lab4 单周期CPU设计

姓名：宋玮　学号：PB20151793　实验日期：2022.4.21

实验题目

lab4 单周期CPU设计

实验目的

> •理解CPU的结构和工作原理

> •掌握单周期CPU的设计和调试方法

> •熟练掌握数据通路和控制器的设计和描述方法

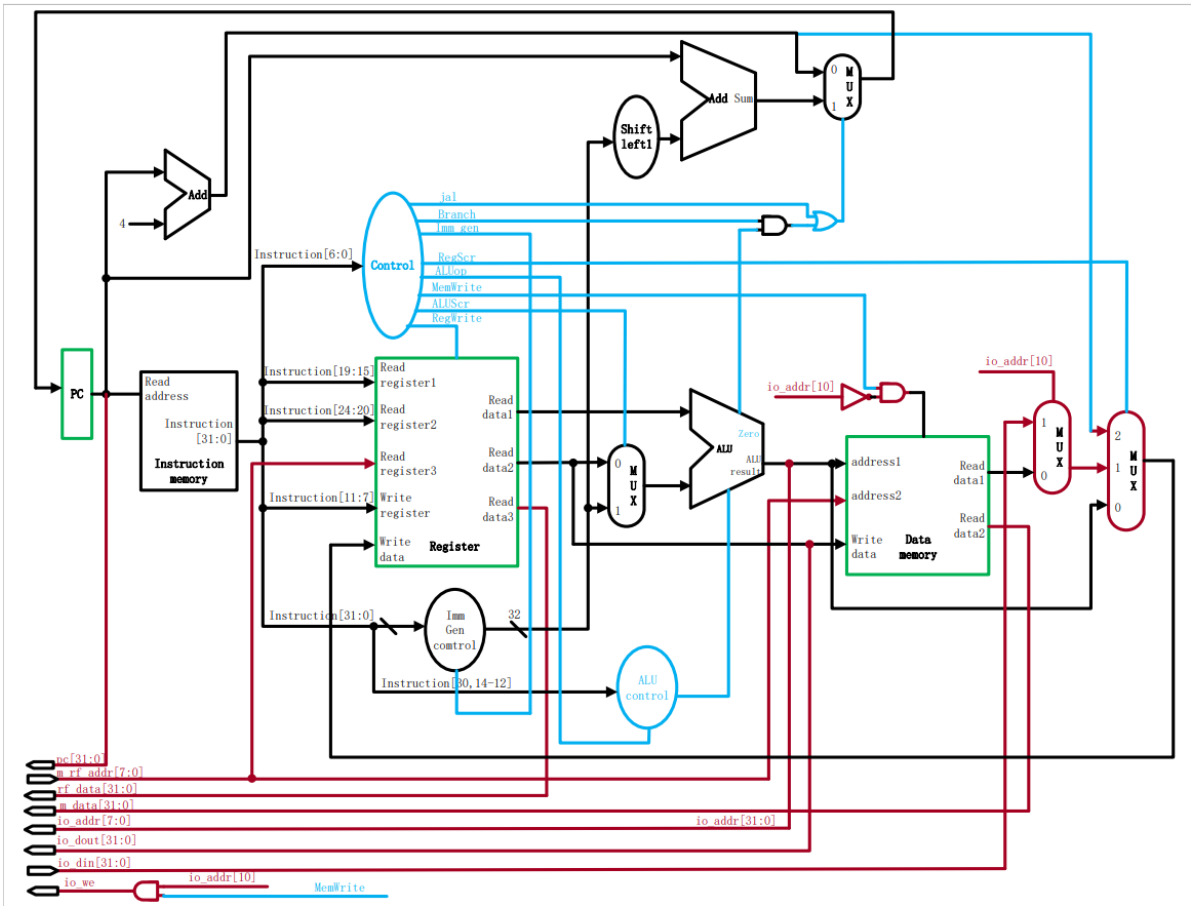
实验平台

Rars，fpgaol，vivado

实验过程

1.设计实现单周期RISC-V CPU

（1）基本数据通路如下



当然，该数据通路适用于6条指令，若想用于10条指令，则需要：

①在Read_data1与pc间加一个mux用于完成auipc指令

②在Branch，jal信号选择器前添加jalr_PC信号，用于完成jalr指令

③sub指令通路与add指令基本一致

④blt指令通路与beq指令基本一致，只需通过ALUop来区分两个指令

（2）CPU各模块

①rf（寄存器堆）

```
module rf#(
    parameter m=5,WIDTH=32
)(
    input clk,we,rst,
    input [m-1:0] wa,
    input [m-1:0]ra0,ra1,
    input [WIDTH-1:0] wd,
    output [WIDTH-1:0] rd0,rd1,
    input [m-1:0] rf_addr,
    output [WIDTH-1:0] rf_data
);

parameter sum = 8'b1 << m;
reg [WIDTH-1:0] regfile [0:sum-1];

assign rd0 = regfile[ra0],   rd1 = regfile[ra1];
assign rf_data = regfile[rf_addr];

always @(posedge clk or posedge rst)
begin
    if(rst)
        begin
            regfile[0] <= 0;
            regfile[1] <= 0;
            regfile[2] <= 0;
            regfile[3] <= 0;
            regfile[4] <= 0;
            regfile[5] <= 0;
            regfile[6] <= 0;
            regfile[7] <= 0;
            regfile[8] <= 0;
            regfile[9] <= 0;
            regfile[10] <= 0;
            regfile[11] <= 0;
            regfile[12] <= 0;
            regfile[13] <= 0;
            regfile[14] <= 0;
            regfile[15] <= 0;
            regfile[16] <= 0;
            regfile[17] <= 0;
            regfile[18] <= 0;
            regfile[19] <= 0;
            regfile[20] <= 0;
            regfile[21] <= 0;
            regfile[22] <= 0;
```

```
            regfile[23] <= 0;
            regfile[24] <= 0;
            regfile[25] <= 0;
            regfile[26] <= 0;
            regfile[27] <= 0;
            regfile[28] <= 0;
            regfile[29] <= 0;
            regfile[30] <= 0;
            regfile[31] <= 0;
        end
    else if (we && wa!= 0)  regfile[wa]  <=  wd;
end
endmodule
```

②immg（立即数处理模块）

```
module immg(
input [31:0] ins,
output reg [31:0] imm
    );

always @(*)
begin
    case(ins[6:0])
        7'b0000011:
            begin
                imm = {{20{ins[31]}},ins[31:20]};//lw
            end
        7'b0100011:
            begin
                imm = {{20{ins[31]}},ins[31:25],ins[11:7]}; //sd
            end
        7'b0010011:  imm = {{20{ins[31]}},ins[31:20]};//addi
        7'b1100011:  imm =
{{20{ins[31]}},ins[7],ins[30:25],ins[11:8],1'b0};//beq,blt
        7'b1101111:  imm =
{{12{ins[31]}},ins[19:12],ins[20],ins[30:21],1'b0};//jal
        7'b0010111:  imm = {ins[31:12],12'b0};//aauipc
        7'b1100111:  imm = {{20{ins[31]}},ins[31:20]};//jalr
        default:    imm = 0; //add,sub
    endcase
end

endmodule
```

③alu模块

```
module alu#( parameter WIDTH = 32
)(
    input[WIDTH-1:0] a,
    input[WIDTH-1:0] b,
    input [31:0] ins,
    input [2:0] f,
```

```
    output reg [WIDTH-1:0] y,
    output z
);

wire signed [31:0] sa;
wire signed [31:0] sb;
assign sa = a;
assign sb = b;

always@(*)
begin
    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;
        3'b101: y = (sa>=sb);
        default: y = 0;
    endcase
end

assign z = y ? 1'b0 : 1'b1;

endmodule
```

④alu_control模块

```
module alu_control(
input [1:0] aluop,
output reg [2:0] sel);

always@(*)
begin
    case(aluop)
        2'b01:   sel = 3'b001;
        2'b11:   sel = 3'b101;
        default: sel = 3'b000;
    endcase
end
endmodule
```

⑤control模块

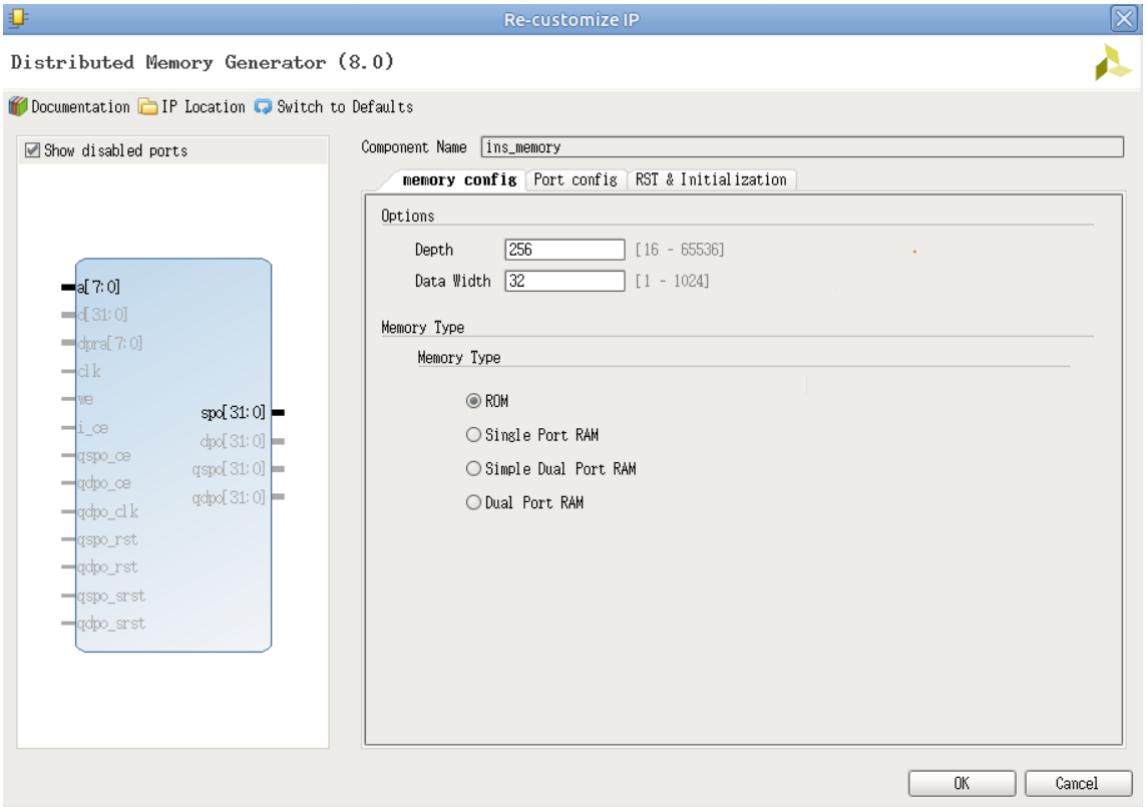
```
module control(
input [31:0] ins,
output reg ALUSrc,RegWrite,MemRead,MemWrite,Branch,JUMP,jalr_PC,auipc,
output reg [1:0] ALUOp, reg [1:0] MemtoReg
    );

always@(*)
begin
    case(ins[6:0])
```

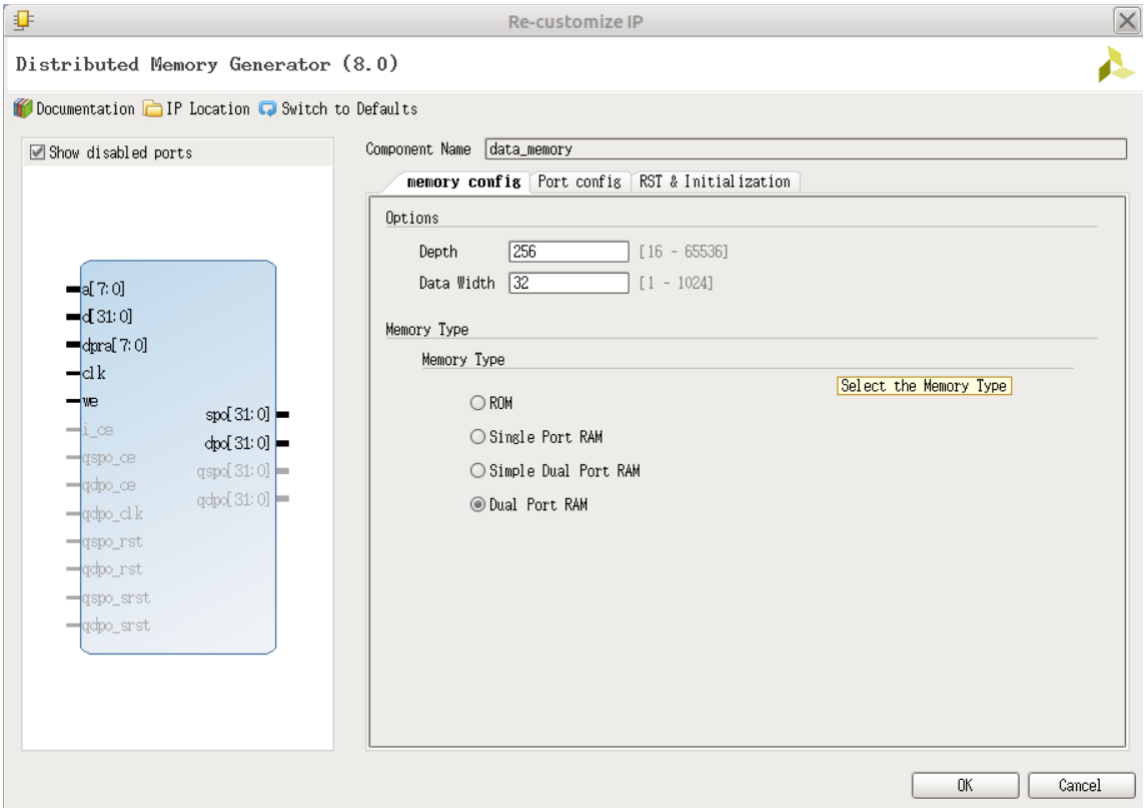
```
        7'b0110011:
            begin
                if(ins[31:25]==7'b0)   //add
                    begin
                    ALUSrc = 0; MemtoReg = 2'b00; RegWrite = 1; MemRead = 0; MemWrite = 0;
                    Branch = 0; JUMP = 0; ALUOp = 2'b10; auipc = 0; jalr_PC = 0;
                    end
                else                   //sub
                    begin
                    ALUSrc = 0; MemtoReg = 2'b00; RegWrite = 1; MemRead = 0; MemWrite = 0;
                    Branch = 0; JUMP = 0; ALUOp = 2'b01; auipc = 0; jalr_PC = 0;
                    end
            end
        7'b0000011:    //lw
            begin
                ALUSrc = 1; MemtoReg = 2'b01; RegWrite = 1; MemRead = 1; MemWrite = 0;
                Branch = 0; JUMP = 0; ALUOp = 2'b00; auipc = 0; jalr_PC = 0;
            end
        7'b0100011:    //sw
            begin
                ALUSrc = 1; MemtoReg = 2'b00; RegWrite = 0; MemRead = 0; MemWrite = 1;
                Branch = 0; JUMP = 0; ALUOp = 2'b00; auipc = 0; jalr_PC = 0;
            end
        7'b0010011:     //addi
            begin
                ALUSrc = 1; MemtoReg = 2'b00; RegWrite = 1; MemRead = 0; MemWrite = 0;
                Branch = 0; JUMP = 0; ALUOp = 2'b10; auipc = 0; jalr_PC = 0;
            end
        7'b1100011:     // beq,blt
            begin
                if(ins[14:12]==3'b000)   //beq
                    begin
                        ALUSrc = 0; MemtoReg = 0; RegWrite = 0; MemRead =0 ; MemWrite = 0;
                        Branch = 1; JUMP = 0; ALUOp = 2'b01; auipc = 0; jalr_PC = 0;
                    end
                 else                   //blt
                     begin
                        ALUSrc = 0; MemtoReg = 0; RegWrite = 0; MemRead =0 ; MemWrite = 0;
                        Branch = 1; JUMP = 0; ALUOp = 2'b11; auipc = 0; jalr_PC = 0;
                     end
            end
        7'b1101111:     //jal
            begin
                ALUSrc = 0; MemtoReg = 2'b10; RegWrite = 1; MemRead =0 ; MemWrite = 0;
                Branch = 0; JUMP = 1; ALUOp = 2'b00; auipc = 0; jalr_PC = 0;
            end
```

```
        7'b1100111://jalr
            begin
                ALUSrc = 1; MemtoReg = 2'b10; RegWrite = 1; MemRead =0 ; MemWrite = 0;
                Branch = 0; JUMP = 0; ALUOp = 2'b00; auipc = 0;  jalr_PC = 1;
            end
        7'b0010111: //auipc
            begin
                ALUSrc = 1; MemtoReg = 2'b00; RegWrite = 1; MemRead =0 ; MemWrite = 0;
                Branch = 0; JUMP = 0; ALUOp = 2'b10; auipc = 1; jalr_PC = 0;
            end
        default:
            begin
                ALUSrc = 0; MemtoReg = 2'b00; RegWrite = 0; MemRead =0 ; MemWrite = 0;
                Branch = 0; JUMP = 0; ALUOp = 2'b00; auipc = 0; jalr_PC = 0;
            end
    endcase
end
endmodule
```

⑥data_memory & ins_memory

分别使用两个分布式存储器ip核，并用对应的data.coe和ins.coe文件对其进行初始化；





⑦CPU主体模块

```
module cpu(
input clk,rst,
input [31:0] io_din,
input [7:0] m_rf_addr ,
output [7:0] io_addr ,
output [31:0] io_dout ,
output io_we ,
output [31:0] rf_data,
output [31:0] m_data ,
output reg [31:0] pc
);

wire Zero;
wire Branch,MemRead,MemWrite,ALUSrc,RegWrite,JUMP,jalr_PC,auipc;
wire MemWrite_true;
wire [1:0] ALUOp ,MemtoReg;
wire [31:0] ins, imm;
wire [31:0] readData1,readData2,ALU_input2,ALU_input1;
wire [31:0] Mem_ReadData, RF_writeData;
wire [31:0] Mem_ReadData_waishe, Mem_ReadData_MEM;
wire [31:0] nPC_4,PC_offset,PC_jalr,PC_mux;
wire  [31:0] nPC;
wire which_PC;
wire [31:0] ALU_result;
reg [31:0] writeData;
wire [2:0] sel;
wire [31:0] io_addr1;
wire [7:0] apc;

ins_memory IMem(.a(apc),.spo(ins));
```

```
    data_memory
DMem(.a(ALU_result[9:2]),.d(readData2),.dpra(m_rf_addr),.clk(clk),.we(MemWrite_true),.spo(Mem_ReadData_MEM),.dpo(m_data));

control control(.ins(ins),
.ALUSrc(ALUSrc),.RegWrite(RegWrite),.MemRead(MemRead),
.MemWrite(MemWrite),.Branch(Branch),.JUMP(JUMP),.jalr_PC(jalr_PC),.auipc(auipc),
.ALUOp(ALUOp), .MemtoReg(MemtoReg)  );

rf #(.m(5),.WIDTH(32)) rf (.clk(clk),.we(RegWrite),
    .wa(ins[11:7]),.ra0(ins[19:15]),.ra1(ins[24:20]),
    .wd(writeData),.rd0(readData1),.rd1(readData2),
    .rf_addr(m_rf_addr[4:0]),
    .rf_data(rf_data),.rst(rst));

immg immg(.ins(ins),.imm(imm));

alu_control alu_control(.aluop(ALUOp),.sel(sel));
alu # (32)
alu(.a(ALU_input1),.b(ALU_input2),.ins(ins),.f(sel),.z(Zero),.y(ALU_result));

assign PC_jalr = ALU_result & (~32'b1);  //jalr PC
assign nPC_4 = pc + 4;     //pc+4
assign PC_offset = pc + imm;
assign which_PC = (Branch & Zero) | JUMP ;
assign PC_mux = which_PC ? PC_offset : nPC_4;
assign nPC = jalr_PC ? PC_jalr : PC_mux;
assign ALU_input2 = ALUSrc ? imm : readData2;
assign ALU_input1 = auipc ? pc : readData1;
assign apc = pc[9:2];

always@(posedge clk or posedge rst)
begin
    if(rst) pc <= 32'h0000_3000;
    else pc <= nPC;
end

//register file writedata
always@(*)
begin
    if((JUMP==0)||(jalr_PC==0))
        begin
            if(MemtoReg==0) writeData = ALU_result;
            else writeData = Mem_ReadData;
        end
    else  writeData = nPC_4 ;
end

assign io_addr1=ALU_result;
assign io_addr = io_addr1[7:0];
assign MemWrite_true = (~io_addr1[10]) & MemWrite;
assign Mem_ReadData_waishe = io_din;
assign Mem_ReadData = io_addr1[10]? Mem_ReadData_waishe : Mem_ReadData_MEM;
assign io_dout = readData2;
assign io_we = io_addr1[10] & MemWrite;

```

```
endmodule
```

2.pdu模块（用于io）

```
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 [7:0] m_rf_addr,
  input [31:0] rf_data,
  input [31:0] m_data,
  input [31:0] pc
);

reg [4:0] in_r;    //同步外部输入用
reg run_r, step_r, step_2r, valid_r, valid_2r;
wire step_p, valid_pn;  //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-PC

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 ready_a;
reg [4:0] out0_a;
reg [31:0] out1_a;
reg [3:0] seg_a;

assign clk_cpu = clk_cpu_r;
```

```
assign io_din = io_din_a;
assign check = check_r;
assign out0 = out0_a;
assign ready = ready_a;
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 m_rf_addr = {{3{1'b0}}, in_r};

//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;
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

//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
  ready_a = 1'b0;
  case (check_r)
    2'b00: begin
      out0_a = out0_r;
      out1_a = out1_r;
      ready_a = ready_r;
    end
    2'b01: begin
      out0_a = in_r;
      out1_a = rf_data;
    end
    2'b10: begin
      out0_a = in_r;
      out1_a = m_data;
    end
    2'b11: begin
      out0_a = 5'b00000;
      out1_a = pc;
    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
```

（1）外设使用说明：

外设使用说明表

sw				button	led			an/seg	说明
7	6	5	4~0		7	6~5	4~0		
rst	run	valid	in	step	ready	check	out0	out1	
↑	-	-	-	-	1	00	0x1f	0x12···8	复位
x	1	valid	in	-	ready	00	out0	out1	连续运行
	0	valid	in	↑	ready	00	out0	out1	单步运行
		↑↓	addr_rf	x	0	01	addr_rf	data_rf	查看寄存器堆
			addr_m		0	10	addr_m	data_m	查看存储器
			-		0	11	0	PC	查看PC

（2）存储器配置：

实验存储器配置

- 指令存储器(256x32bit)地址：0x0000_3000 ~ 0x0000_33ff
- 数据存储器(256x32bit)地址：0x0000_0000 ~ 0x0000_03ff
- 外设端口地址：0x0000_0400 ~ 0x0000_07ff

存储器映射外设端口地址表

存储器地址	I/O_addr	输出端口名	输入端口名	外设
0x0000_0400	0	out0	-	led4-0
0x0000_0404	1	ready	-	led7
0x0000_0408	2	out1	-	an, seg
0x0000_040C	3	-	in	sw4-0
0x0000_0410	4	-	valid	sw5

3.10条指令测试汇编程序以及仿真

（1）汇编程序

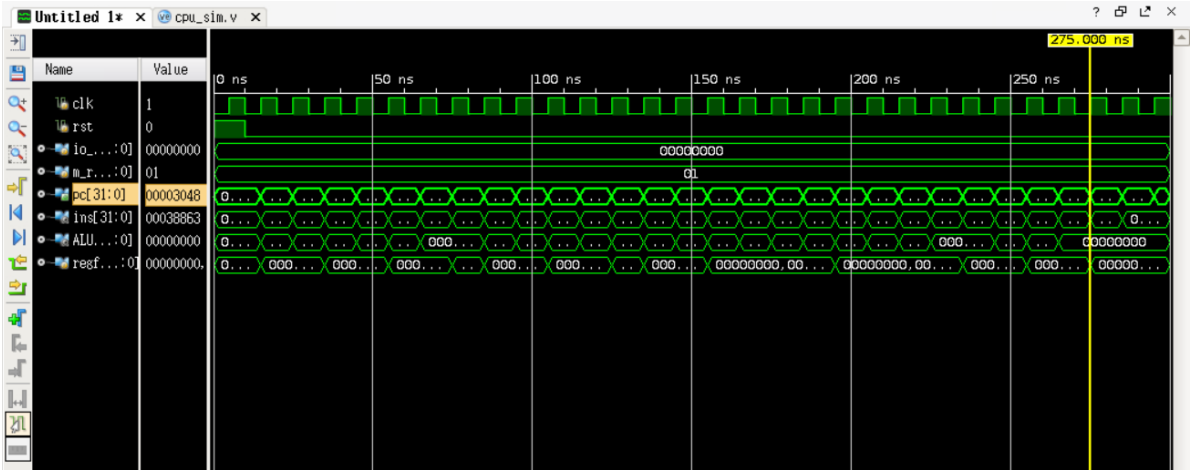
```
.data
0x400
0xfe

.text
lw t3, 0(x0)
sw x0, 8(t3)             #test sw
addi t0, x0, 0xff        #test addi
```

```
sw t0, 8(t3)
lw t0, 4(x0)             #test lw
sw t0, 8(t3)
addi t1,x0,1
add t0,t1,t0            #test add
sw t0, 8(t3)            #t0=t0+t1
sub t0,t0,t1            #t0=t0-t1, test sub
sw t0, 8(t3)
addi t2,x0,1
auipc t1,0              #test auipc
loop1:
blt t2,x0,be            #test blt
addi t2,t2,-1
sw t2, 8(t3)
jalr x1,t1,4               #test jalr
be: addi t2,x0,1
loop:
beq t2,x0,exit             #test beq
addi t2,t2,-1
sw t2, 8(t3)
jal x1,loop                #test jal
exit:
```

(2) 仿真文件

```
module cpu_sim(
);
reg clk;
reg rst;
reg [31:0] io_din;
reg [7:0] m_rf_addr;

initial clk = 0;
always #5 clk = ~clk;
initial rst = 1;
initial #10 rst = 0;
initial m_rf_addr = 1;
initial io_din = 0;
initial #300 $stop;

cpu cpu1(.clk(clk),.rst(rst),.m_rf_addr(m_rf_addr),.io_din(io_din));

endmodule
```

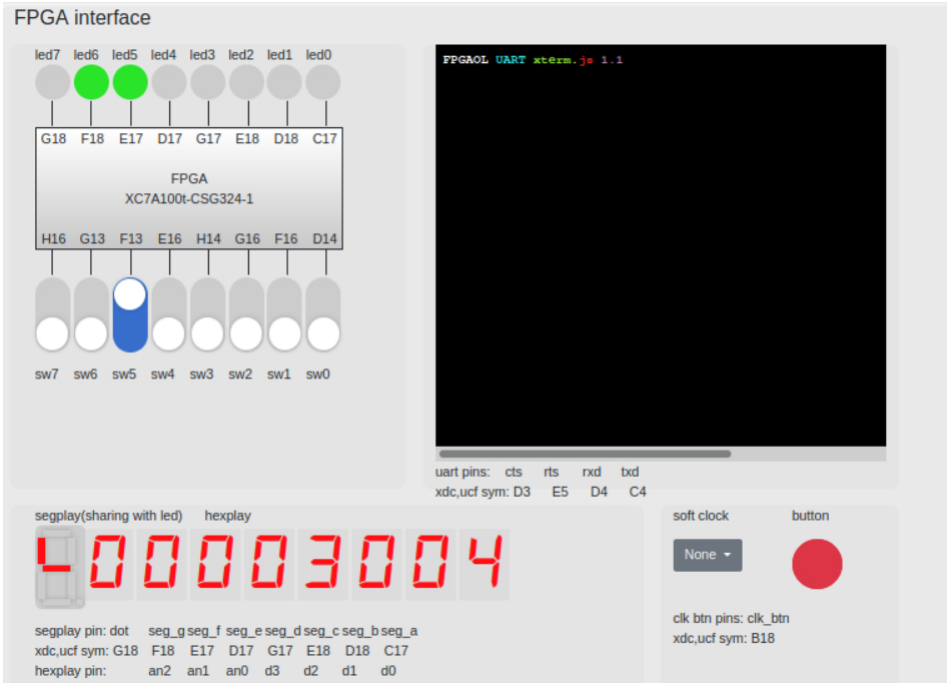
(3) 仿真截图

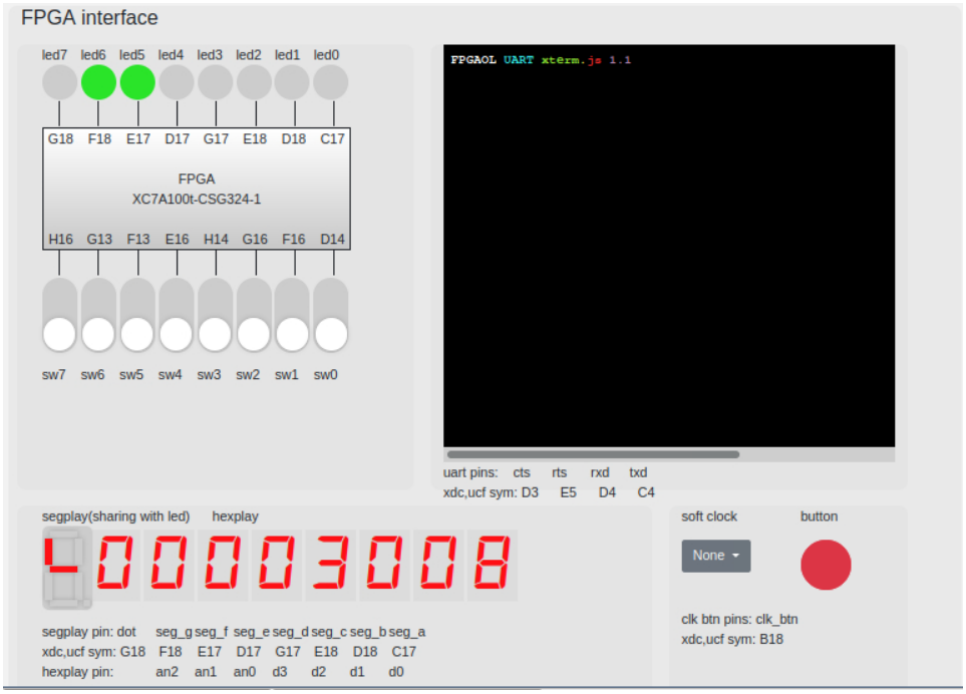


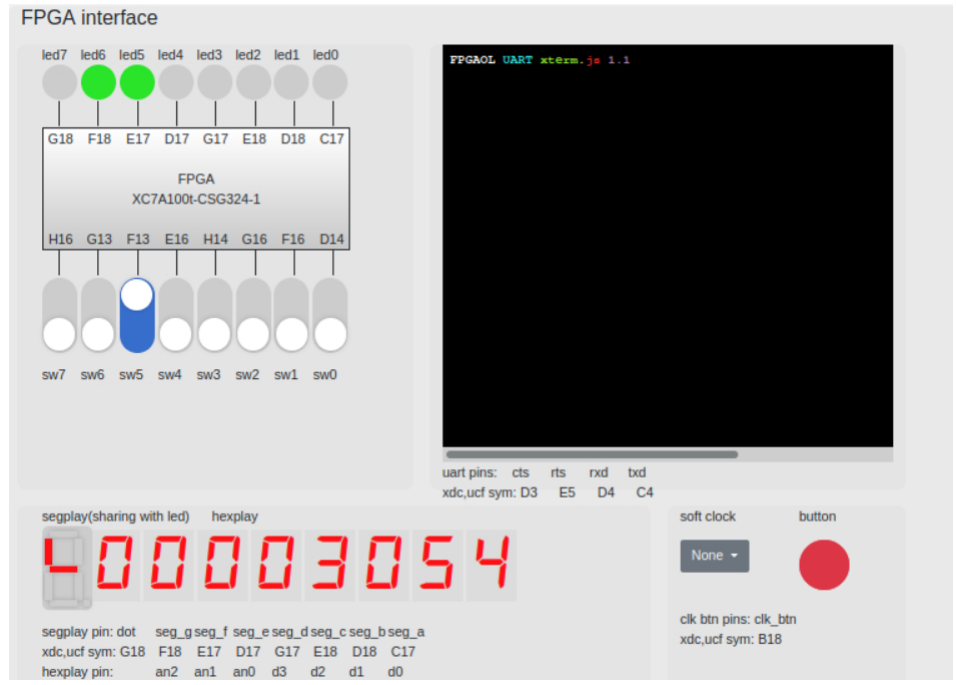
从图中可以看到pc，ins，regfile等值的变化，可验证该执行流程是正确的。

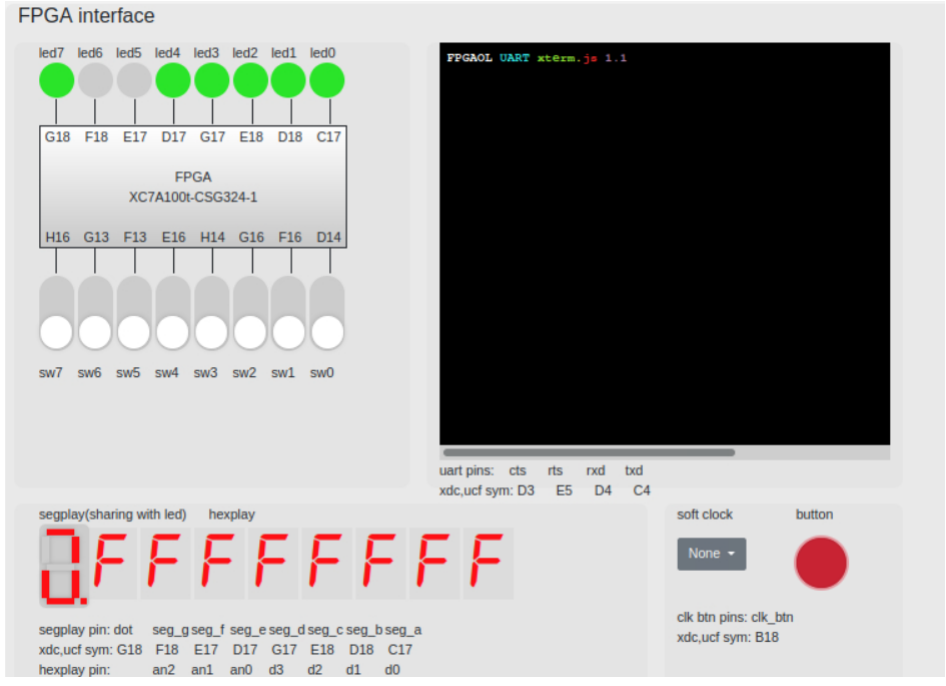
4.10条指令上板测试（fpga平台）

图1,2,3中显示均为pc值，图4显示为某一步的read_data2值




FPGA interface
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,ucf sym: D3 E5 D4 C4
segplay(sharing with led) hexplay
L 0 0 0 0 3 0 0 8
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
hexplay pin: an2 an1 an0 d3 d2 d1 d0
soft clock button
None
clk btn pins: clk_btn
xdc,ucf sym: B18


FPGA interface
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,ucf sym: D3 E5 D4 C4
segplay(sharing with led) hexplay
L 0 0 0 0 3 0 5 4
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
hexplay pin: an2 an1 an0 d3 d2 d1 d0
soft clock button
None
clk btn pins: clk_btn
xdc,ucf sym: B18


FPGA interface
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,ucf sym: D3 E5 D4 C4
segplay(sharing with led) hexplay
F F F F F F F F
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
hexplay pin: an2 an1 an0 d3 d2 d1 d0
soft clock button
None
clk btn pins: clk_btn
xdc,ucf sym: B18

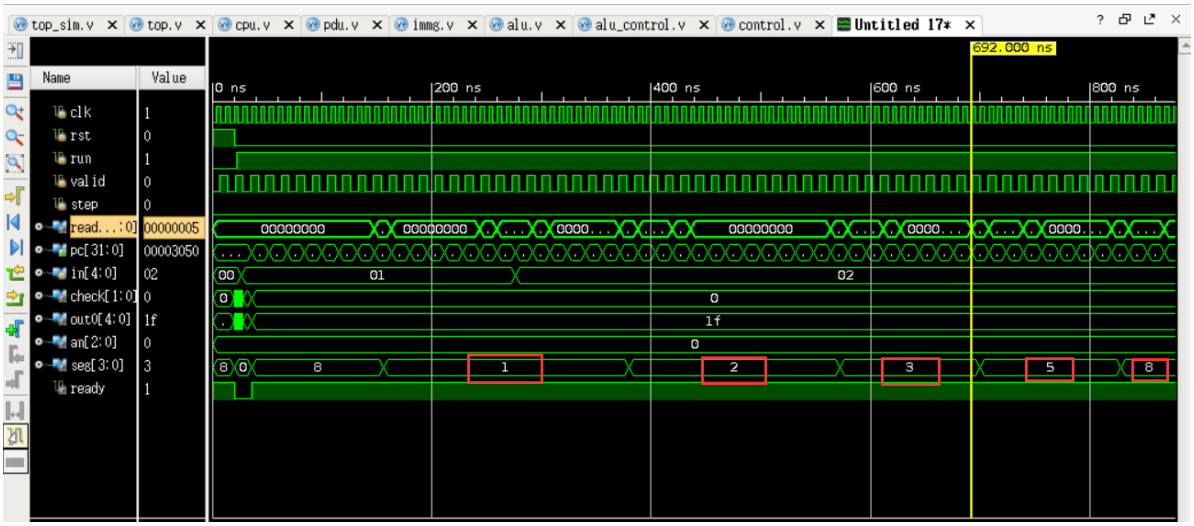
5.fib仿真

（1）仿真文件

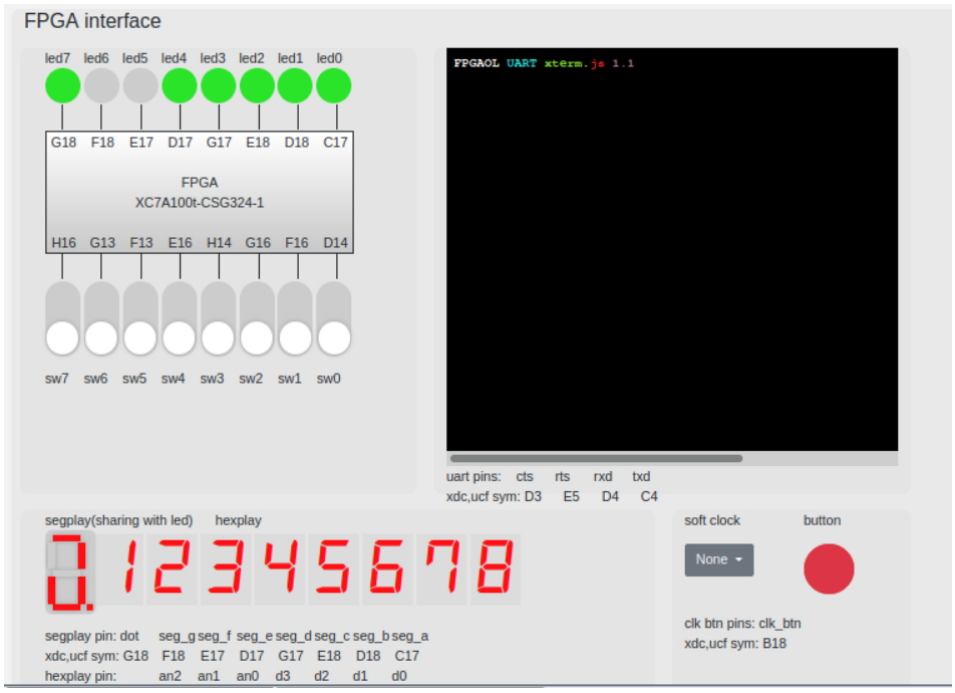
```
module top_sim(
    );

reg clk,rst,run,valid,step;
reg [4:0] in;
wire [1:0]check;
wire [4:0] out0;
wire [2:0] an;
wire [3:0] seg;
wire ready;

top top1(.clk(clk),.rst(rst),.run(run),.valid(valid),.step(step),.in(in),
.check(check),.out0(out0),.an(an),.seg(seg),.ready(ready));

initial clk = 0;
always #4 clk = ~clk;
initial valid = 0;
always #7 valid = ~valid;
initial
begin
        rst = 1;run = 0; in = 0; step = 0;
        #20 rst = 0;
        #2 run=1;
        #5 in = 1;
        #250 in = 2;
        #600 $finish;
end

endmodule
```

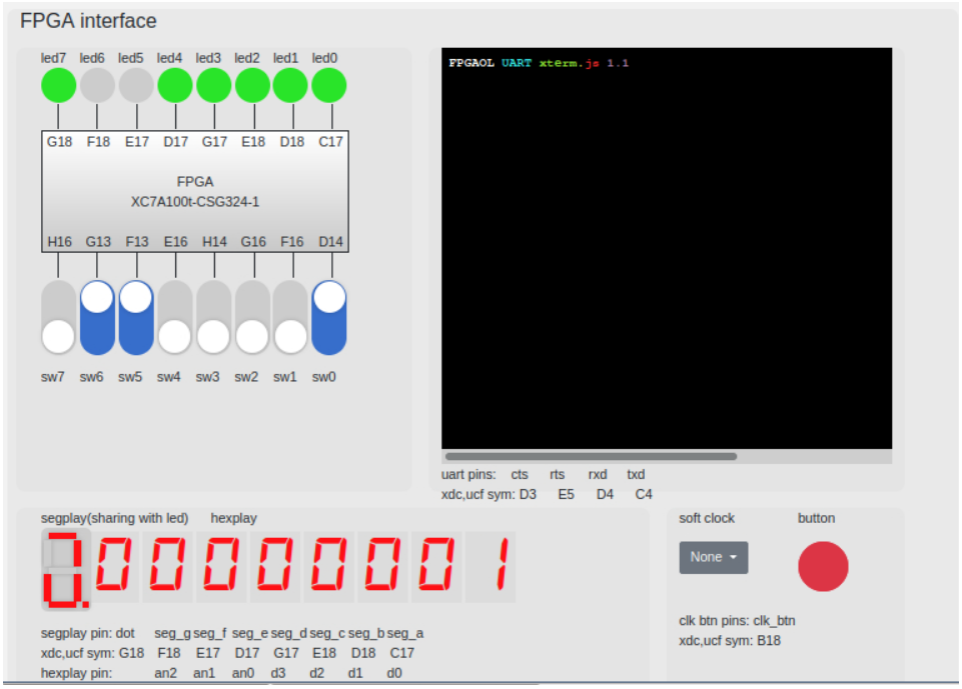
（2）仿真截图

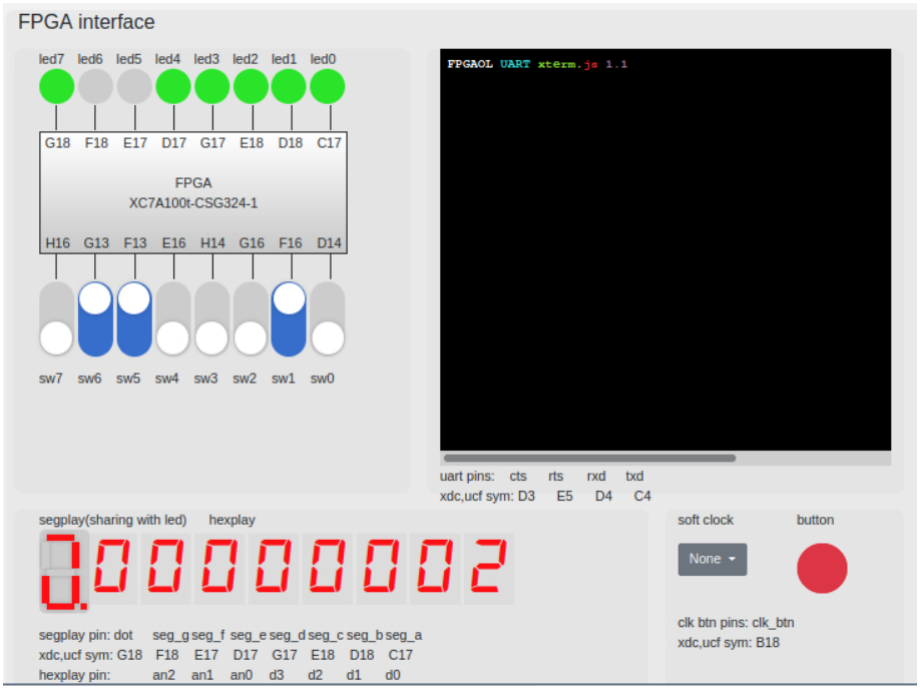
输入前两项为1和2，后面自动输出3，5，8 .....

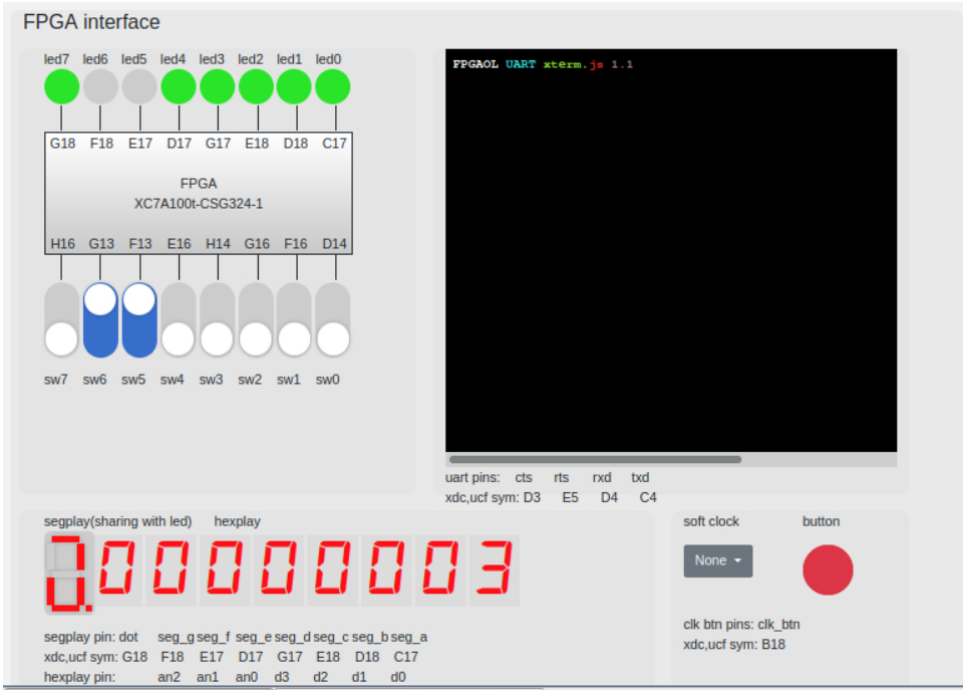


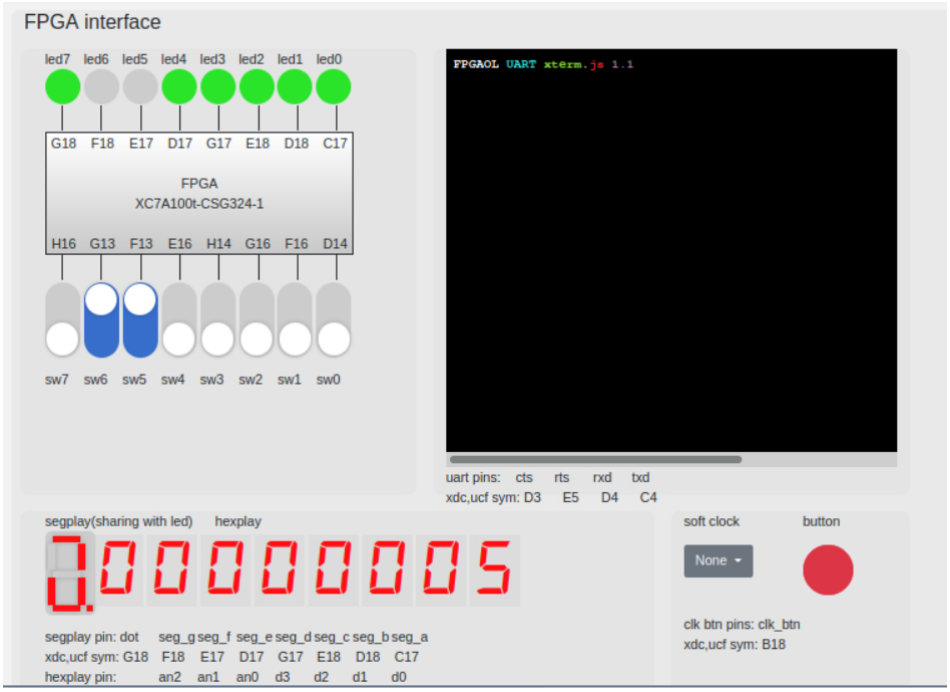
6.fib上板（fpga平台测试）






FPGA interface
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,ucf sym: D3 E5 D4 C4
segplay(sharing with led) hexplay
00000002
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
hexplay pin: an2 an1 an0 d3 d2 d1 d0
soft clock button
None
clk btn pins: clk_btn
xdc,ucf sym: B18


FPGA interface
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,ucf sym: D3 E5 D4 C4
segplay(sharing with led) hexplay
00000003
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
hexplay pin: an2 an1 an0 d3 d2 d1 d0
soft clock button
None
clk btn pins: clk_btn
xdc,ucf sym: B18


FPGA interface
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,ucf sym: D3 E5 D4 C4
segplay(sharing with led) hexplay
00000005
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
hexplay pin: an2 an1 an0 d3 d2 d1 d0
soft clock button
None
clk btn pins: clk_btn
xdc,ucf sym: B18