt_m_rf};代替因为寄存器堆只需要低5位就可以了，不关心高3位
always @(*) begin
  case (check_r[1])
    1'b0:
      m_rf_addr = {3'b000,cnt_m_rf};
    1'b1:
      m_rf_addr = {in_r[4:2],cnt_m_rf};
  endcase
end

//流水线寄存器数据选择输入
always @(*)begin
  case (cnt_ah_plr)
  //PC/IF/ID
  2'b00:
      case (cnt_al_plr[1:0])
      2'b00: plr_data = pc;
      2'b01: plr_data = pcd;
      2'b10: plr_data = ir;
      2'b11: plr_data = pcin;
      endcase
   //ID/EX
   2'b01:
   begin
      case (cnt_al_plr)
      3'b000: plr_data = pce;
      3'b001: plr_data = a;
      3'b010: plr_data = b;
      3'b011: plr_data = imm;
      3'b100: plr_data = {{27{1'b0}},rd};
      3'b101: plr_data = ctrl;
      default: plr_data = pce;
      endcase
    end
    //EX/MEM
    2'b10:
      case (cnt_al_plr[1:0])
      2'b00: plr_data = y;
      2'b01: plr_data = bm;
      2'b10: plr_data = {{27{1'b0}},rdm};
      2'b11: plr_data = ctrlm;
      endcase
    //MEM/WB
    2'b11:
      case (cnt_al_plr[1:0])
      2'b00: plr_data = yw;
      2'b01: plr_data = mdr;
      2'b10: plr_data = {{27{1'b0}},rdw};
      2'b11: plr_data = ctrlw;
      endcase
    endcase
end
```

```
//LED和数码管查看类型
always @(posedge clk, posedge rst) begin
if(rst)
    check_r <= 2'b00;
  else if(run_r)
    check_r <= 2'b00;
  else if (step_p)
    check_r <= 2'b00;
  else if (valid_pn)
    check_r <= check - 2'b01;
end

//LED和数码管显示内容
always @(*)begin
  case (check_r)
    2'b00: begin
      out0_a = out0_r;
      out1_a = out1_r;
    end
    2'b01: begin
      out0_a = cnt_m_rf;
      out1_a = rf_data;
    end
    2'b10: begin
      out0_a = cnt_m_rf;
      out1_a = m_data;
    end
    2'b11: begin
      out0_a = addr_plr;
      out1_a = plr_data;    //更改为流水线寄存器地址和数据显示
    end
  endcase
end

//扫描数码管
always @(posedge clk, posedge rst) begin
  if (rst) cnt <= 20'h0_0000;
  else cnt <= cnt + 20'h0_0001;
end

always @* begin
  case (an)
    3'd0: seg_a = out1_a[3:0];
    3'd1: seg_a = out1_a[7:4];
    3'd2: seg_a = out1_a[11:8];
    3'd3: seg_a = out1_a[15:12];
    3'd4: seg_a = out1_a[19:16];
    3'd5: seg_a = out1_a[23:20];
    3'd6: seg_a = out1_a[27:24];
    3'd7: seg_a = out1_a[31:28];
    default: ;
  endcase
end

endmodule
```

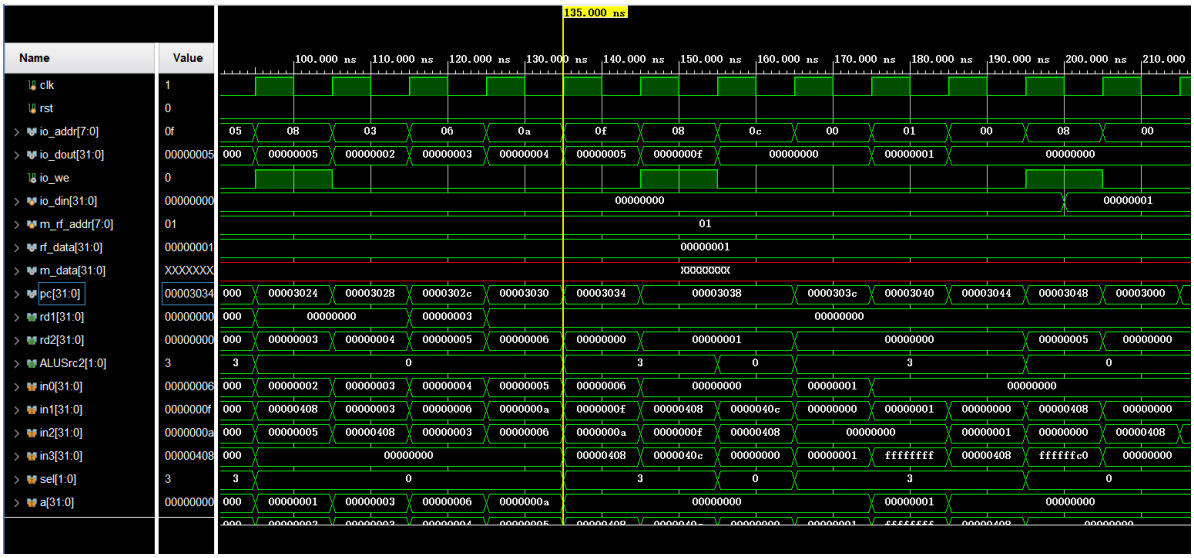
设置一个top文件，调用CPU和PDU

```
module top(
    input clk,
    input rst,
    input [4:0] in,
    input valid,
    input step,
    input run,

    output [1:0] check,
    output [4:0] out0,
    output ready,
    output [2:0] an,
    output [3:0] seg
    );

    wire clk_cpu;

    //IO_BUS
    wire [7:0] io_addr;
    wire [31:0] io_dout;
    wire io_we;
    wire [31:0] io_din;//FIB数列前两项

    //Debug_BUS
    wire [7:0] m_rf_addr;
    wire [31:0] rf_data;
    wire [31:0] m_data;

    //流水段寄存器   CPU  output ,  PDU  input
    wire [31:0] pcin, pc, pcd, pce;
    wire [31:0] ir, imm, mdr;
    wire [31:0] a, b, y, bm, yw;
    wire [4:0]  rd, rdm, rdw;
    wire [31:0] ctrl, ctrlm, ctrlw;


   pdu pdu(
  .clk(clk),
  .rst(rst),

  //选择CPU工作方式;
  .run(run),
  .step(step),
  .clk_cpu(clk_cpu),

  //输入switch的端口
  .valid(valid),
  .in(in),

  //输出led和seg的端口
  .check(check),  //led6-5:查看类型
  .out0(out0),   //led4-0
  .an(an),     //8个数码管
  .seg(seg),
  .ready(ready),        //led7

  //IO_BUS
```

```
  .io_addr(io_addr),
  .io_dout(io_dout),
  .io_we(io_we),
  .io_din(io_din),

  //Debug_BUS
  .m_rf_addr(m_rf_addr),
  .rf_data(rf_data),
  .m_data(m_data),

  //增加流水线寄存器调试接口
  .pcin(pcin),
  .pc(pc),
  .pcd(pcd),
  .pce(pce),
  .ir(ir),
  .imm(imm),
  .mdr(mdr),
  .a(a),
  .b(b),
  .y(y),
  .bm(bm),
  .yw(yw),
  .rd(rd),
  .rdm(rdm),
  .rdw(rdw),
  .ctrl(ctrl),
  .ctrlm(ctrlm),
  .ctrlw(ctrlw)
);


   CPU cpu(
  .clk(clk_cpu),
  .rst(rst),

  //IO_BUS
  .io_addr(io_addr),
  .io_dout(io_dout),
  .io_we(io_we),
  .io_din(io_din),

  //Debug_BUS
  .m_rf_addr(m_rf_addr),
  .rf_data(rf_data),
  .m_data(m_data),

  //PC/IF/ID
  .pc(pc),              //PC
  .pcd(pcd),
  .ir(ir),
  .pcin(pcin),

  //ID/EX
  .pce(pce),
  .a(a),
  .b(b),
  .imm(imm),
```

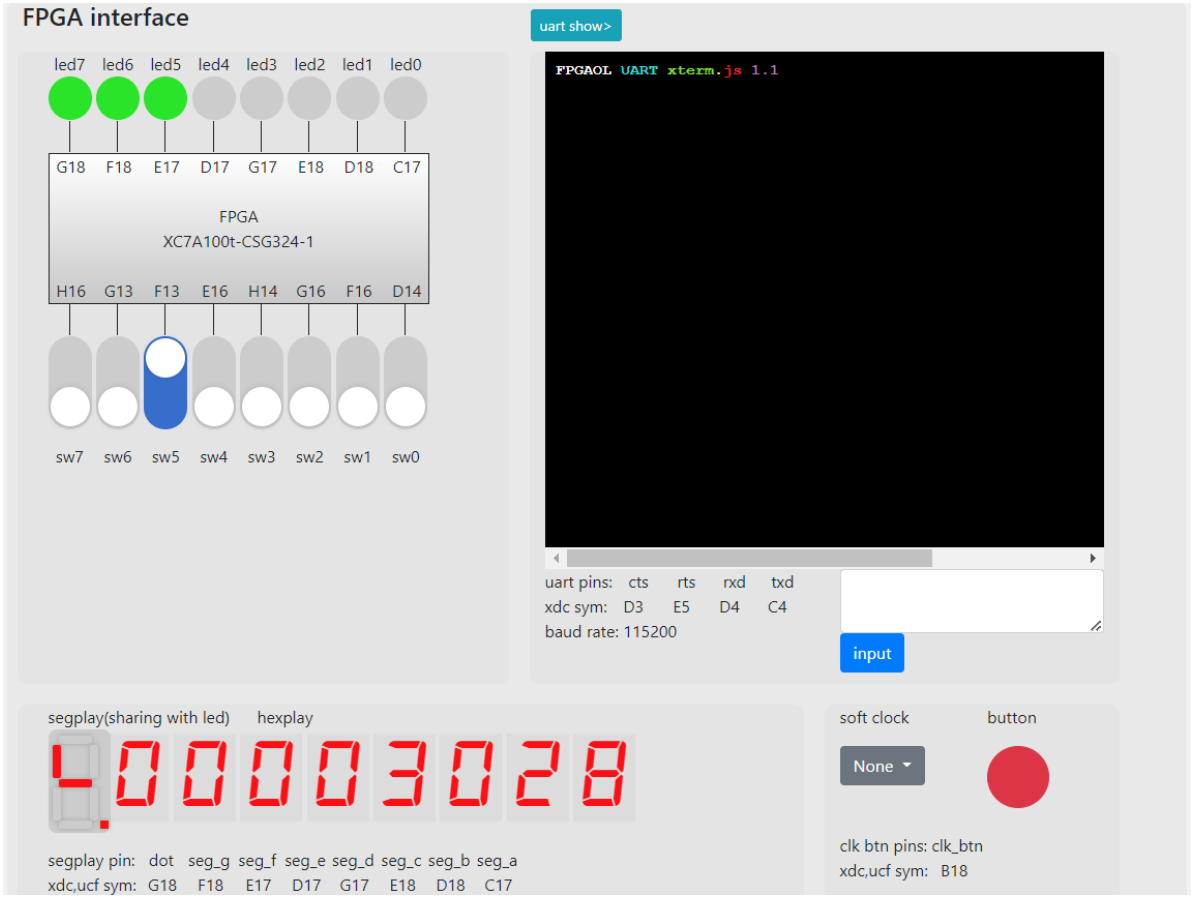
```
    .rd(rd),
    .ctrl(ctrl),

    //EX/MEM
    .y(y),
    .bm(bm),
    .rdm(rdm),
    .ctrlm(ctrlm),

    //MEM/WB
    .yw(yw),
    .mdr(mdr),
    .rdw(rdw),
    .ctrlw(ctrlw)
);

endmodule
```

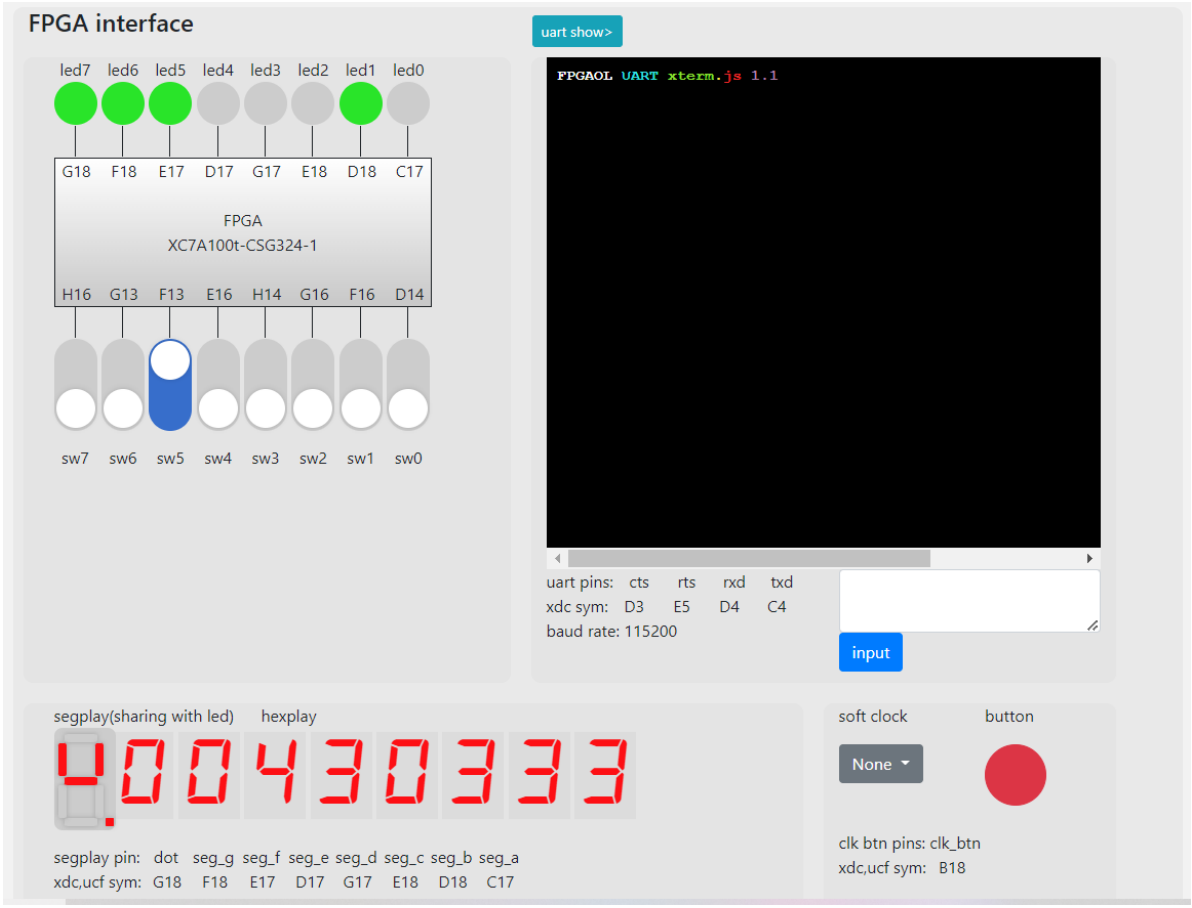
仿真文件验证指令如图所示：



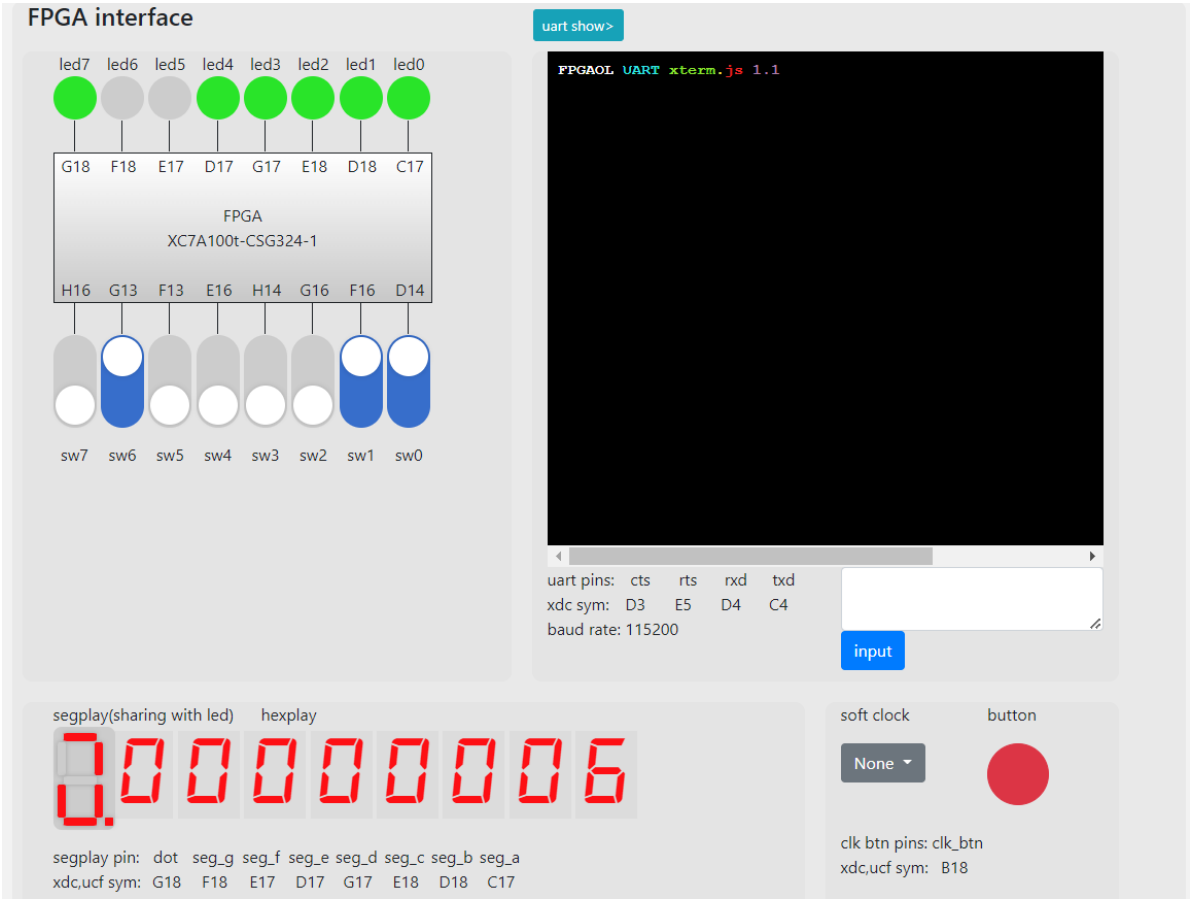
验证正确

进行FPGAOL的实验，首先是hazard_test：


FPGA interface
uart show>
led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
FPGAOL UART xterm.js 1.1
uart pins: cts rts rxd txd
xdc sym: D3 E5 D4 C4
baud rate: 115200
input
segplay(sharing with led)
hexplay
segplay pin: dot seg_g seg_f seg_e seg_d seg_c seg_b seg_a
xdc,ucf sym: G18 F18 E17 D17 G17 E18 D18 C17
soft clock
button
None
clk btn pins: clk_btn
xdc,ucf sym: B18


FPGA interface
uart show>
led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
FPGAOL UART xterm.js 1.1
uart pins: cts rts rxd txd
xdc sym: D3 E5 D4 C4
baud rate: 115200
input
segplay(sharing with led)
hexplay
segplay pin: dot seg_g seg_f seg_e seg_d seg_c seg_b seg_a
xdc,ucf sym: G18 F18 E17 D17 G17 E18 D18 C17
soft clock
button
None
clk btn pins: clk_btn
xdc,ucf sym: B18

其次进行fib：



验证正确

实验总结

难度稍大