lab5 流水线CPU设计

姓名：宋玮　学号：PB20151793　实验日期：2022.5.11

实验题目

lab5 流水线CPU设计

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

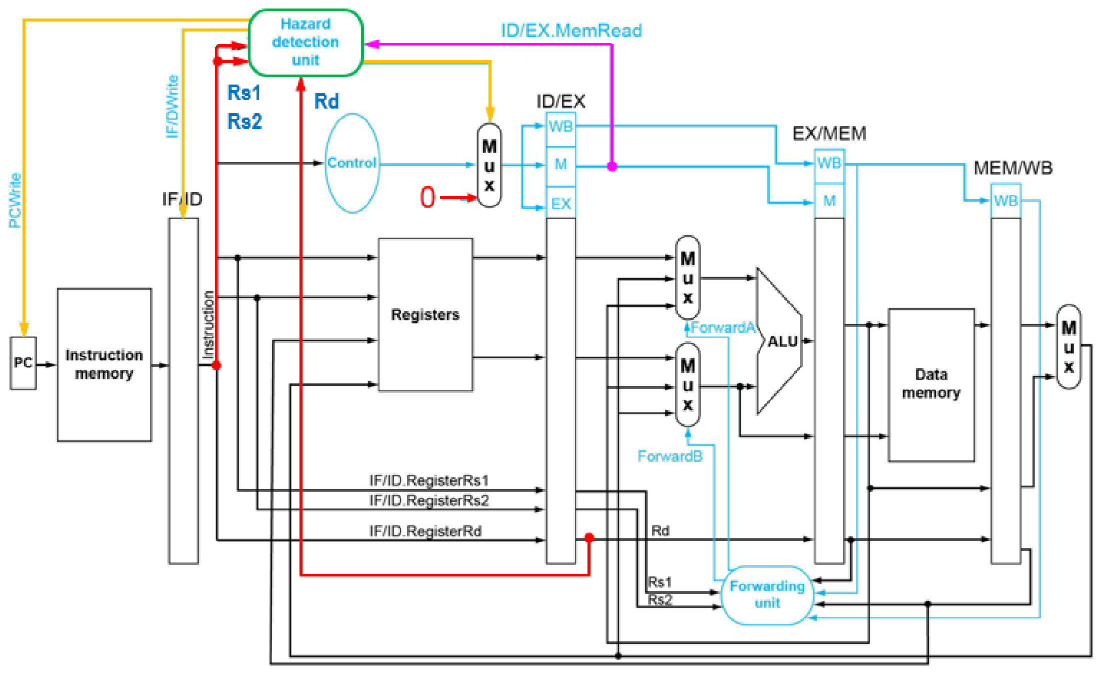
实验平台

Rars，fpgaol，vivado

实验过程

1.设计有数据和控制相关处理的流水线CPU

(1)基本数据通路如下



> **1.add，addi指令可能用到前递（Fowarding）**
>
> **2.load-use：需要用到停顿（stall）和前递**
>
> **3.branch&jal：需要用到控制冒险模块**

(2)CPU各模块

①rf（寄存器堆）

本寄存器堆是在Lab4寄存器堆模块基础上修改而来，使其满足**写优先(Write First)**，即在对同一寄存器读写时，写数据可立即从读数据输出.

```
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 = (we && wa != 0 && wa == ra0) ? wd : regfile[ra0];
assign rd1 = (we && wa != 0 && wa == ra1) ? wd : 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
```

②alu

```
module alu #(
    parameter WIDTH = 32
)(
    input[WIDTH-1:0] a,
    input[WIDTH-1:0] b,
    input [2:0] f,
    output reg [WIDTH-1:0] y,
    output z
);

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;
        default: y = 0;
    endcase
end

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

endmodule
```

③alu_control

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

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

④control

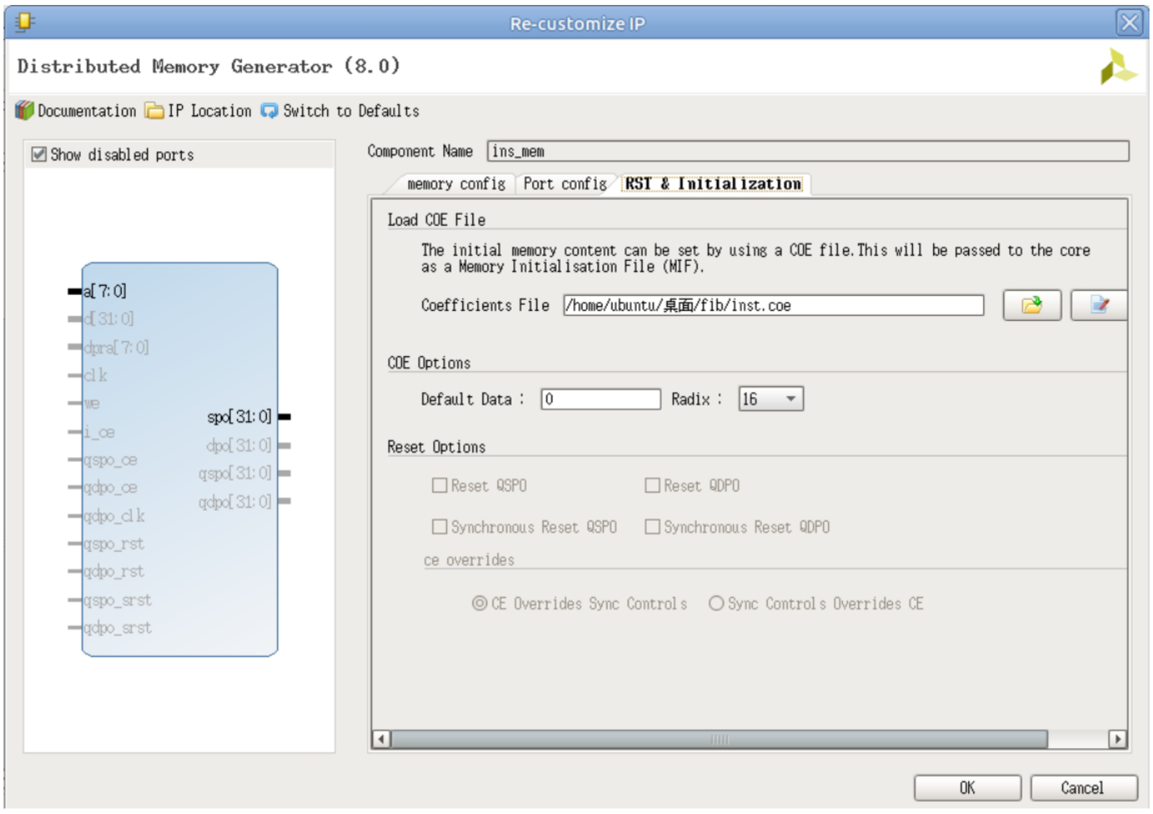
```
module control(
input [6:0] ins,
output reg ALUSrc,RegWrite,MemRead,MemWrite,Branch,JUMP,
output reg [1:0] ALUOp, reg [1:0] MemtoReg
    );

always@(*)
begin
    case(ins)
        7'b0110011:    //add
            begin
                ALUSrc = 0; MemtoReg = 2'b00; RegWrite = 1; MemRead = 0; MemWrite = 0;
                Branch = 0; JUMP = 0; ALUOp = 2'b10;
            end
        7'b0000011:    //lw
            begin
                ALUSrc = 1; MemtoReg = 2'b01; RegWrite = 1; MemRead = 1; MemWrite = 0;
                Branch = 0; JUMP = 0; ALUOp = 2'b00;
            end
        7'b0100011:    //sw
            begin
                ALUSrc = 1; MemtoReg = 0; RegWrite = 0; MemRead = 0; MemWrite = 1;
                Branch = 0; JUMP = 0; ALUOp = 2'b00;
            end
        7'b0010011:     //addi
            begin
                ALUSrc = 1; MemtoReg = 2'b00; RegWrite = 1; MemRead = 0; MemWrite = 0;
                Branch = 0; JUMP = 0; ALUOp = 2'b10;
            end
        7'b1100011:     // beq
            begin
                ALUSrc = 0; MemtoReg = 0; RegWrite = 0; MemRead =0 ; MemWrite = 0;
                Branch = 1; JUMP = 0; ALUOp = 2'b01;
            end
        7'b1101111:     //jal
            begin
                ALUSrc = 0; MemtoReg = 2'b10; RegWrite = 1; MemRead =0 ; MemWrite = 0;
                Branch = 1; JUMP = 1; ALUOp = 2'b00;
            end
        default:
            begin
                ALUSrc = 0; MemtoReg = 2'b00; RegWrite = 0; MemRead =0 ; MemWrite = 0;
                Branch = 0; JUMP = 0; ALUOp = 2'b00;
            end
    endcase
end
endmodule
```

⑤immg（立即数模块）

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

always @(*)
begin
    case(ins[6:0])
        7'b0000011:  imm = {{20{ins[31]}},ins[31:20]};//lw
        7'b0100011:  imm = {{20{ins[31]}},ins[31:25],ins[11:7]}; //sw
        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
        7'b1101111:  imm = {{12{ins[31]}},ins[19:12],ins[20],ins[30:21],1'b0};//jal
        default:    imm = 0; //add
    endcase
end
endmodule
```

⑥ins_mem & data_mem

指令存储器（ins_mem）和数据存储器（data_mem）由分布式存储器ip核生成



Distributed Memory Generator (8.0)

[✓] Show disabled ports

Component Name: ins_mem

Load COE File

The initial memory content can be set by using a COE file.This will be passed to the core as a Memory Initialisation File (MIF).

Coefficients File: /home/ubuntu/桌面/fib/inst.coe

COE Options

Default Data : 0   Radix : 16

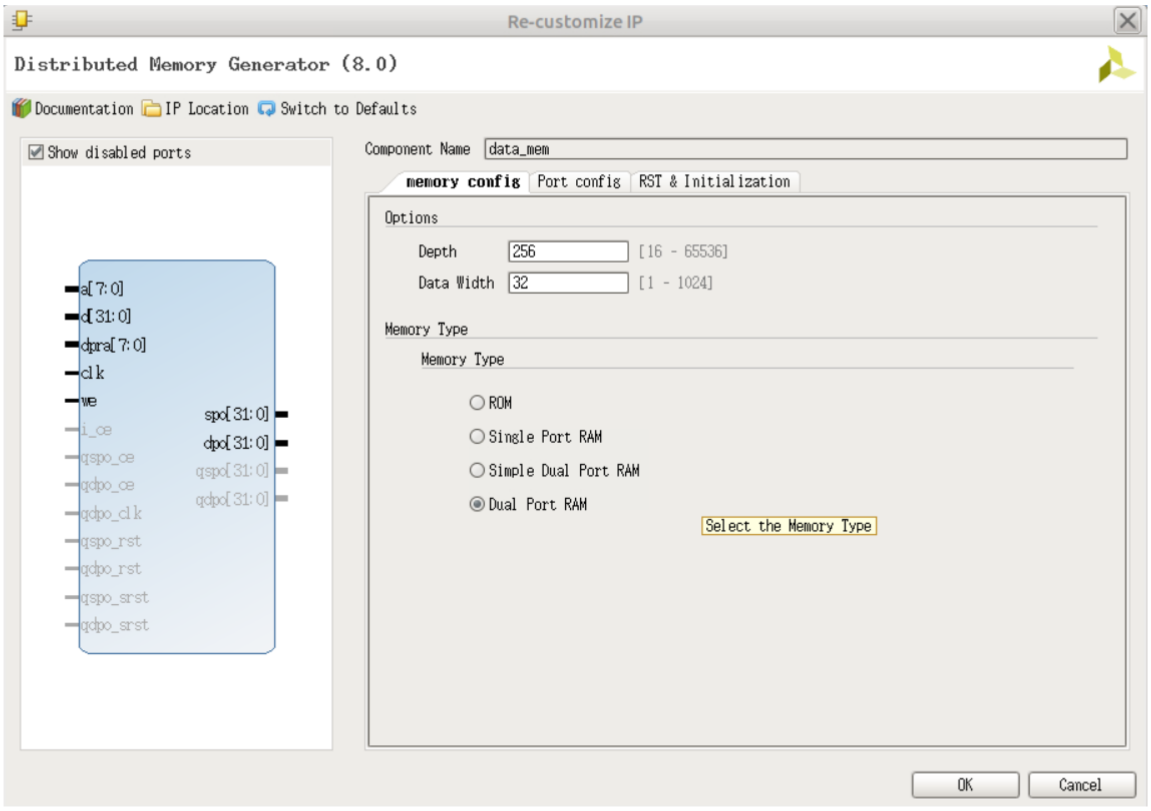
Reset Options

☐ Reset QSPO   ☐ Reset QDPO

☐ Synchronous Reset QSPO   ☐ Synchronous Reset QDPO

ce overrides

[✓ CE Overrides Sync Controls]   Sync Controls Overrides CE


Re-customize IP

Distributed Memory Generator (8.0)

Documentation IP Location Switch to Defaults

[✓] Show disabled ports

Component Name: data_mem

memory config | Port config | RST & Initialization

Options

Depth: 256 [16 - 65536]

Data Width: 32 [1 - 1024]

Memory Type

Memory Type

○ ROM
○ Single Port RAM
○ Simple Dual Port RAM
[✓ ● Dual Port RAM]

Select the Memory Type

⑦CPU主体

```
module cpu_pl(
input clk,rst,
output [7:0] io_addr,
output [31:0] io_dout,
output io_we,
input [31:0] io_din,
input [7:0] m_rf_addr ,
output [31:0] rf_data ,
output [31:0] m_data ,
//PC/IF/ID
  output [31:0] pc,
  output [31:0] pcd,
  output [31:0] ir,
  output [31:0] pcin,

//ID/EX
  output [31:0] pce,
  output [31:0] a,
  output [31:0] b,
  output [31:0] IMM,
  output [4:0] rd,
  output [31:0] ctrl,

  //EX/MEM
  output [31:0] y,
  output [31:0] bm,
  output [4:0] rdm,
  output [31:0] ctrlm,

  //MEM/WB
  output [31:0] yw,
  output [31:0] mdr,
  output [4:0] rdw,
  output [31:0] ctrlw
);

reg [31:0] PC;
wire [7:0] apc; //[9:2]
wire Zero;
wire Branch,MemRead,MemWrite,ALUSrc,RegWrite,JUMP;
wire [1:0] ALUOp ,MemtoReg;
wire [31:0] ins, imm;//, imm1; // imm1Ïshift1λ
wire [31:0] readData1,readData2,ALU_input2;
reg [31:0] alu_input1,alu_input2;
wire [31:0] Mem_ReadData;
reg [31:0] mem_write_data;
wire [31:0] nPC_4,PC_offset;
wire [31:0] nPC;
wire which_PC;
wire [31:0] ALU_result;
reg [31:0] writeData;
wire [2:0] sel;
reg [2:0] forwardA,forwardB,forwardC;
reg PCWrite, IF_ID_Write , ID_EX_FLUSH  ,IF_ID_FLUSH;
wire Branch_hazard;

assign apc = PC[9:2];
ins_mem IMem(.a(apc),.spo(ins));

//IF
reg [31:0]IF_ID_PC;
reg [31:0]IF_ID_ins;

always@(posedge clk or posedge rst)
begin
    if(rst || IF_ID_FLUSH)
        begin
            IF_ID_PC <= 0;
            IF_ID_ins <= 0;
        end
    else if(IF_ID_Write)
        begin
            IF_ID_PC <= PC;
            IF_ID_ins <= ins;
        end
    else
        begin
            IF_ID_PC <= IF_ID_PC;
            IF_ID_ins <= IF_ID_ins;
        end
end

//ID
immg immg(.ins(IF_ID_ins),.imm(imm));

control control(.ins(IF_ID_ins[6:0]),
.ALUSrc(ALUSrc),.RegWrite(RegWrite),.MemRead(MemRead),
.MemWrite(MemWrite),.Branch(Branch),.JUMP(JUMP),
.ALUOp(ALUOp), .MemtoReg(MemtoReg)  );
```

```verilog

reg [31:0]ID_EX_PC;
reg [31:0]ID_EX_rd1, ID_EX_rd2;
reg [31:0] ID_EX_imm;
reg [4:0] ID_EX_ra1,ID_EX_ra2;
reg [4:0] ID_EX_rs1; //ID_EX_rs1=rd
reg ID_EX_ALUSrc,ID_EX_RegWrite,ID_EX_MemRead,ID_EX_MemWrite,ID_EX_Branch,ID_EX_JUMP;
reg [1:0] ID_EX_ALUOp, ID_EX_MemtoReg;
always@(posedge clk or posedge rst)
begin
    if(rst || ID_EX_FLUSH)
        begin
            ID_EX_ra1 <= 0;
            ID_EX_ra2 <= 0;
            ID_EX_PC  <= 0;
            ID_EX_rd1 <= 0;
            ID_EX_rd2 <= 0;
            ID_EX_imm <= 0;
            ID_EX_rs1 <= 0;
            ID_EX_ALUSrc <= 0; ID_EX_RegWrite <= 0; ID_EX_MemRead <= 0;
            ID_EX_MemWrite <= 0; ID_EX_Branch <= 0; ID_EX_JUMP <= 0;
            ID_EX_ALUOp <= 0; ID_EX_MemtoReg <= 0;
        end
    else
        begin
            ID_EX_ra1 <= IF_ID_ins[19:15];//rs1
            ID_EX_ra2 <= IF_ID_ins[24:20];//rs2
            ID_EX_PC <= IF_ID_PC;
            ID_EX_rd1 <= readData1;   //rf_data1
            ID_EX_rd2 <= readData2;   //rf_data2
            ID_EX_imm <= imm;
            ID_EX_rs1 <= IF_ID_ins[11:7];//rd
            ID_EX_ALUSrc <= ALUSrc; ID_EX_RegWrite <= RegWrite; ID_EX_MemRead <= MemRead;
            ID_EX_MemWrite <= MemWrite; ID_EX_Branch <= Branch; ID_EX_JUMP <= JUMP;
            ID_EX_ALUOp <= ALUOp; ID_EX_MemtoReg <= MemtoReg;
        end
end

//EX
//assign imm1 = ID_EX_imm >> 1;
assign PC_offset = ID_EX_PC + ID_EX_imm;//+ imm1[7:0];   //JUMP_BRANCH PC
assign ALU_input2 = ID_EX_ALUSrc ? ID_EX_imm : ID_EX_rd2;

alu_control alu_control(.ALUOP(ID_EX_ALUOp),.sel(sel));
alu # (32) alu(.a(alu_input1),.b(alu_input2),.f(sel),.z(Zero),.y(ALU_result));

```

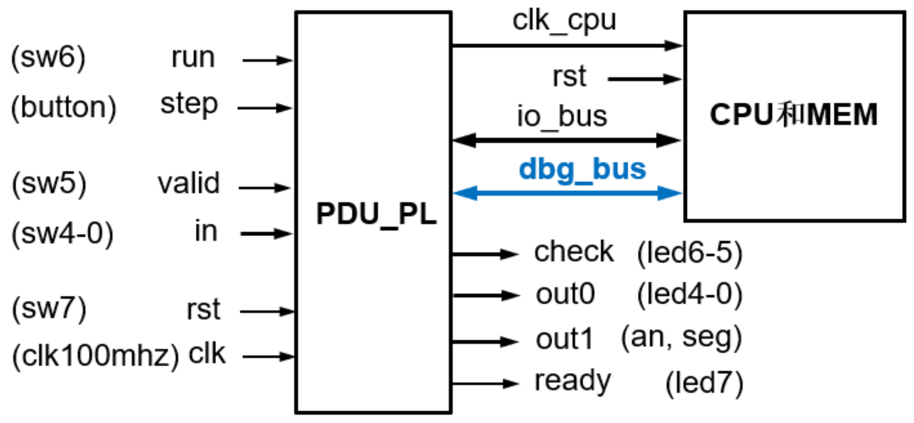
```verilog
reg [4:0] EX_MEM_rs1;
reg EX_MEM_RegWrite,EX_MEM_MemRead,EX_MEM_MemWrite,EX_MEM_Branch,EX_MEM_JUMP;
reg [1:0] EX_MEM_MemtoReg;
reg [31:0] EX_MEM_PC;
reg [31:0] EX_MEM_ALU_result, EX_MEM_rd2;
reg EX_MEM_Zero;

always@(posedge clk or posedge rst)
begin
    if (rst )
        begin
            EX_MEM_rs1 <= 0;
            EX_MEM_RegWrite <= 0; EX_MEM_MemRead <= 0;
            EX_MEM_MemWrite <= 0; EX_MEM_Branch <= 0;
            EX_MEM_JUMP <= 0; EX_MEM_MemtoReg <= 0;
            EX_MEM_Zero <= 0; EX_MEM_ALU_result <=  0;
            EX_MEM_rd2 <= 0; EX_MEM_PC <= 0;
        end
    else
        begin
            EX_MEM_rs1 <= ID_EX_rs1;
            EX_MEM_RegWrite <= ID_EX_RegWrite; EX_MEM_MemRead <= ID_EX_MemRead;
            EX_MEM_MemWrite <= ID_EX_MemWrite; EX_MEM_Branch <= ID_EX_Branch;
            EX_MEM_JUMP <= ID_EX_JUMP; EX_MEM_MemtoReg <= ID_EX_MemtoReg;
            EX_MEM_Zero <= Zero; EX_MEM_ALU_result <=  ALU_result;
            EX_MEM_rd2 <= mem_write_data; EX_MEM_PC <= ID_EX_PC;
        end
end

//MEM
reg [1:0] MEM_WB_MemtoReg;
reg MEM_WB_RegWrite;
reg [31:0] MEM_WB_ALU_result;
reg [4:0] MEM_WB_rs1;
reg [31:0] MEM_WB_Mem_ReadData;
reg [31:0] MEM_WB_PC ;
wire [31:0] IO_ADDRESS;
wire DMem_write;
wire [31:0] MemReadDataTrue;
wire isIO;

//IO
assign IO_ADDRESS = EX_MEM_ALU_result; //<< 2;
assign isIO = IO_ADDRESS[10];
assign DMem_write = ~isIO && EX_MEM_MemWrite;
assign io_we = (isIO && EX_MEM_MemWrite) ? 1'b1 : 1'b0;
assign MemReadDataTrue = isIO ? io_din : Mem_ReadData;
assign io_addr = IO_ADDRESS[7:0];
assign io_dout = EX_MEM_rd2;

data_mem DMem(.a(EX_MEM_ALU_result[9:2]),.d(EX_MEM_rd2),.dpra(m_rf_addr),.clk(clk),.we(DMem_write),.spo(Mem_ReadData),.dpo(m_data));

always@(posedge clk or posedge rst)
```

```verilog
begin
    if(rst)
        begin
            MEM_WB_MemtoReg <= 0; MEM_WB_RegWrite <= 0;
            MEM_WB_ALU_result <= 0; MEM_WB_rs1 <= 0;
            MEM_WB_Mem_ReadData <= 0; MEM_WB_PC <= 0;
        end
    else
        begin
            MEM_WB_MemtoReg <= EX_MEM_MemtoReg; MEM_WB_RegWrite <= EX_MEM_RegWrite;
            MEM_WB_ALU_result <= EX_MEM_ALU_result; MEM_WB_rs1 <= EX_MEM_rs1;
            MEM_WB_Mem_ReadData <= MemReadDataTrue; MEM_WB_PC <= EX_MEM_PC;
        end
end

//WB
always@(*)
begin
    case(MEM_WB_MemtoReg)
        2'b00: writeData = MEM_WB_ALU_result;
        2'b01: writeData = MEM_WB_Mem_ReadData;
        default: writeData = MEM_WB_PC + 4;
    endcase
end

//PC
assign nPC_4 = PC + 4;
assign nPC = Branch_hazard ? PC_offset : nPC_4;
always@(posedge clk or posedge rst)
begin
    if(rst) PC <= 32'h0000_3000;
    else if(PCWrite) PC <= nPC;
    else PC <= PC;
end

rf #(.m(5),.WIDTH(32)) rf (.clk(clk),.we(MEM_WB_RegWrite),
    .wa(MEM_WB_rs1),.ra0(IF_ID_ins[19:15]),.ra1(IF_ID_ins[24:20]),
    .wd(writeData),.rd0(readData1),.rd1(readData2),
    .rf_addr(m_rf_addr[4:0]),
    .rf_data(rf_data),.rst(rst));

//Fowarding Unit
always@(*)
begin
    if(EX_MEM_RegWrite && EX_MEM_rs1 != 0 && EX_MEM_rs1 == ID_EX_ra1) forwardA = 2'b10;
    else if(MEM_WB_RegWrite && MEM_WB_rs1 != 0 && MEM_WB_rs1 == ID_EX_ra1) forwardA = 2'b01;
    else forwardA = 2'b00;
    if(EX_MEM_RegWrite && EX_MEM_rs1 != 0 && EX_MEM_rs1 == ID_EX_ra2 && ID_EX_ALUSrc != 1) forwardB = 2'b10;
    else if(MEM_WB_RegWrite && MEM_WB_rs1 != 0 && MEM_WB_rs1 == ID_EX_ra2 && ID_EX_ALUSrc != 1) forwardB = 2'b01;
    else forwardB = 2'b00;
    if(EX_MEM_RegWrite && EX_MEM_rs1 != 0 && EX_MEM_rs1 == ID_EX_ra2) forwardC = 2'b10;
    else if(MEM_WB_RegWrite && MEM_WB_rs1 != 0 && MEM_WB_rs1 == ID_EX_ra2) forwardC = 2'b01;
    else forwardC = 2'b00;
end

always@(*)
begin
    case(forwardA)
        2'b10:   alu_input1 = EX_MEM_ALU_result;
        2'b01:   alu_input1 = writeData;
        default: alu_input1 = ID_EX_rd1;
    endcase
    case(forwardB)
        2'b10:   alu_input2 = EX_MEM_ALU_result;
        2'b01:   alu_input2 = writeData;
        default: alu_input2 = ALU_input2;
    endcase
    case(forwardC)
        2'b10:   mem_write_data = EX_MEM_ALU_result;
        2'b01:   mem_write_data = writeData;
        default: mem_write_data = ID_EX_rd2;
    endcase
end

//Hazard detection unit
always@(*)
begin
    if(Branch_hazard)
        begin
            PCWrite = 1;
            IF_ID_Write = 0;
            ID_EX_FLUSH = 1;
            IF_ID_FLUSH = 1;
        end
    else if(ID_EX_MemRead && ((ID_EX_rs1 == IF_ID_ins[19:15])||(ID_EX_rs1 == IF_ID_ins[24:20])))
        begin
            PCWrite = 0;
            IF_ID_Write = 0;
            ID_EX_FLUSH = 1;
            IF_ID_FLUSH = 0;
        end
    else
        begin
            PCWrite = 1;
            IF_ID_Write = 1;
            ID_EX_FLUSH = 0;
            IF_ID_FLUSH = 0;
        end
end

//branch_hazard?
assign Branch_hazard = (ID_EX_Branch & Zero) | ID_EX_JUMP;


//PC/IF/ID
assign pc = PC;
assign pcin = nPC;
assign pcd = IF_ID_PC;
assign ir = IF_ID_ins;

//ID/EX
assign pce = ID_EX_PC;
assign a = ID_EX_rd1;
assign b = ID_EX_rd2;
assign IMM = ID_EX_imm;
assign rd = ID_EX_rs1;
assign ctrl =
{~PCWrite,~IF_ID_Write,IF_ID_FLUSH,ID_EX_FLUSH,2'b0,forwardA,2'b0,
 forwardB,1'b0,ID_EX_RegWrite,ID_EX_MemtoReg,2'b0,ID_EX_MemRead,
 ID_EX_MemWrite,2'b0,ID_EX_JUMP,ID_EX_Branch,4'b0,ID_EX_ALUOp,2'b0};

  //EX/MEM
assign y = EX_MEM_ALU_result;
assign bm = EX_MEM_rd2;
assign rdm = EX_MEM_rs1;
assign ctrlm =
{~PCWrite,~IF_ID_Write,IF_ID_FLUSH,ID_EX_FLUSH,2'b0,forwardA,2'b0,
 forwardB,1'b0,EX_MEM_RegWrite,EX_MEM_MemtoReg,2'b0,EX_MEM_MemRead,
 EX_MEM_MemWrite,2'b0,EX_MEM_JUMP,EX_MEM_Branch,4'b0,ID_EX_ALUOp,2'b0};

  //MEM/WB
assign yw = MEM_WB_ALU_result;
assign mdr = writeData;
assign rdw = MEM_WB_rs1;
assign ctrlw =
{~PCWrite,~IF_ID_Write,IF_ID_FLUSH,ID_EX_FLUSH,2'b0,forwardA,2'b0,
 forwardB,1'b0,MEM_WB_RegWrite,MEM_WB_MemtoReg,2'b0,EX_MEM_MemRead,
 EX_MEM_MemWrite,2'b0,EX_MEM_JUMP,EX_MEM_Branch,4'b0,ID_EX_ALUOp,2'b0};

endmodule
```

2.PDU（外设）

流水线处理器调试单元

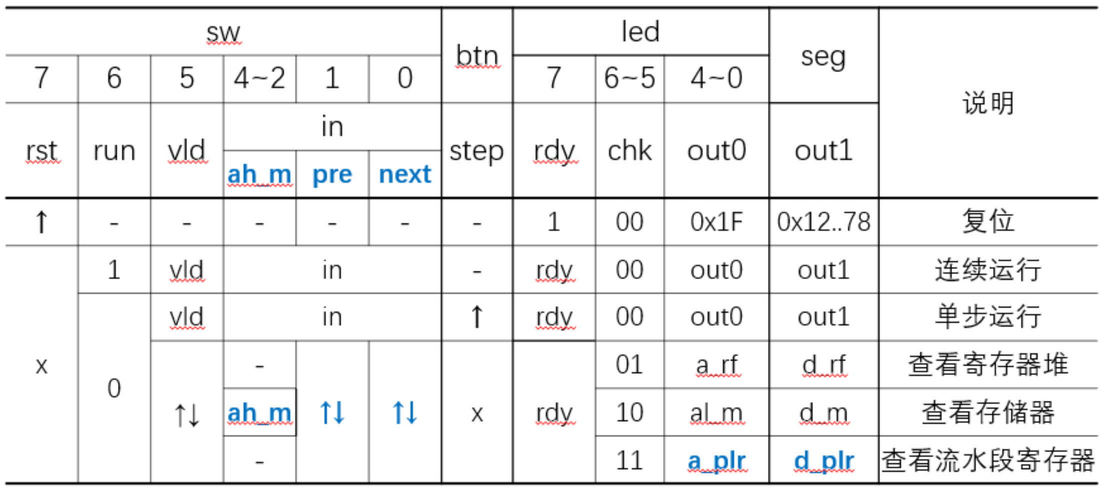
- **PDU_PL (Processor Debug Unit for Pipe-Line)**
  - 控制CPU的运行方式：**run = 1** 连续运行，**0** 单步运行
  - 管理外设 (开关sw、指示灯led、数码管seg)，显示CPU运行结果和数据通路状态



外设使用说明：

外设使用说明表

- 利用**vld**和**in**选择需要查看的信息，**chk**指示**out0**和**out1**上显示信息的类型



sw						btn	led			seg	说明
7	6	5	4~2	1	0		7	6~5	4~0		
rst	run	vld	in			step	rdy	chk	out0	out1	
			ah_m	**pre**	**next**						
↑	-	-	-	-	-	-	1	00	0x1F	0x12..78	复位
x	1	vld	in			-	rdy	00	out0	out1	连续运行
	0	vld	in			↑	rdy	00	out0	out1	单步运行
		↑↓	-	↑↓	↑↓	x	rdy	01	a_rf	d_rf	查看寄存器堆
			ah_m					10	al_m	d_m	查看存储器
			-					11	**a_plr**	**d_plr**	查看流水段寄存器

3.TOP（CPU与PDU的连接）

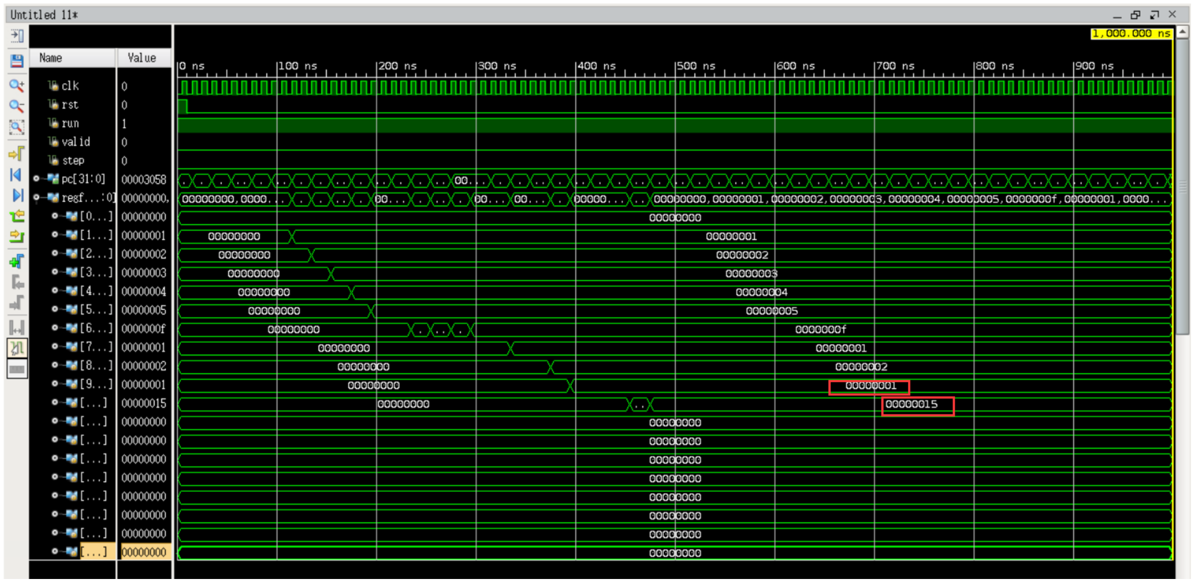
```verilog
module top(
input clk,rst,run,step,valid,
input [4:0] in,
output [1:0] check,
output [4:0] out0,
output [2:0] an,
output [3:0] seg,
output ready
    );

wire clk_cpu,io_we;
wire [7:0] io_addr,m_rf_addr;
wire [31:0] io_dout,io_din,rf_data,m_data,pc,pcd,ir,pcin,pce,a,b,imm,ctrl,y,bm,ctrlm,yw,mdr,ctrlw;
wire [4:0] rd,rdm,rdw;

pdu_pl pdu1(.clk(clk),.rst(rst),.run(run),.step(step),.clk_cpu(clk_cpu),.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),.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_pl cpu1(.clk(clk_cpu),.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),.pcd(pcd),.ir(ir),.pcin(pcin),
 .pce(pce),.a(a),.b(b),.IMM(imm),.rd(rd),.ctrl(ctrl),.y(y),.bm(bm),.rdm(rdm),.ctrlm(ctrlm),
 .yw(yw),.mdr(mdr),.rdw(rdw),.ctrlw(ctrlw));
endmodule
```

4.仿真

（1）hazard_test

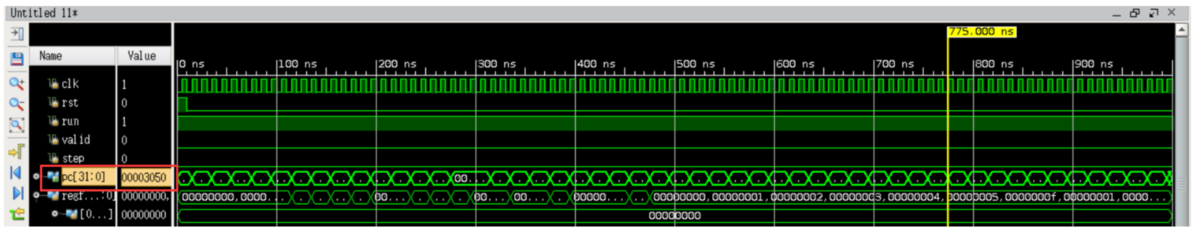
①**in = 1**，因此可以看到寄存器**x10**里出现**0x15**（1+20）

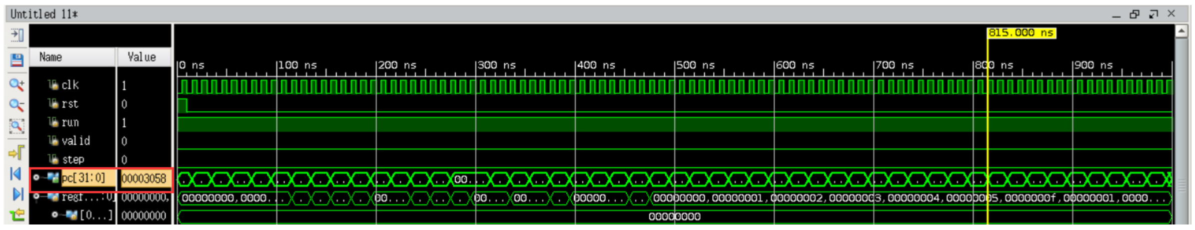
寄存器x1~x10的值都可由图中读出

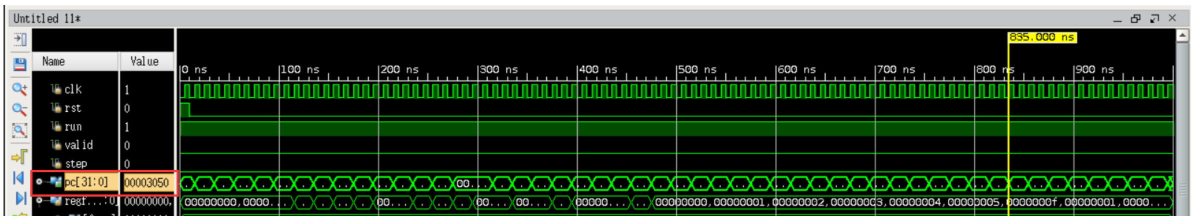


②也可以看到，最后CPU陷入循环（**stop: jal x0, stop**）

pc值从3050到3058的循环

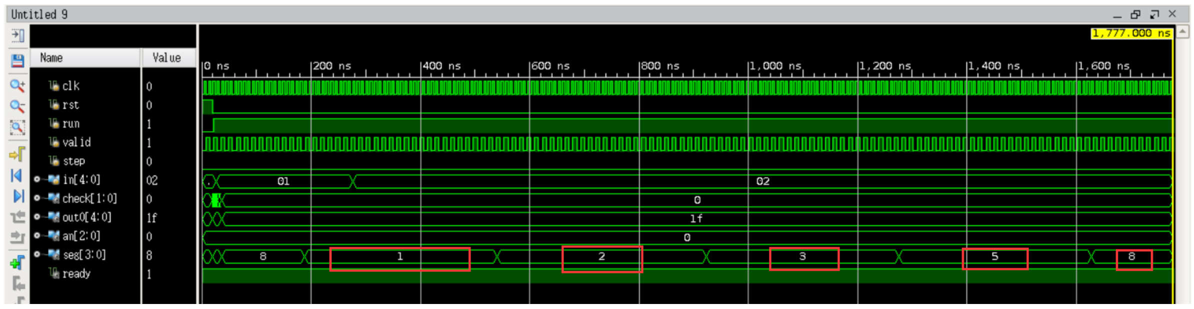






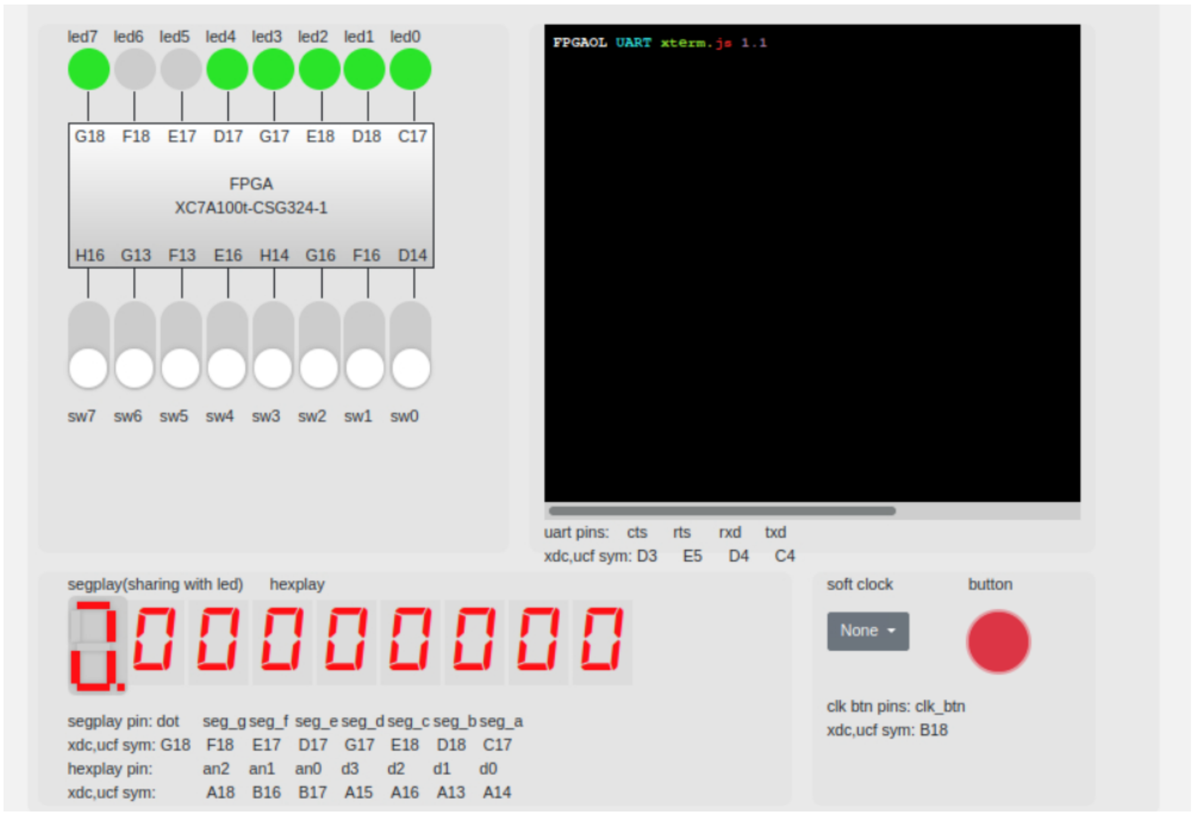
（2）fib test

输入前两个值为1和2，后续输出为3 , 5 , 8......



5.上板

fpgaol平台测试hazard_test


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
segplay(sharing with led)
hexplay
soft clock
None
button

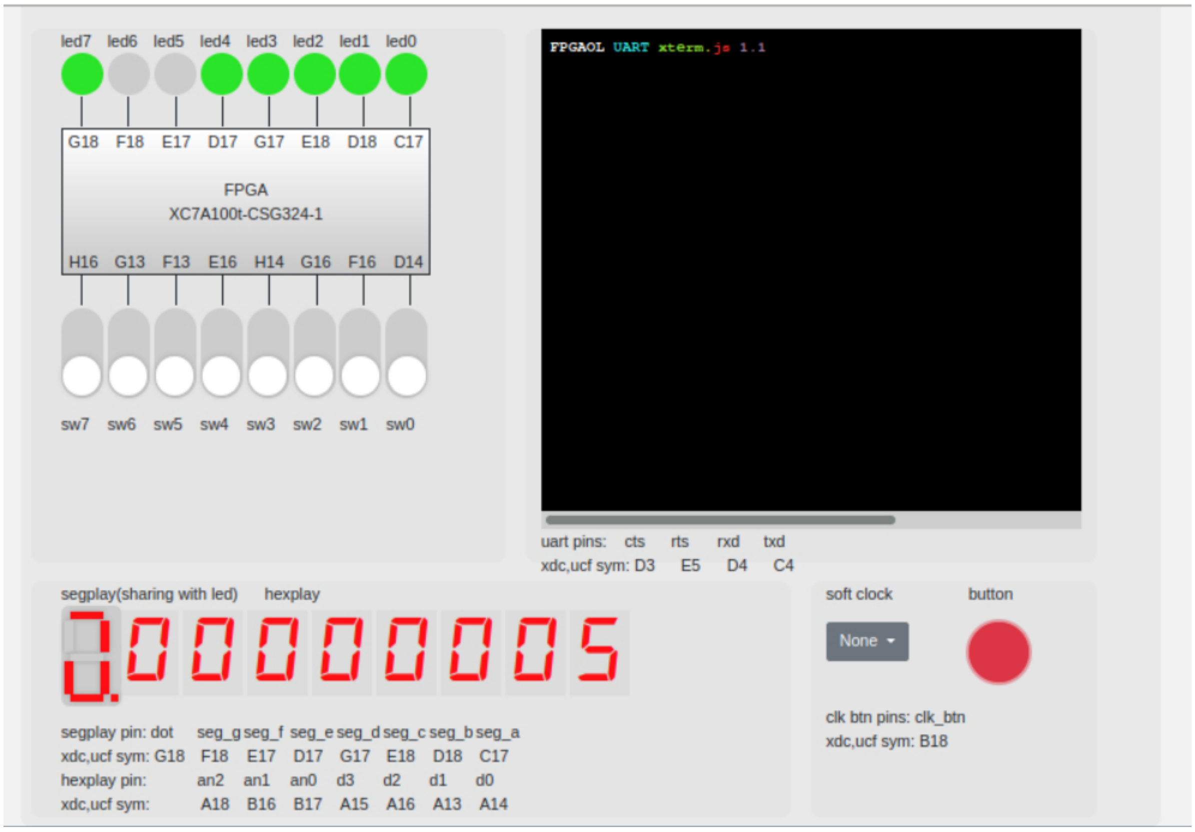
```
FPGAOL UART xterm.js 1.1
```

uart pins: cts rts rxd txd
xdc,ucf sym: D3 E5 D4 C4

00000000

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

clk btn pins: clk_btn
xdc,ucf sym: B18


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
segplay(sharing with led)
hexplay
soft clock
None
button

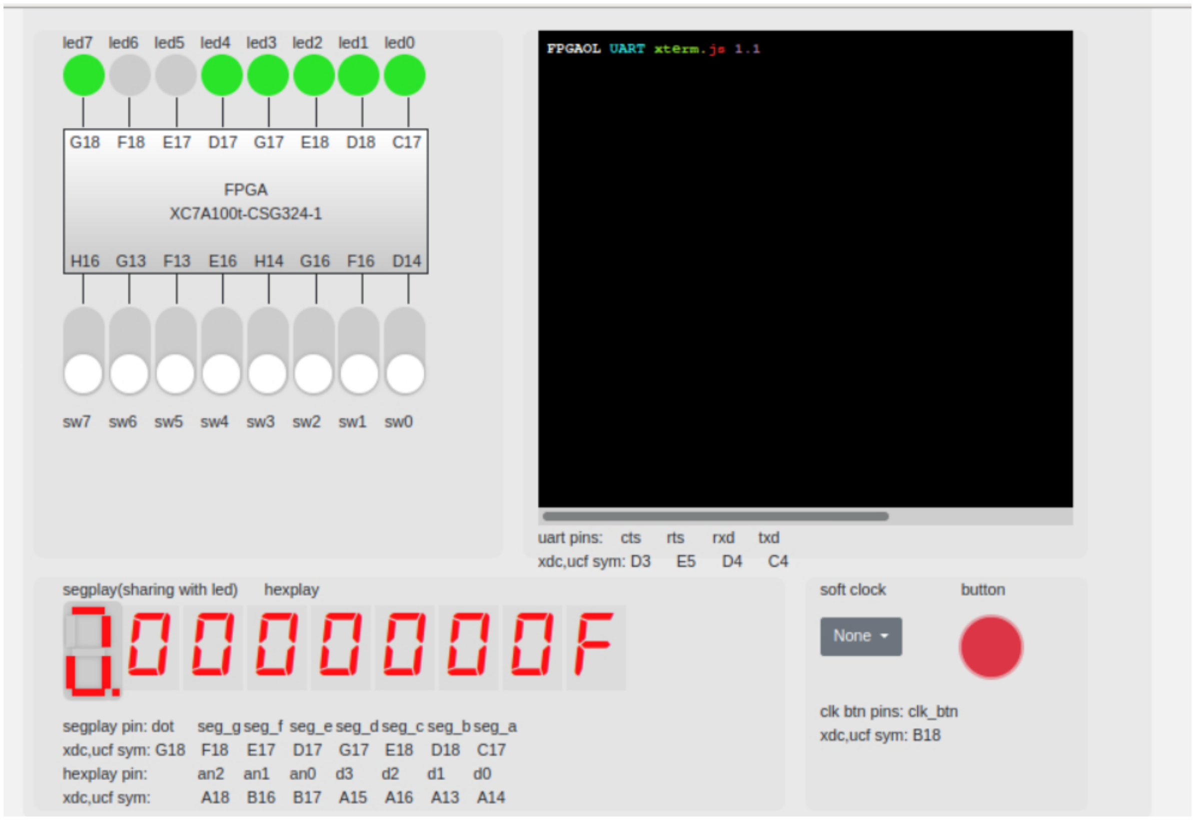
```
FPGAOL UART xterm.js 1.1
```

uart pins: cts rts rxd txd
xdc,ucf sym: D3 E5 D4 C4

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

clk btn pins: clk_btn
xdc,ucf sym: B18

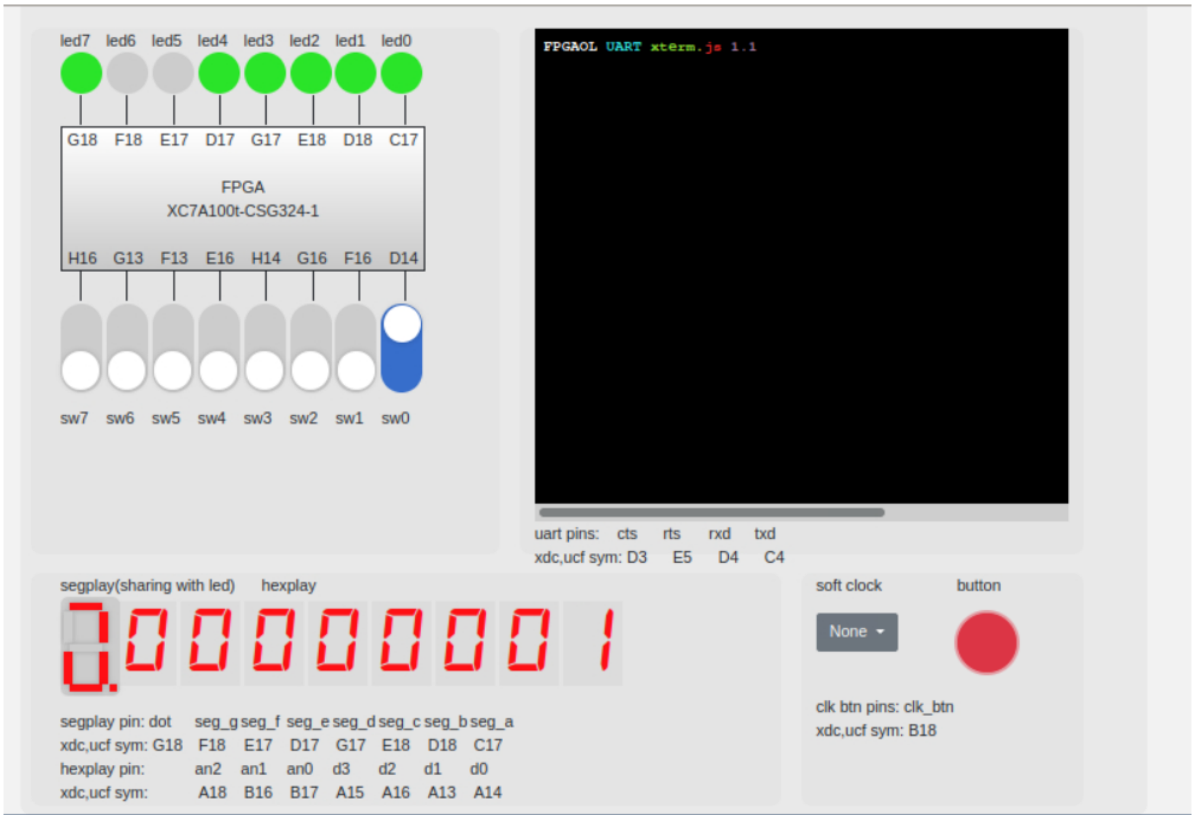

led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
FPGAOL UART xterm.js 1.1
segplay(sharing with led)
hexplay
0000000F
soft clock
button
None

uart pins: cts rts rxd txd
xdc,ucf sym: D3 E5 D4 C4

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

clk btn pins: clk_btn
xdc,ucf sym: B18

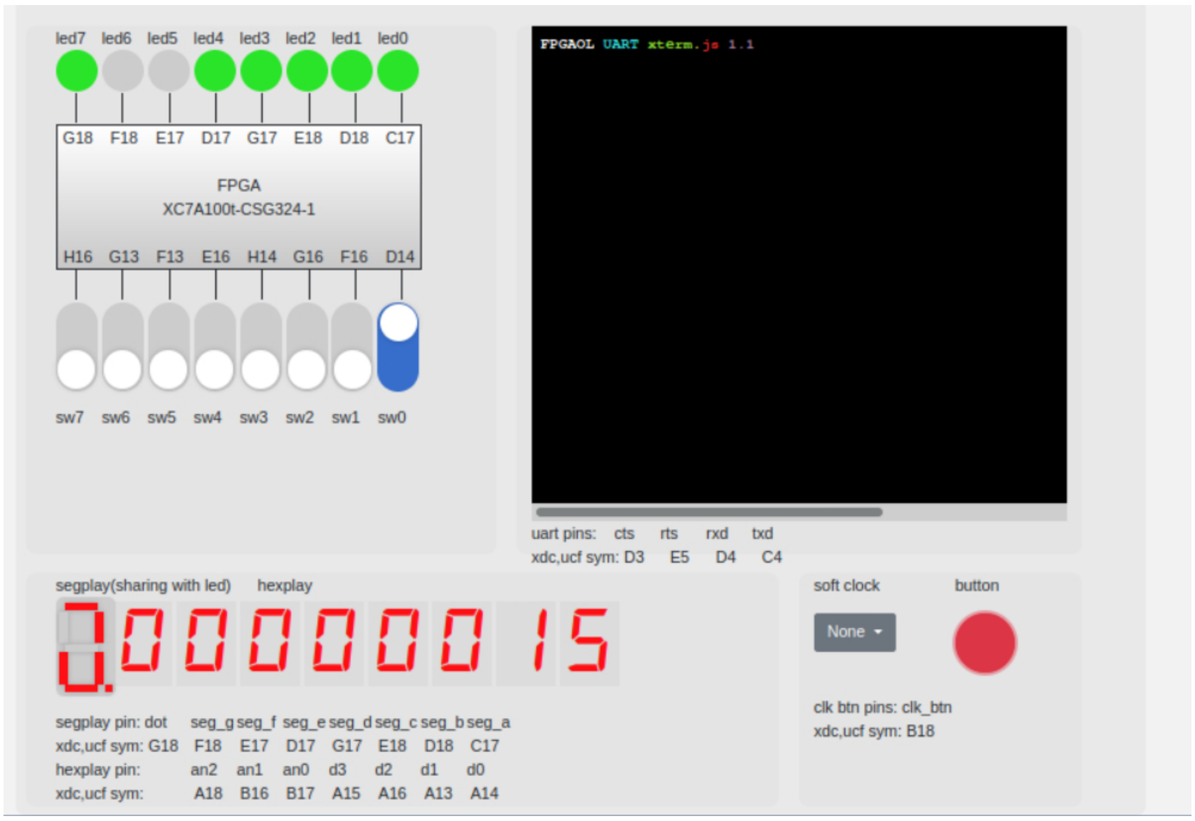
输入 in = 1 后:


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
segplay(sharing with led)
hexplay
00000001
soft clock
button
None

uart pins: cts rts rxd txd
xdc,ucf sym: D3 E5 D4 C4

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

clk btn pins: clk_btn
xdc,ucf sym: B18


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
segplay(sharing with led)
hexplay
soft clock
None
button

uart pins: cts rts rxd txd
xdc,ucf sym: D3 E5 D4 C4

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

clk btn pins: clk_btn
xdc,ucf sym: B18