计算机组成原理 实验报告

实验题目：单周期CPU设计
学生姓名：阿非提
学生学号：PB20111633
完成日期：2022.4.21

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

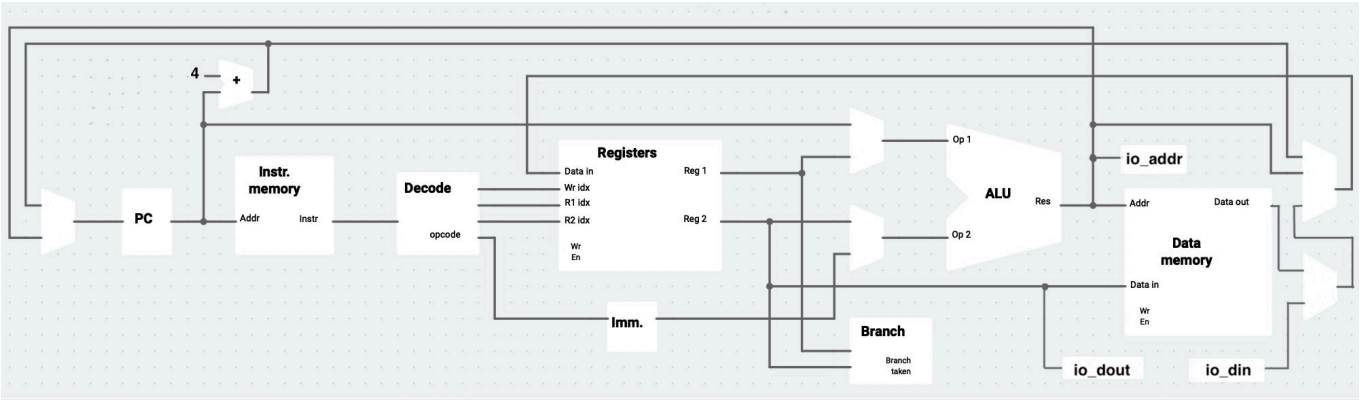
实验平台

- Vivado
- Rars

实验练习

- **cpu设计**

  在Ripes 提供的单周期处理器的数据通路的基础上根据实验要求做修改得到如下数据通路



  设计文件

  寄存器堆

```
module registers(
    input clk,rst,
    input [4:0] read_register1,
    input [4:0] read_register2,
    input [4:0] write_register,
```

```
    input [31:0] write_data,
    input write_enable,
    output [31:0] read_data1,
    output [31:0] read_data2,
    //debug port
    input [4:0] debug_register,
    output [31:0] debug_register_data
);
reg [31:0] Registers[0:31];
assign read_data1 = read_register1? Registers[read_register1] : 0;
assign read_data2 = read_register2? Registers[read_register2] : 0;
assign debug_register_data = debug_register? Registers[debug_register] : 0;
integer  i;
always@(posedge clk or posedge rst)begin
    if(rst)
        for(i = 0; i < 32; i = i + 1)
            Registers[i] <= 0;
    else if(write_enable && (write_register != 5'b0))
        Registers[write_register] <= write_data;
end


endmodule
```

alu

```
module alu(
    input [31:0] operant1,operant2,
    input operation_code, //0:add , 1:sub
    output zero,
    output reg [31:0] result
);

wire signed signed_operant1;
wire signed signed_operant2;

assign signed_operant1 = operant1;
assign signed_operant2 = operant2;

assign zero = (result == 0)? 1:0;

always@(*)begin
    if(operation_code)
        result = operant1 - operant2;
    else
        result = operant1 + operant2;
end
```

```
endmodule
```

立即数扩展

```
module immediate_Generator(
    input [31:0] inst,
    output reg [31:0] imme
);
always@(*)begin
    case(inst[6:0])
    7'b0000011: imme = {{21{inst[31]}},inst[30:20]};//I-type
    7'b0010011: imme = {{21{inst[31]}},inst[30:20]};//I-type
    7'b1100111: imme = {{21{inst[31]}},inst[30:20]};//I-type
    7'b0100011: imme = {{21{inst[31]}},inst[30:25],inst[11:7]};//S-type
    7'b1100011: imme = {{20{inst[31]}},inst[7],inst[30:25],inst[11:8],1'b0};//SB-
type
    7'b1101111: imme = {{13{inst[31]}},inst[19:12],inst[20],inst[30:21],1'b0};//
UJ-type
    7'b0010111: imme = {inst[31:12],12'h0};//U-type
    default: imme = 32'h0;
    endcase
end

endmodule
```

pc

```
module PC(
    input clk,rst,
    input [31:0] pc_input,
    output reg [31:0] pc,
    input stop
);

always@(posedge clk or posedge rst)begin
    if(rst)
        pc <= 32'h00003000;
    else if(~stop)
        pc <= pc_input;
end

endmodule
```

branch

```
module branch(
    input comparsion, //0: beq, 1: blt
```

```
    input [31:0] input1,
    input [31:0] input2,
    output reg branch_signal
);
wire signed [31:0] input11;
wire signed [31:0] input22;

assign input11 = input1;
assign input22 = input2;
always@(*)begin
    if(comparsion == 0)
        branch_signal = (input11 == input22)? 1:0;
    else
        branch_signal = (input11 < input22)? 1:0;
end

endmodule
```

cpu

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



//data paths
reg [31:0] pcInput;

wire branchType;
wire branchSignal;

wire [31:0] instruction;

wire regWrite;
reg [31:0] registerIn;
wire [31:0] register1;
wire [31:0] register2;
```

```
wire aluOperation;
wire aluZero;
wire [31:0] aluOperant1;
wire [31:0] aluOperant2;
wire [31:0] aluResult;

wire memoryWrite;
wire [31:0] memoryData;
wire [31:0] signExtension;

wire stop = (instruction == 0)? 1:0;


//decoder
wire [6:0] opcode = instruction[6:0];
wire [6:0] funct7 = instruction[31:25];
wire [2:0] funct3 = instruction[14:12];

//indentify instruction
wire is_addi = (opcode == 7'b0010011) && (funct3 == 3'b000);
wire is_add = (opcode == 7'b0110011) && (funct3 == 3'b000) && (funct7 ==
7'b0000000);
wire is_sub = (opcode == 7'b0110011) && (funct3 == 3'b000) && (funct7 ==
7'b0100000);
wire is_auipc = (opcode == 7'b0010111);
wire is_jal = (opcode == 7'b1101111);
wire is_jalr = (opcode == 7'b1100111);
wire is_beq = (opcode == 7'b1100011) && (funct3 == 3'b000);
wire is_blt = (opcode == 7'b1100011) && (funct3 == 3'b100);
wire is_lw = (opcode == 7'b0000011) && (funct3 == 3'b010);
wire is_sw = (opcode == 7'b0100011) && (funct3 == 3'b010);

//instructions that write to register
assign regWrite = is_add | is_addi | is_sub | is_auipc | is_lw | is_jal |
is_jalr;
always@(*)begin
    if(is_jal | is_jalr)
        registerIn = pc + 4;
    else if(is_lw)
        registerIn = memoryData;
    else
        registerIn = aluResult;
end

//instructions that write to memory
assign memoryWrite = is_sw;
```

```
//instructions that use alu
assign aluOperant1 = (is_auipc | is_beq | is_blt | is_jal)? pc : register1;
assign aluOperant2 = (is_add | is_sub)? register2 : signExtension;
assign aluOperation = is_sub ? 1 : 0; //0:add , 1:sub

//instructions that branch on condition
assign branchType = (is_beq)? 0 : 1; //0:beq , 1:blt

//instructions that change pc
always@(*)begin
    if(((is_beq | is_blt) & branchSignal) | is_jal)
        pcInput = aluResult;
    else if(is_jalr)
        pcInput = {aluResult[31:1],1'b0};
    else
        pcInput = pc + 4;
end

//PC
PC PC(
    .clk(clk),
    .rst(rst),
    .pc_input(pcInput),
    .pc(pc),
    .stop(stop)
);

//branch
branch branch(
    .comparsion(branchType),
    .input1(register1),
    .input2(register2),
    .branch_signal(branchSignal)
);

//registers
registers registers(
    .clk(clk),
    .rst(rst),
    .read_register1(instruction[19:15]),
    .read_register2(instruction[24:20]),
    .write_register(instruction[11:7]),
    .write_data(registerIn),
    .write_enable(regWrite),
    .read_data1(register1),
    .read_data2(register2),
    //debug port
    .debug_register(m_rf_addr[4:0]),
    .debug_register_data(rf_data)
```

```
);


//alu
alu alu(
    .operant1(aluOperant1),
    .operant2(aluOperant2),
    .operation_code(aluOperation),
    .zero(aluZero),
    .result(aluResult)
);


//immediate_generator
immediate_Generator immediate_Generator(
    .inst(instruction),
    .imme(signExtension)
);

//instrution_ram
wire [7:0] inst_ram_addr;


instruction_mem instruction_mem(
    .a(pc[9:2]),
    .spo(instruction)
);


//data_ram
wire [31:0] data_ram_data;
wire data_ram_write;
wire is_data_ram;

// is address in 0x0000_0000 ~ 0x0000_03ff range
assign is_data_ram = (aluResult[31:10] == 22'b0)? 1 : 0;

//if address is I/O memory address
assign io_we = (is_data_ram == 0)? memoryWrite : 0;
assign io_dout = register2;
assign io_addr = aluResult[7:0];

assign memoryData = (is_data_ram == 1)? data_ram_data : io_din;

//if address is memory address
assign data_ram_write = (is_data_ram == 1)? memoryWrite : 0;


data_mem data_mem(
```

```
    .clk(clk),
    .a(aluResult[9:2]),
    .d(register2),
    .we(data_ram_write),
    .spo(data_ram_data),
    //debug port
    .dpra(m_rf_addr),
    .dpo(m_data)
);

endmodule
```

- **cpu功能仿真**

运行使用的汇编代码

```
.data
NUM: .word 0xd

.text
    lw t1,NUM
    add t2,t1,zero
    addi t1,t1,23
    sub t3,t1,t2
    sw t3,0(zero)
    auipc t1,1
    beq zero,zero,GOTO
    add t1,zero,zero
GOTO:
    blt t3,t2,GOTO1
    auipc t1,0
GOTO1:
    jal GOTO2
    add t1,zero,zero
GOTO2:  jalr t1

Exit:
```

rars 生成的代码

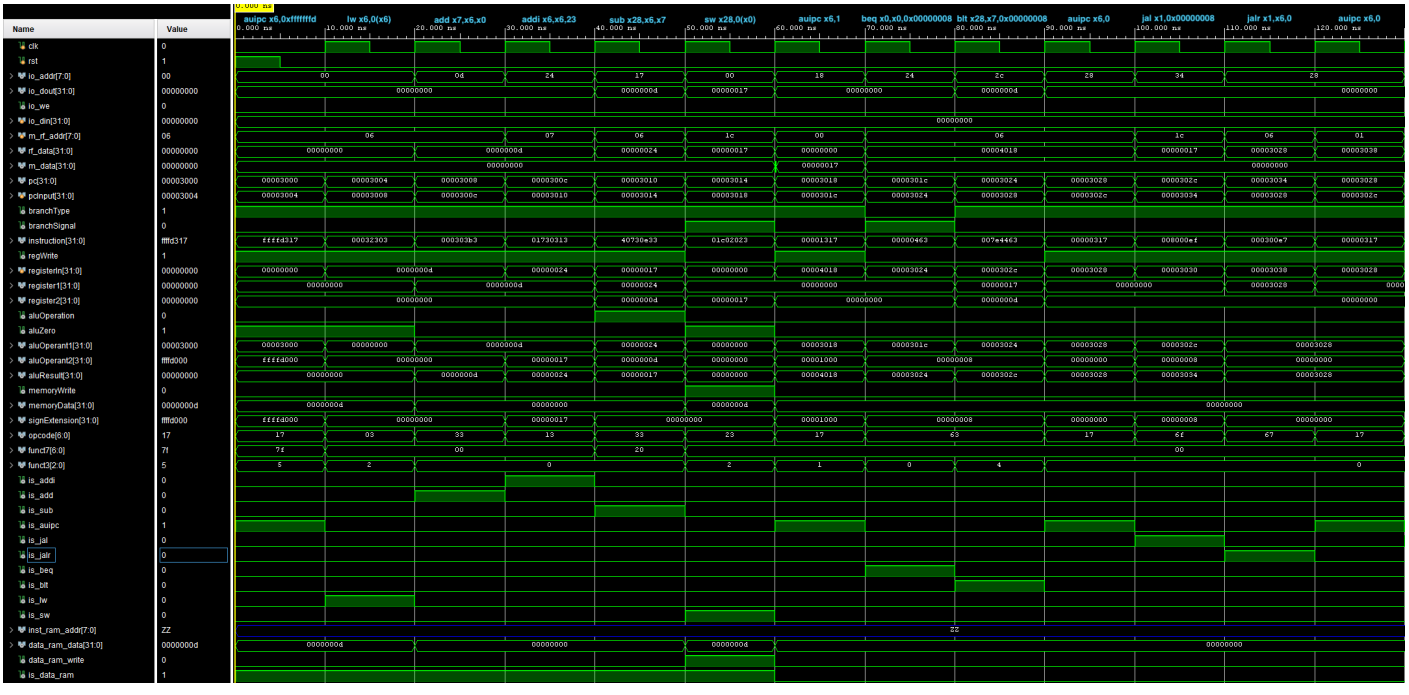
```
Address     Code        Basic                        Line Source

0x00003000  0xffffd317  auipc x6,0xfffffffd          5      lw t1,NUM
0x00003004  0x00032303  lw x6,0(x6)
0x00003008  0x000303b3  add x7,x6,x0                 6      add t2,t1,zero
0x0000300c  0x01730313  addi x6,x6,23                7      addi t1,t1,23
0x00003010  0x40730e33  sub x28,x6,x7                8      sub t3,t1,t2
```

```
0x00003014  0x01c02023  sw x28,0(x0)                    9     sw t3,0(zero)
0x00003018  0x00001317  auipc x6,1                      10    auipc t1,1
0x0000301c  0x00000463  beq x0,x0,0x00000008            11    beq zero,zero,GOTO
0x00003020  0x00000333  add x6,x0,x0                    12    add t1,zero,zero
0x00003024  0x007e4463  blt x28,x7,0x00000008           14    blt t3,t2,GOTO1
0x00003028  0x00000317  auipc x6,0                      15    auipc t1,0
0x0000302c  0x008000ef  jal x1,0x00000008               17    jal GOTO2
0x00003030  0x00000333  add x6,x0,x0                    18    add t1,zero,zero
0x00003034  0x000300e7  jalr x1,x6,0                    19    jalr t1
```

rars 生成的 text COE文件 与 data COE文件

```
ffffd317          0000000d
00032303
000303b3
01730313
40730e33
01c02023
00001317
00000463
00000333
007e4463
00000317
008000ef
00000333
000300e7
```

仿真结果



- **测试斐波那契数列**

测试汇编代码

```
.data
    F0: .word 0x0
    F1: .word 0x1
    N: .word 0xa
.text
    lw t0,F0
    lw t1,F1
    lw t2,N
    addi t3,t2,-2
LOOP:   blt t3,zero,STOP
    add t2,t1,t0
    add t0,t1,zero
    add t1,t2,zero
    addi t3,t3,-1
    jal LOOP
STOP:
```

rars 生成的 text COE文件 与 data COE文件

```
ffffd297        00000000
0002a283        00000001
ffffd317        0000000a
ffc32303
ffffd397
ff83a383
ffe38e13
000e4c63
005303b3
000302b3
00038333
fffe0e13
fedff0ef
```

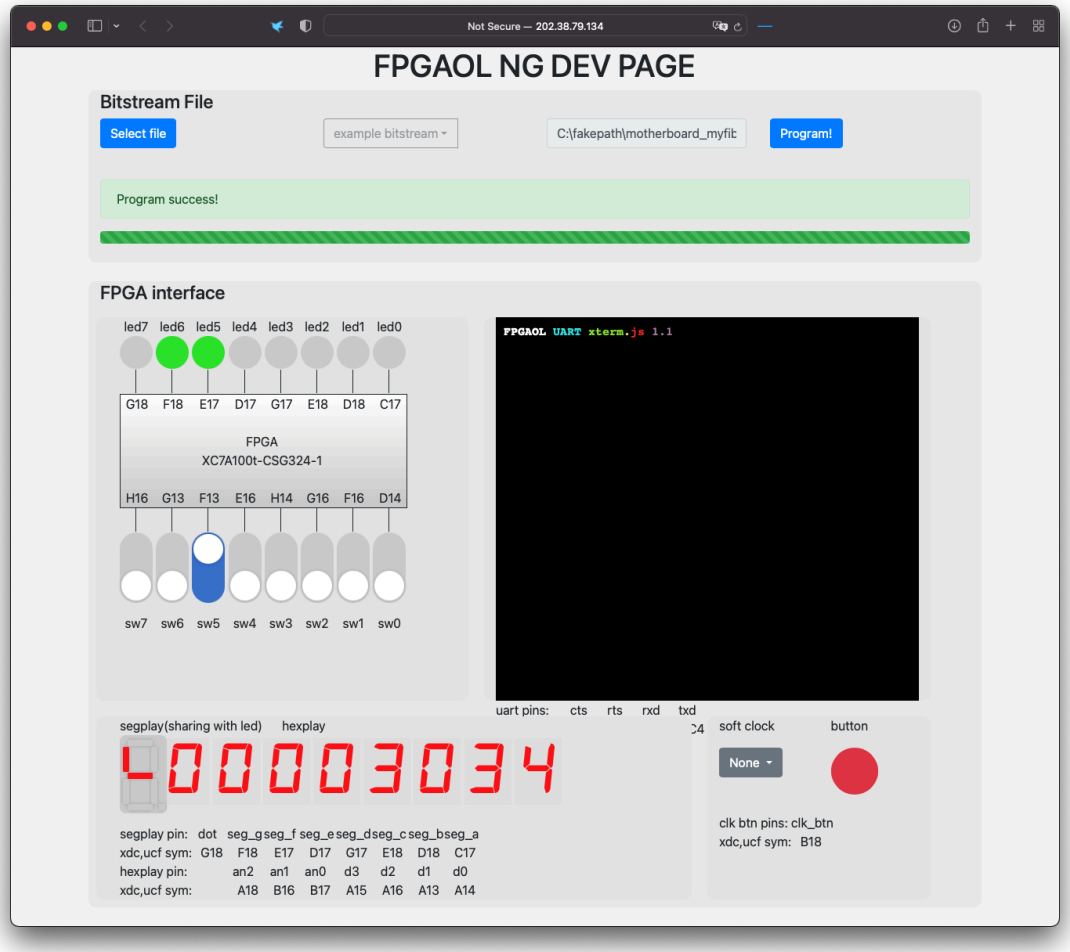
设计文件（其中pdu为群文件提供的模块）

```
module motherboard(
    input clk,rst,      //clk,sw7
    input run,          //sw6
    input step,         //button
    input valid,        //sw5
    input [4:0] in,     //sw4-0
    output [1:0] check, //led6-5
```

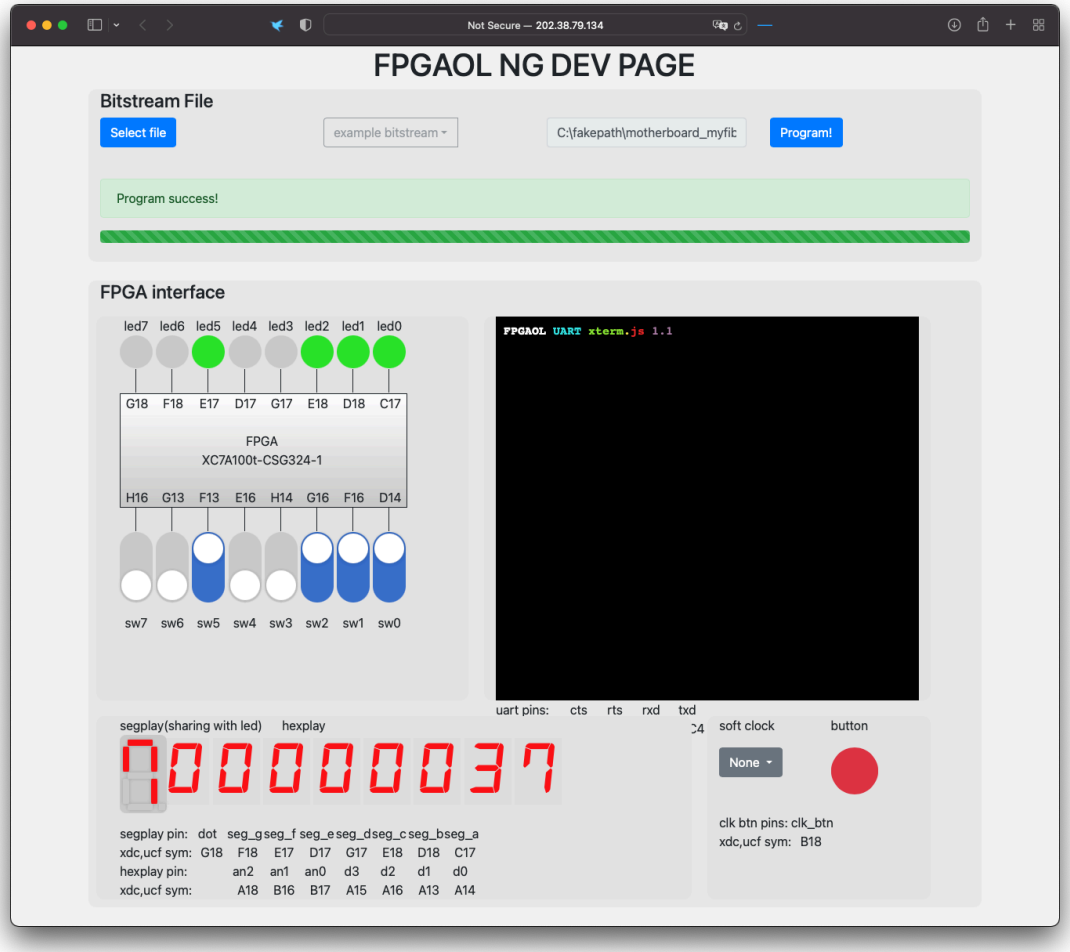
```
    output [4:0] out0,  //led4-0
    output [2:0] an,    //an
    output [3:0] seg,   //seg
    output ready       //led7
);
wire cpu_clk;
wire [7:0] io_addr;
wire [31:0] io_dout;
wire io_we;
wire [31:0] io_din;
wire [7:0] m_rf_addr;
wire [31:0] rf_data;
wire [31:0] m_data;
wire [31:0] pc;

cpu cpu(
    .clk(cpu_clk),
    .rst(rst),
    .io_addr(io_addr),
    .io_dout(io_dout),
    .io_we(io_we),
    .io_din(io_din),
    .m_rf_addr(m_rf_addr),
    .rf_data(rf_data),
    .m_data(m_data),
    .pc(pc)
);

pdu pdu(
    .clk(clk),
    .rst(rst),
    .run(run),
    .step(step),
    .clk_cpu(cpu_clk),
    .valid(valid),
    .in(in),
    .check(check),
    .out0(out0),
    .an(an),
    .seg(seg),
    .ready(ready),
    .io_addr(io_addr),
    .io_dout(io_dout),
    .io_we(io_we),
    .io_din(io_din),
    .m_rf_addr(m_rf_addr),
    .rf_data(rf_data),
    .m_data(m_data),
    .pc(pc)
);

endmodule
```

运行结果

当停止运行时pc的状态


Not Secure — 202.38.79.134
FPGAOL NG DEV PAGE
Bitstream File
Select file
example bitstream
C:\fakepath\motherboard_myfil
Program!
Program success!
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
segplay(sharing with led) hexplay
00003034
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
xdc,ucf sym: A18 B16 B17 A15 A16 A13 A14
soft clock
button
None
clk btn pins: clk_btn
xdc,ucf sym: B18

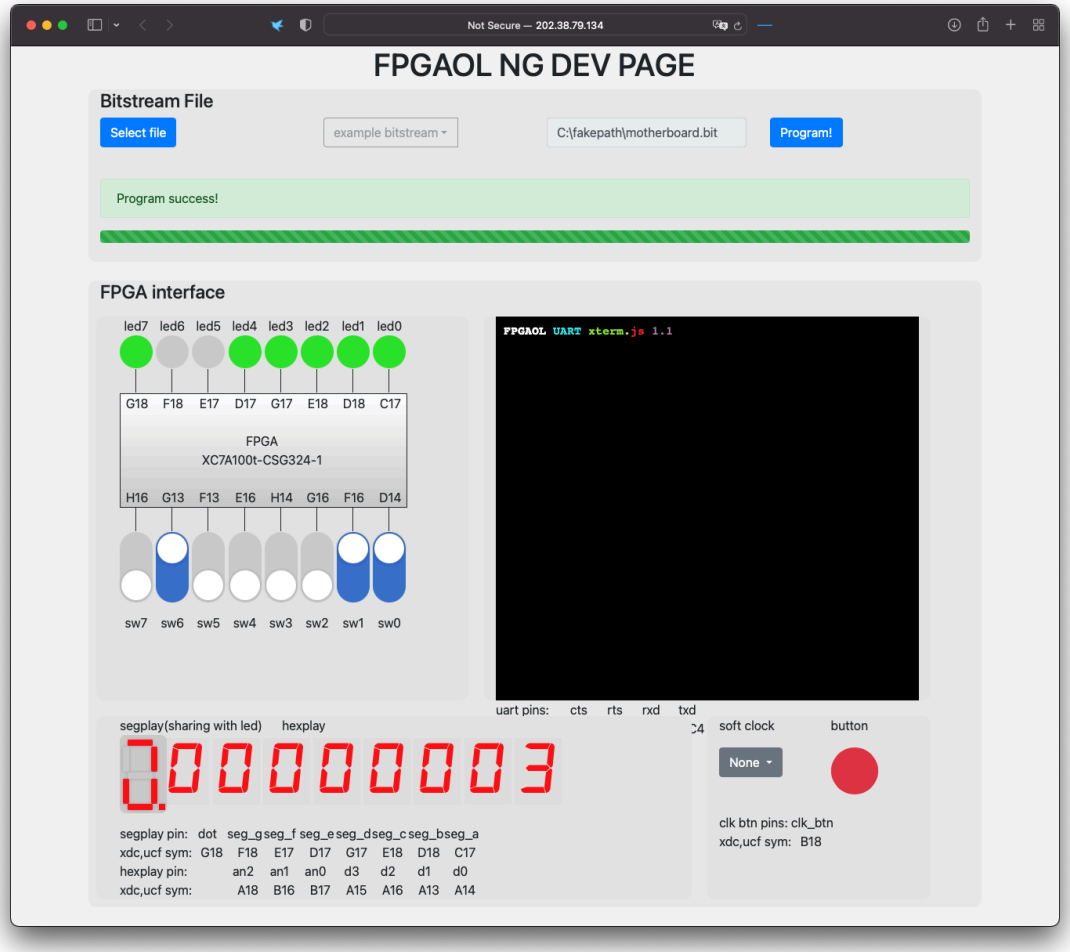
t2（x7）寄存器的状态

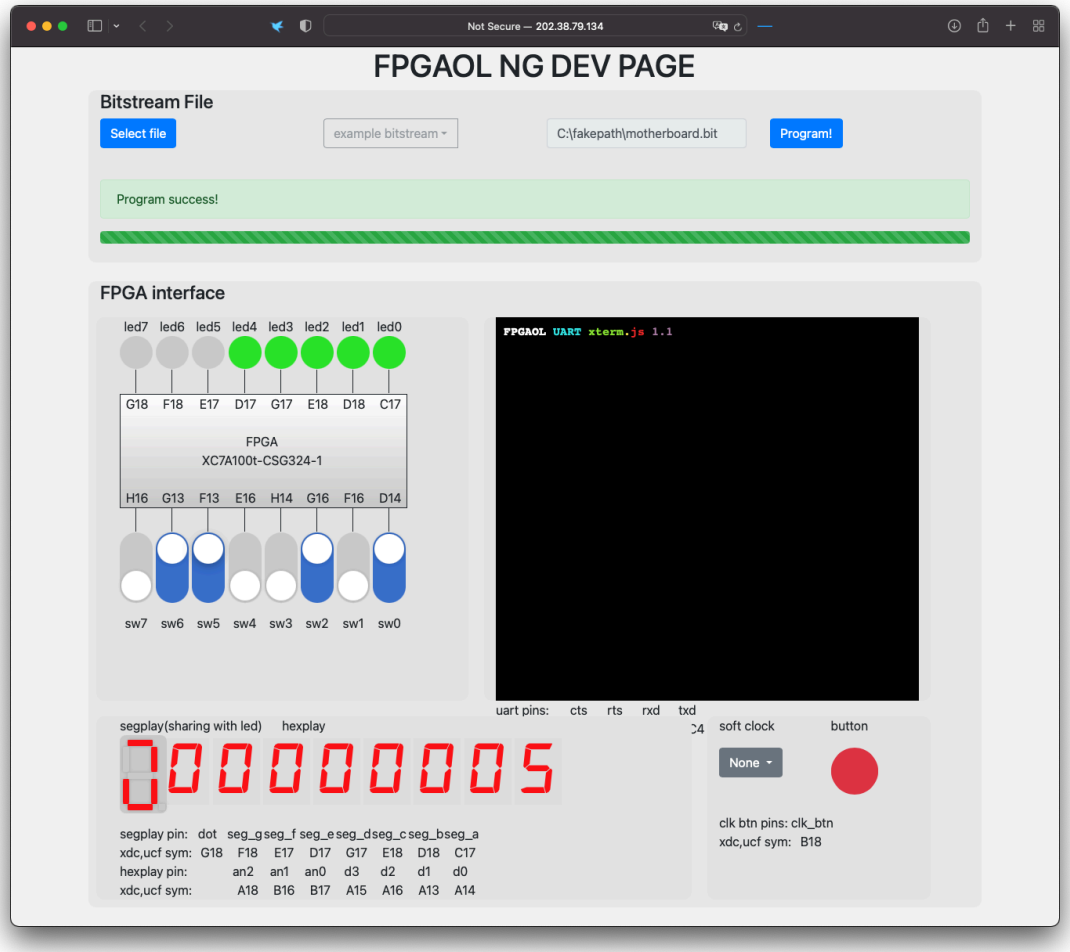

Not Secure — 202.38.79.134
FPGAOL NG DEV PAGE
Bitstream File
Select file
example bitstream
C:\fakepath\motherboard_myfil
Program!
Program success!
FPGA interface
led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
FPGAOL UART xterm.js 1.1
uart pins: cts rts rxd txd
segplay(sharing with led) hexplay
00000037
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
xdc,ucf sym: A18 B16 B17 A15 A16 A13 A14
soft clock
button
None
clk btn pins: clk_btn
xdc,ucf sym: B18

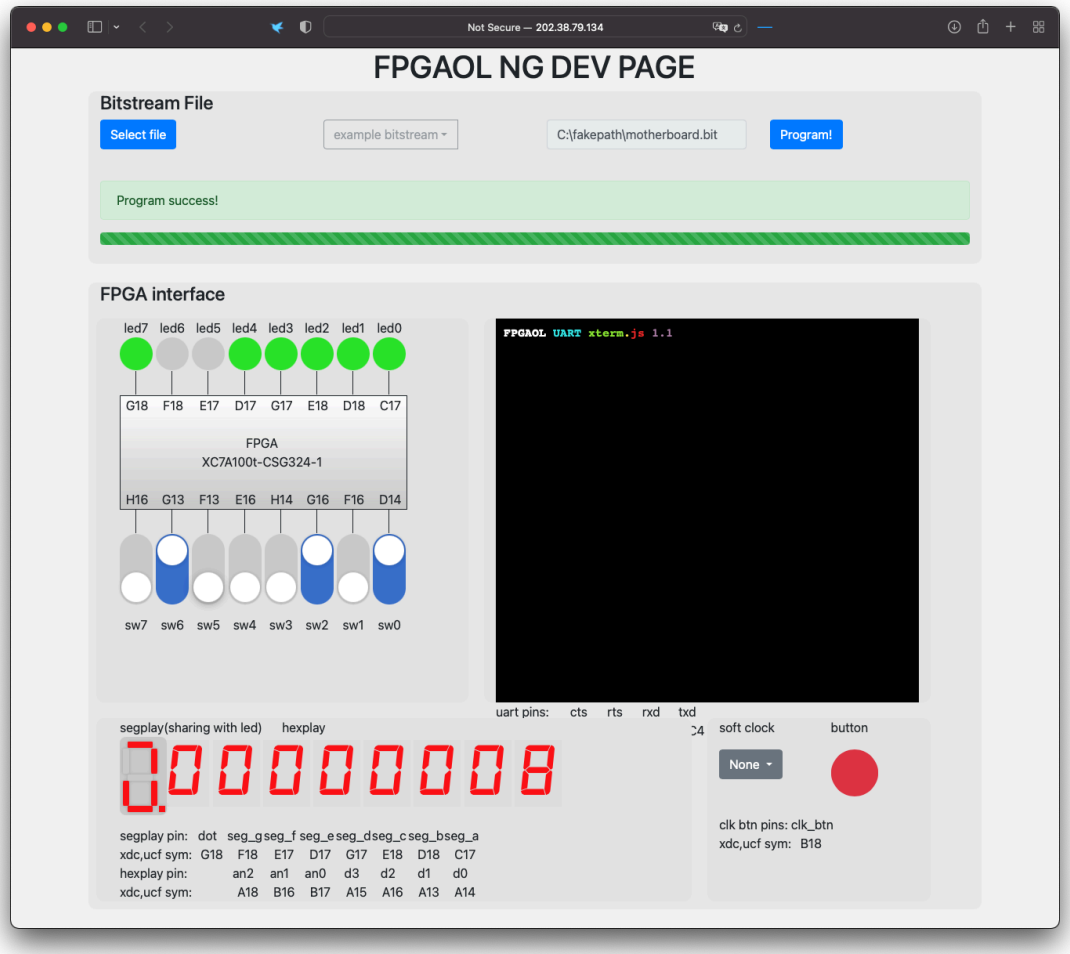
- **测试I/O版斐波拉契数列**

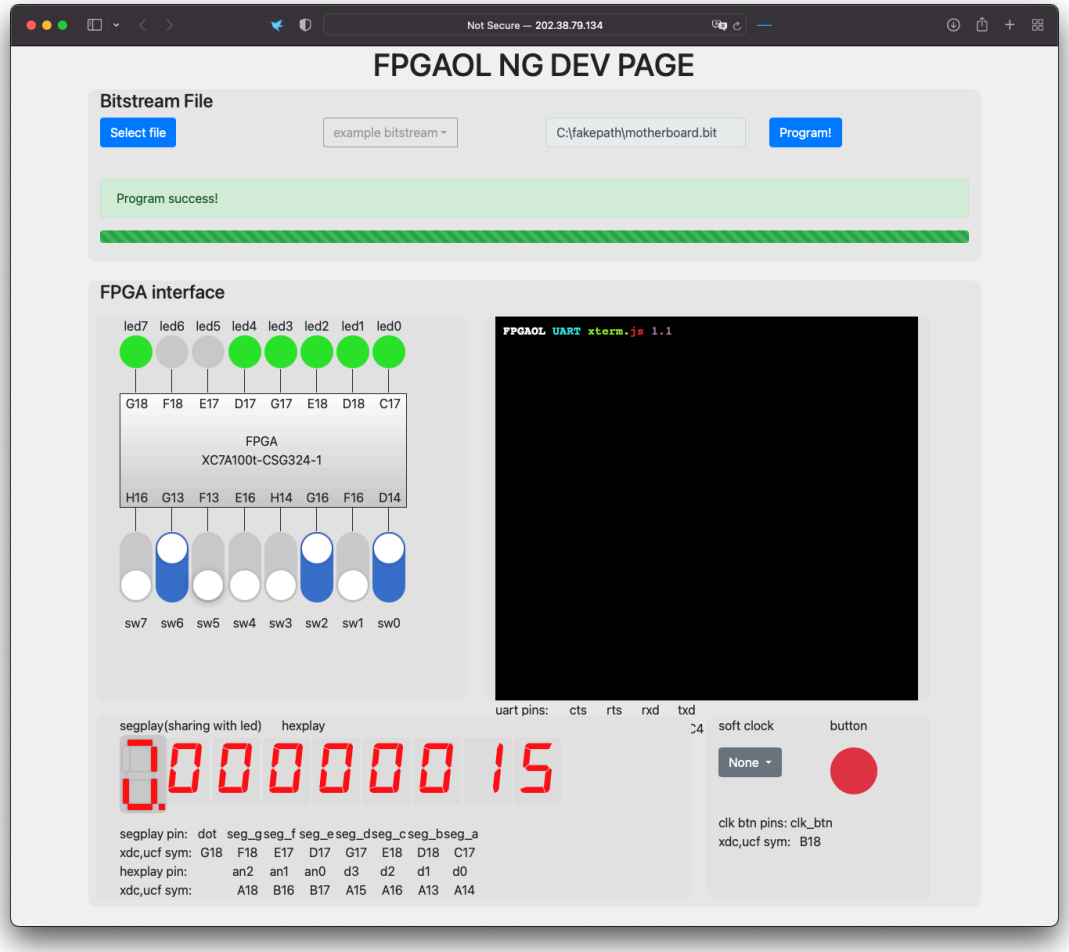
  测试所需的 coe 文件为群文件提供的 fib_test.coe 文件

  运行结果




Not Secure — 202.38.79.134
FPGAOL NG DEV PAGE
Bitstream File
Select file
example bitstream
C:\fakepath\motherboard.bit
Program!
Program success!
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
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
xdc,ucf sym: A18 B16 B17 A15 A16 A13 A14
soft clock
button
None
clk btn pins: clk_btn
xdc,ucf sym: B18


Not Secure — 202.38.79.134
FPGAOL NG DEV PAGE
Bitstream File
Select file
example bitstream
C:\fakepath\motherboard.bit
Program!
Program success!
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
segplay(sharing with led) hexplay
00000008
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
xdc,ucf sym: A18 B16 B17 A15 A16 A13 A14
soft clock
button
None
clk btn pins: clk_btn
xdc,ucf sym: B18



总结与思考

- 通过本次试验我更深入的了解了risc指令集下的单周期cpu的实现以及需要注意的事项 。
- 本次试验难易程适中。
- 本次试验任务量适中。