

# 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：刘恒远

学生学号：PB20111642

完成日期：2022.4.28

## 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

## 实验环境：

vivado, FPGAOOL

## 实验过程：

### 题目一：

寄存器堆添加调试端口：

```
module reg_file(  
    input clk,  
    input [4:0] a1,a2,a3,  
    input [7:0] m_rf_addr,  
    input [31:0] wd3,  
    input we3,  
    output [31:0] rd1,rd2,  
    output [31:0] rf_data    //添加的读端口  
);  
reg [31:0] reg_file[0:31];  
integer i;  
initial  
begin  
    for(i=0;i<32;i=i+1) reg_file[i]=0;  
end  
always@(posedge clk)  
begin  
    if(we3&&(a3))  
        reg_file[a3]<=wd3;  
end  
assign rd1 = a1 ? reg_file[a1] : 32'h0;
```

```

assign rd2 = a2 ? reg_file[a2] : 32'h0;
assign rf_data = m_rf_addr[4:0] ? reg_file[m_rf_addr[4:0]] : 32'h0;
endmodule

```

## 题目二:

设计数据通路，并独立构建每一部分。

寄存器堆:

```

module reg_file(
input clk,
input [4:0]a1,a2,a3,
input [31:0]wd3,
input we3,
output [31:0]rd1,rd2,x18,x19,x20,x21,x22,x23
);
reg [31:0] reg_file[0:31];
integer i;
initial
begin
    for(i=0;i<32;i=i+1) reg_file[i]=0;
end
always@(posedge clk)
begin
    if(we3&&(a3))
        reg_file[a3]<=wd3;
end
assign rd1 = a1 ? reg_file[a1] : 32'h0;
assign rd2 = a2 ? reg_file[a2] : 32'h0;
assign x18 = reg_file[18];
assign x19 = reg_file[19];
assign x20 = reg_file[20];
assign x21 = reg_file[21];
assign x22 = reg_file[22];
assign x23 = reg_file[23];
endmodule

```

ALU:

```

module alu(
input [31:0]a,b,
input [1:0] alu_ctrl,          //0是清零, 1是加, 2是减
output reg [31:0] alu_out
);
/*wire signed [31:0] signed_a ;
wire signed [31:0] signed_b ;*/ //用不到, 这里计算不用补码

always@(*)
begin
    case(alu_ctrl)
        2'h0: alu_out=32'h0;
        2'h1: alu_out=a+b;
        2'h2: alu_out=a-b;
        default: alu_out =32'h0;
    endcase
end

```

```

    endcase
end
endmodule

```

PC:

```

module pc_calc(
input clk,
input rst,
input branch,
input [31:0] alu_out,
output reg [31:0] pc,
output [31:0] pc_plus4
);
always@(posedge clk or posedge rst)
begin
    if(rst)
        pc<=32'h00003000;
    else if(branch)
        pc<=alu_out;
    else
        pc<=pc_plus4;
end
assign pc_plus4=pc+32'h4;
endmodule

```

Branch:

```

module bran(
input [31:0] a,
input [31:0] b,
input [2:0] comp_ctrl,      //0是beq, 4是blt
input do_branch,
input do_jump,
output branch
);
wire signed [31:0] signed_a;
wire signed [31:0] signed_b;
reg taken;                  //是1且do_branch则跳转

assign signed_a=a;
assign signed_b=b;

always@(*)
begin
    case(comp_ctrl)
        3'h0: taken = (signed_a==signed_b);
        3'h4: taken = (signed_a<signed_b);
        default: taken = 0;
    endcase
end

assign branch = (taken&&do_branch) || do_jump; //b型或jalr, jal都需跳转
endmodule

```

controller:

```

module controller(
input  [31:0] inst,
output rf_wr_en,
output alu_a_sel,
output alu_b_sel,
output reg [1:0] alu_ctrl,
output reg [2:0] dm_rd_ctrl,
output reg [1:0] dm_wr_ctrl,
output reg [1:0] rf_wr_sel,
output [2:0] comp_ctrl,
output do_branch,
output do_jump
);
wire [6:0] opcode;
wire [2:0] funct3;
wire [6:0] funct7;
assign opcode = inst[6:0];
assign funct3 = inst[14:12];
assign funct7 = inst[31:25];

wire is_add;
wire is_addi;
wire is_sub;
wire is_auipc;
wire is_lw;
wire is_sw;
wire is_beq;
wire is_blt;
wire is_jal;
wire is_jalr;

assign is_add = (opcode==7'h33)&&(funct3==3'h0)&&(funct7==7'h00);
assign is_addi = (opcode==7'h13)&&(funct3==3'h0);
assign is_sub = (opcode==7'h33)&&(funct3==3'h0)&&(funct7==7'h20);
assign is_auipc = (opcode==7'h17);
assign is_lw = (opcode==7'h03)&&(funct3==3'h2);
assign is_sw = (opcode==7'h23)&&(funct3==3'h2);
assign is_beq = (opcode==7'h63)&&(funct3==3'h0);
assign is_blt = (opcode==7'h63)&&(funct3==3'h4);
assign is_jal = (opcode==7'h6f);
assign is_jalr = (opcode==7'h67)&&(funct3==3'h0);

wire is_add_type;
wire is_u_type;
wire is_jump_type;
wire is_b_type;
wire is_r_type;
wire is_i_type;
wire is_s_type;

assign is_b_type = is_beq | is_blt;
assign is_r_type = is_add | is_sub;
assign is_i_type = is_jalr | is_lw | is_addi;
assign is_s_type = is_sw;
assign is_u_type = is_auipc;
assign is_jump_type = is_jal;

```

```

assign is_add_type = is_auipc | is_add | is_addi | is_jal | is_jalr | is_b_type
| is_s_type | is_lw;    //除了sub指令alu全用加

always@(*)
begin
    if(is_add_type) alu_ctrl=2'h1;
    else if(is_sub) alu_ctrl=2'h2;
    else alu_ctrl=2'h0;
end

assign rf_wr_en = is_u_type | is_jump_type | is_r_type | is_i_type;
assign alu_a_sel = is_r_type | is_i_type | is_s_type;
assign alu_b_sel = ~ is_r_type;

always@(*)
begin
    if(is_lw) dm_rd_ctrl=3'h5;
    else dm_rd_ctrl=3'h0;
end

always@(*)
begin
    if(is_sw) dm_wr_ctrl=2'h3;
    else dm_wr_ctrl=2'h0;
end

always@(*)
begin
    if(is_lw) rf_wr_sel=2'h3;
    else if(((~is_jalr)&is_i_type) | is_u_type | is_r_type) rf_wr_sel=2'h2;
    else if(is_jalr | is_jal) rf_wr_sel=2'h1;
    else rf_wr_sel=2'h0;
end

assign comp_ctrl = funct3;
assign do_branch = is_b_type;
assign do_jump = is_jal | is_jalr;
endmodule

```

立即数imm:

```

module imm(
input [31:0] inst,
output reg [31:0] imm_out
);
wire [6:0] inst_type;
assign inst_type = inst[6:0];

always@(*)
begin
    case(inst_type)
        7'h13 : imm_out={{21{inst[31]}},inst[30:20]};//addi
        7'h17 : imm_out={inst[31:12],12'h0};          //auipc
        7'h03 : imm_out={{21{inst[31]}},inst[30:20]};//lw
        7'h23 : imm_out={{21{inst[31]}},inst[30:25],inst[11:7]};//sw
        7'h63 : imm_out=
{{20{inst[31]}},inst[7],inst[30:25],inst[11:8],1'b0};//beq,blt
        7'h6f : imm_out={{12{inst[31]}},inst[19:12],inst[20],inst[30:21],1'b0};//jal
    endcase
end

```

```

7'h67 : imm_out={{21{inst[31]}},inst[30:20]};//jalr
default : imm_out = 32'h0;
endcase
end
endmodule

```

memory (包括指令和数据两部分) :

```

module mem(
input clk,
input [31:0] im_addr, //就是pc
output [31:0] im_dout, //就是inst
input [2:0] dm_rd_ctrl,
input [1:0] dm_wr_ctrl,
input [31:0] dm_addr, //就是alu_out, 比如sw计算要修改的数据位置
input [31:0] dm_din, //就是rf_rd2
input [7:0] m_rf_addr,
output reg [31:0] dm_dout, //eg, 读取数据段给寄存器
output [31:0] m_data
);
wire [31:0] im_addr_use;
assign im_addr_use=(im_addr-32'h00003000)/4; //就比如把3000的指令位置变成0, 方便下面调用ram
dist_mem_inst mem_inst(im_addr_use[7:0], 32'h0, clk, 1'b0, im_dout); //得到im_dout, 且im_mem不会改, 所以we=0;

wire [31:0] dm_addr_use;
wire [31:0] dm_out;

assign dm_addr_use=dm_addr/4;
dist_mem_data
mem_data(dm_addr_use[7:0], dm_din, m_rf_addr, clk, dm_wr_ctrl, dm_out, m_data);

always@(*)
begin
case(dm_rd_ctrl)
3'h5: dm_dout<=dm_out;
default: dm_dout<=32'h0;
endcase
end
endmodule

```

构建完毕后, 建立CPU将所有模块连接起来:

```

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)或寄存器堆(RF)的调试读口地址

```

```

    output [31:0] m_data,      //从MEM读取的数据
    output [31:0] x18,x19,x20,x21,x22,x23,
    output [31:0] pc          //PC的内容

);
wire branch;
wire [31:0] pc_plus4;
wire [31:0] inst;
wire [31:0] imm_out;
wire rf_wr_en;
wire alu_a_sel;
wire alu_b_sel;
wire [1:0] alu_ctrl;
wire [2:0] dm_rd_ctrl;
wire [1:0] dm_wr_ctrl;
wire [1:0] rf_wr_sel;
wire [2:0] comp_ctrl;
wire do_branch;
wire do_jump;

reg [31:0] rf_wd3;
wire [31:0] rf_rd1,rf_rd2;
wire [31:0] alu_a,alu_b,alu_out;
wire [31:0] dm_dout;

pc_calc pc_calc(clk,rst,branch,alu_out,pc,pc_plus4);

mem
mem(clk,pc,inst,dm_rd_ctrl,dm_wr_ctrl,alu_out,rf_rd2,m_rf_addr,dm_dout,m_data);

imm imm(inst,imm_out);

controller
controller(inst,rf_wr_en,alu_a_sel,alu_b_sel,alu_ctrl,dm_rd_ctrl,dm_wr_ctrl,rf_w
r_sel,comp_ctrl,do_branch,do_jump);

always@(*)
begin
    case(rf_wr_sel)
        2'b00: rf_wd3 = 32'h0;
        2'b01: rf_wd3 = pc_plus4;
        2'b10: rf_wd3 = alu_out; //eg, sw要修改的数据位置
        2'b11: rf_wd3 = dm_dout; //eg, 读取数据段给寄存器
        default: rf_wd3 = 32'h0;
    endcase
end

reg_file
reg_file(clk,inst[19:15],inst[24:20],inst[11:7],rf_wd3,rf_wr_en,rf_rd1,rf_rd2,x1
8,x19,x20,x21,x22,x23);

assign alu_a = alu_a_sel ? rf_rd1 : pc;
assign alu_b = alu_b_sel ? imm_out : rf_rd2;

alu alu(alu_a,alu_b,alu_ctrl,alu_out);

bran bran(rf_rd1,rf_rd2,comp_ctrl,do_branch,do_jump,branch);
endmodule

```

测试指令是否能正常运行（测试指令如下）：

```
.data
x18data:.word 0x18
pc : .word 0x3000
.text
la a0,x18data #auipc测试
lw x18,0(a0)
sw x0,0(a0) #给x18赋值，并清空数据存储器

addi x19,x19,0x19 #x19值就是0x19
sub x20,x19,x18
blt x18,x19,L0

addi x20,x20,1
L0: addi x18,x18,1
beq x18,x19,L1

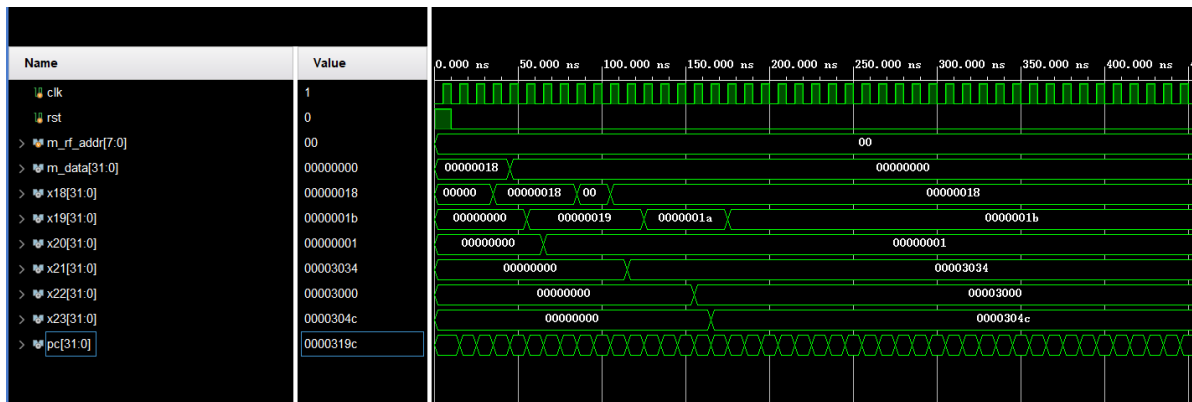
addi x20,x20,1
L1: sub x18,x18,x20

jal x21,L2
addi x20,x20,1
L2: addi x19,x19,1

la a0,pc
lw x22,0(a0)
jalr x23,0x50(x22) #jalr测试

addi x20,x20,1
addi x19,x19,1 #addi x20应该都不执行
```

导出coe文件后，将指令寄存器和数据寄存器初始化，仿真波形如图所示：



仿真结果与理论结果相同。

题目三：

已经给出fib数列的前两项，进行FPGAOL上板测试。

寄存器堆：

```
module reg_file(
```



```

input clk,
input [4:0] a1,a2,a3,
input [7:0] m_rf_addr,
input [31:0] wd3,
input we3,
output [31:0] rd1,rd2,
output [31:0] rf_data
);
reg [31:0] reg_file[0:31];
integer i;
initial
begin
    for(i=0;i<32;i=i+1) reg_file[i]=0;
end
always@(posedge clk)
begin
    if(we3&&(a3))
        reg_file[a3]<=wd3;
end
assign rd1 = a1 ? reg_file[a1] : 32'h0;
assign rd2 = a2 ? reg_file[a2] : 32'h0;
assign rf_data = m_rf_addr[4:0] ? reg_file[m_rf_addr[4:0]] : 32'h0;
endmodule

```

注意CPU中将IO\_BUS,Debug\_BUS全部启用。

PDU:

```

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

```

```

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

```

```

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

```

构建顶层文件core将CPU与PDU联系起来:

```

module core(
    input clk,
    input rst,

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

    //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
);
wire clk_cpu;
//IO_BUS
wire [7:0] io_addr;
wire [31:0] io_dout;
wire io_we;
wire [31:0] io_din;
//Debug_BUS
wire [7:0] m_rf_addr; //存储器(MEM)或寄存器堆(RF)的调试读口地址
wire [31:0] rf_data; //从RF读取的数据
wire [31:0] m_data; //从MEM读取的数据
wire [31:0] pc; //PC的内容

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

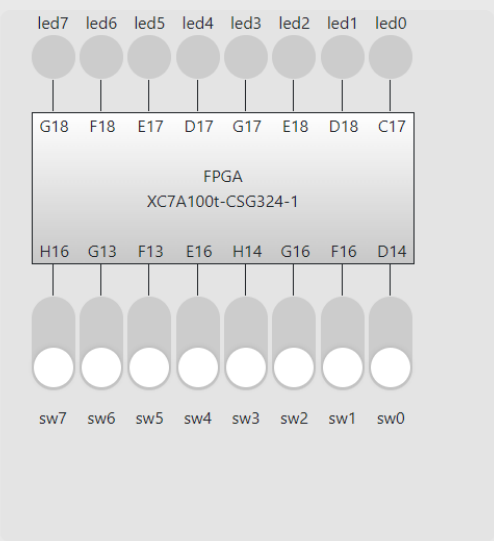
```

测试指令如下：

```
.text
addi t0,t0,1
sw t0,0x408(x0)
addi t1,t1,5
sw t1,0x408(x0)
loop:
add t2,t0,t1
sw t2,0x408(x0)
add t0,t1,x0
add t1,t2,x0
jal loop
```

上板测试结果如图所示：

**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

uart pins: cts rts rxd txd

segplay(sharing with led) hexplay

1 2 3 4 5 6 7 8

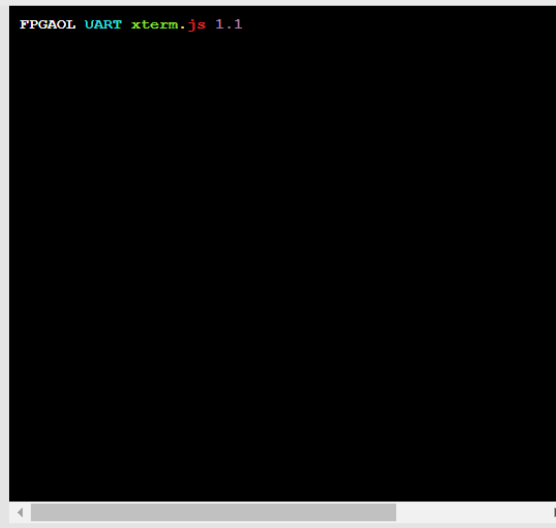
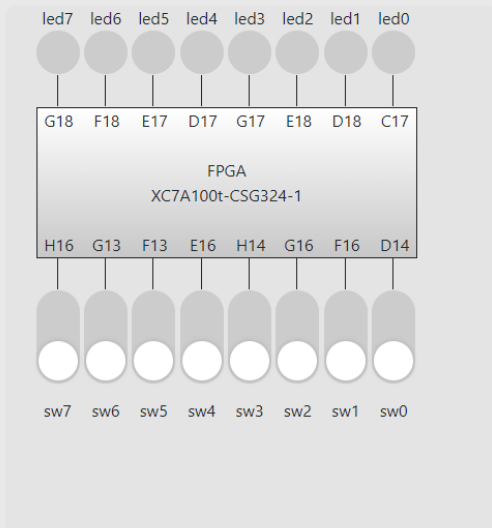
segplay pin: dot seg\_g seg\_f seg\_e seg\_d seg\_c seg\_b seg\_a  
xdc,ucf sym: G18 F18 E17 D17 G17 E18 D18 C17  
hexplay pin: an2 an1 an0 d3 d2 d1 d0  
xdc,ucf sym: A18 B16 B17 A15 A16 A13 A14

soft clock button

None

clk btn pins: clk\_btn  
xdc,ucf sym: B18

## FPGA interface



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
hexplay pin:	an2	an1	an0	d3	d2	d1	d0	
xdc,ucf sym:	A18	B16	B17	A15	A16	A13	A14	

soft clock

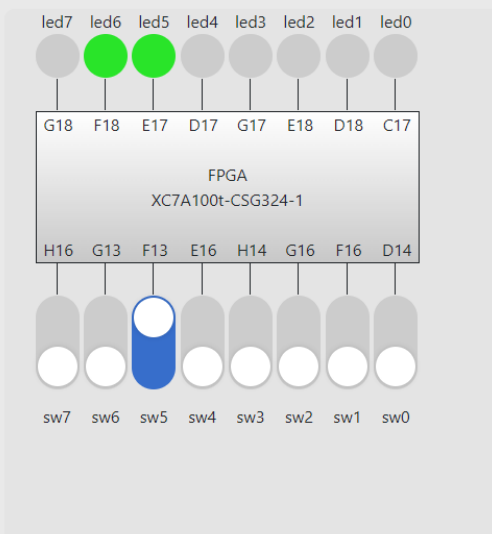
button

None



clk btn pins: clk\_btn  
xdc,ucf sym: B18

## FPGA interface



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
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



测试成功。

### 选做：

对于通过IO信号输入计算fib数列，

PDU：

```
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, //I/O外设的地址
    input [31:0] io_dout, //CPU向led和seg输出的数据
    input io_we, //CPU向led和seg输出时的使能信号,
```

```

        //利用该信号将io_dout存入输出缓冲寄存器（OBR），
        //再经数码管显示电路将其显示在数码管（an，seg）
output [31:0] io_din, //CPU接收来自输入缓冲寄存器（IBR）的sw输入数据

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

```

CPU:

```

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)或寄存器堆(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
    output [31:0] pc             //PC的内容
);
wire branch;
wire [31:0] pc_plus4;
wire [31:0] inst;
wire [31:0] imm_out;
wire rf_wr_en;
wire alu_a_sel;
wire alu_b_sel;
wire [1:0] alu_ctrl;
wire [2:0] dm_rd_ctrl;
wire [1:0] dm_wr_ctrl;
wire [1:0] rf_wr_sel;
wire [2:0] comp_ctrl;
wire do_branch;
wire do_jump;

```

```

reg [31:0] rf_wd3;
wire [31:0] rf_rd1, rf_rd2;
wire [31:0] alu_a, alu_b, alu_out;
wire [31:0] dm_dout;

pc_calc pc_calc(clk, rst, branch, alu_out, pc, pc_plus4);

mem
mem(clk, pc, inst, dm_rd_ctrl, dm_wr_ctrl, alu_out, rf_rd2, m_rf_addr, dm_dout, m_data, io
_addr, io_dout, io_we, io_din);

imm imm(inst, imm_out);

controller
controller(inst, rf_wr_en, alu_a_sel, alu_b_sel, alu_ctrl, dm_rd_ctrl, dm_wr_ctrl, rf_w
r_sel, comp_ctrl, do_branch, do_jump);

always@(*)
begin
    case(rf_wr_sel)
        2'b00: rf_wd3 = 32'h0;
        2'b01: rf_wd3 = pc_plus4;
        2'b10: rf_wd3 = alu_out;
        2'b11: rf_wd3 = dm_dout;
        default: rf_wd3 = 32'h0;
    endcase
end

reg_file
reg_file(clk, inst[19:15], inst[24:20], inst[11:7], m_rf_addr, rf_wd3, rf_wr_en, rf_rd1
, rf_rd2, rf_data);

assign alu_a = alu_a_sel ? rf_rd1 : pc;
assign alu_b = alu_b_sel ? imm_out : rf_rd2;

alu alu(alu_a, alu_b, alu_ctrl, alu_out);

bran bran(rf_rd1, rf_rd2, comp_ctrl, do_branch, do_jump, branch);
endmodule

module reg_file(
input clk,
input [4:0] a1, a2, a3,
input [7:0] m_rf_addr,
input [31:0] wd3,
input we3,
output [31:0] rd1, rd2,
output [31:0] rf_data
);
reg [31:0] reg_file[0:31];
integer i;
initial
begin
    for(i=0; i<32; i=i+1) reg_file[i]=0;
end
always@(posedge clk)
begin
    if(we3&&(a3))

```

```

        reg_file[a3] <= wd3;
    end
    assign rd1 = a1 ? reg_file[a1] : 32'h0;
    assign rd2 = a2 ? reg_file[a2] : 32'h0;
    assign rf_data = m_rf_addr[4:0] ? reg_file[m_rf_addr[4:0]] : 32'h0;
endmodule

module alu(
    input [31:0] a,b,
    input [1:0] alu_ctrl,          //0是清零, 1是加, 2是减
    output reg [31:0] alu_out
);
/*wire signed [31:0] signed_a ;
wire signed [31:0] signed_b ;*/ //用不到, 这里计算不用补码

always@(*)
begin
    case(alu_ctrl)
        2'h0: alu_out=32'h0;
        2'h1: alu_out=a+b;
        2'h2: alu_out=a-b;
        default: alu_out =32'h0;
    endcase
end
endmodule

module pc_calc(
    input clk,
    input rst,
    input branch,
    input [31:0] alu_out,
    output reg [31:0] pc,
    output [31:0] pc_plus4
);
always@(posedge clk or posedge rst)
begin
    if(rst)
        pc<=32'h00003000;
    else if(branch)
        pc<=alu_out;
    else
        pc<=pc_plus4;
    end
    assign pc_plus4=pc+32'h4;
endmodule

module bran(
    input [31:0] a,
    input [31:0] b,
    input [2:0] comp_ctrl,        //0是beq, 4是blt
    input do_branch,
    input do_jump,
    output branch
);
wire signed [31:0] signed_a;
wire signed [31:0] signed_b;
reg taken;                      //是1且do_branch则跳转

```

```

assign signed_a=a;
assign signed_b=b;

always@(*)
begin
    case(comp_ctrl)
        3'h0: taken = (signed_a==signed_b);
        3'h4: taken = (signed_a<signed_b);
        default: taken = 0;
    endcase
end

assign branch = (taken&&do_branch) || do_jump;//b型或jalr, jal都需跳转
endmodule

module controller(
input  [31:0]inst,
output rf_wr_en,
output alu_a_sel,
output alu_b_sel,
output reg [1:0]alu_ctrl,
output reg [2:0]dm_rd_ctrl,
output reg [1:0]dm_wr_ctrl,
output reg [1:0]rf_wr_sel,
output [2:0] comp_ctrl,
output do_branch,
output do_jump
);
wire [6:0] opcode;
wire [2:0] funct3;
wire [6:0] funct7;
assign opcode = inst[6:0];
assign funct3 = inst[14:12];
assign funct7 = inst[31:25];

wire is_add;
wire is_addi;
wire is_sub;
wire is_auipc;
wire is_lw;
wire is_sw;
wire is_beq;
wire is_blt;
wire is_jal;
wire is_jalr;

assign is_add = (opcode==7'h33)&&(funct3==3'h0)&&(funct7==7'h00);
assign is_addi = (opcode==7'h13)&&(funct3==3'h0);
assign is_sub = (opcode==7'h33)&&(funct3==3'h0)&&(funct7==7'h20);
assign is_auipc = (opcode==7'h17);
assign is_lw = (opcode==7'h03)&&(funct3==3'h2);
assign is_sw = (opcode==7'h23)&&(funct3==3'h2);
assign is_beq = (opcode==7'h63)&&(funct3==3'h0);
assign is_blt = (opcode==7'h63)&&(funct3==3'h4);
assign is_jal = (opcode==7'h6f);
assign is_jalr = (opcode==7'h67)&&(funct3==3'h0);

```

```

wire is_add_type;
wire is_u_type;
wire is_jump_type;
wire is_b_type;
wire is_r_type;
wire is_i_type;
wire is_s_type;

assign is_b_type = is_beq | is_blt;
assign is_r_type = is_add | is_sub;
assign is_i_type = is_jalr | is_lw | is_addi;
assign is_s_type = is_sw;
assign is_u_type = is_auipc;
assign is_jump_type = is_jal;
assign is_add_type = is_auipc | is_add | is_addi | is_jal | is_jalr | is_b_type
| is_s_type | is_lw;    //除了sub指令alu全用加

always@(*)
begin
    if(is_add_type) alu_ctrl=2'h1;
    else if(is_sub) alu_ctrl=2'h2;
    else alu_ctrl=2'h0;
end

assign rf_wr_en = is_u_type | is_jump_type | is_r_type | is_i_type;
assign alu_a_sel = is_r_type | is_i_type | is_s_type;
assign alu_b_sel = ~ is_r_type;

always@(*)
begin
    if(is_lw) dm_rd_ctrl=3'h5;
    else dm_rd_ctrl=3'h0;
end

always@(*)
begin
    if(is_sw) dm_wr_ctrl=2'h3;
    else dm_wr_ctrl=2'h0;
end

always@(*)
begin
    if(is_lw) rf_wr_sel=2'h3;
    else if(((~is_jalr)&is_i_type) | is_u_type | is_r_type) rf_wr_sel=2'h2;
    else if(is_jalr | is_jal) rf_wr_sel=2'h1;
    else rf_wr_sel=2'h0;
end

assign comp_ctrl = funct3;
assign do_branch = is_b_type;
assign do_jump = is_jal | is_jalr;
endmodule

module imm(
input [31:0] inst,
output reg [31:0] imm_out
);
wire [6:0] inst_type;
assign inst_type = inst[6:0];

```

```

always@(*)
begin
    case(inst_type)
        7'h13 : imm_out={{21{inst[31]}},inst[30:20]}; //addi
        7'h17 : imm_out={inst[31:12],12'h0}; //auipc
        7'h03 : imm_out={{21{inst[31]}},inst[30:20]}; //lw
        7'h23 : imm_out={{21{inst[31]}},inst[30:25],inst[11:7]}; //sw
        7'h63 : imm_out=
{{20{inst[31]}},inst[7],inst[30:25],inst[11:8],1'b0}; //beq,blt
        7'h6f : imm_out={{12{inst[31]}},inst[19:12],inst[20],inst[30:21],1'b0}; //jal
        7'h67 : imm_out={{21{inst[31]}},inst[30:20]}; //jalr
        default : imm_out = 32'h0;
    endcase
end

endmodule

module mem(
input clk,
input [31:0] im_addr, //就是pc
output [31:0] im_dout, //就是inst
input [2:0] dm_rd_ctrl,
input [1:0] dm_wr_ctrl,
input [31:0] dm_addr, //就是alu_out,比如sw计算要修改的数据位置
input [31:0] dm_din, //就是rf_rd2
input [7:0] m_rf_addr,
output reg [31:0] dm_dout, //eg, 读取数据段给寄存器
output [31:0] m_data,
output [7:0] io_addr, //led和seg的地址
output reg [31:0] io_dout, //输出led和seg的数据
output reg io_we,
input [31:0] io_din
);
wire [31:0] im_addr_use;
assign im_addr_use=(im_addr-32'h00003000)/4; //就比如把3000的指令位置变成0, 方便下面调用
ram
dist_mem_inst mem_inst(im_addr_use[7:0],32'h0,clk,1'b0,im_dout); //得到im_dout, 且
im_mem不会改, 所以we=0;

wire [31:0] dm_addr_use;
wire [31:0] dm_out;
reg io_dm_wr_ctrl;
assign io_addr = dm_addr[7:0];
always@(*)
begin
    if(((dm_addr==32'h00000400)|| (dm_addr==32'h00000404)||
(dm_addr==32'h00000408))&&(dm_wr_ctrl[0]==1))
    begin
        io_dm_wr_ctrl <= 0;
        io_we <= 1;
        io_dout <= dm_din;
    end
    else
    begin
        io_dm_wr_ctrl <= dm_wr_ctrl[0];
        io_we <= 0;
        io_dout <= 32'h0;
    end
end

```

```

        end
    end
    assign dm_addr_use=dm_addr/4;
    dist_mem_data
    mem_data(dm_addr_use[7:0],dm_din,m_rf_addr,clk,io_dm_wr_ctrl,dm_out,m_data);

    always@(*)
    begin
        case(dm_rd_ctrl)
            3'h5:begin if(dm_addr==32'h0000040c || dm_addr==32'h00000410)
dm_dout<=io_din;          //is_lw, 才读数据
                else dm_dout<=dm_out;
            end
            default: dm_dout<=32'h0;
        endcase
    end
endmodule

```

顶层文件core:

```

module core(
    input clk,
    input rst,

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

    //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
);
wire clk_cpu;
//IO_BUS
wire [7:0] io_addr;
wire [31:0] io_dout;
wire io_we;
wire [31:0] io_din;
//Debug_BUS
wire [7:0] m_rf_addr;//存储器(MEM)或寄存器堆(RF)的调试读口地址
wire [31:0] rf_data;//从RF读取的数据
wire [31:0] m_data;//从MEM读取的数据
wire [31:0] pc;//PC的内容

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

```



测试指令如下:

```
.text
    addi x1, x0, 1      #x1=1
    add t1, x0, x0      #store fib series @t1

#### input f0
    sw x1, 0x404(x0)    #rdy=1
l1:
    lw t0, 0x410(x0)    #wait vld=1
    #   addi a7, x0, 5    #for debug begin
    #   ecall
    #   mv t0, a0        #for debug end

    beq t0, x0, l1
    lw s0, 0x40c(x0)    #s0=vin
    #   addi a7, x0, 5    #for debug begin
    #   ecall
    #   mv s0, a0        #for debug end

    sw s0, 0x408(x0)    #out1=f0
    sw s0, 0(t1)        #store f0
    addi t1, t1, 4

    sw x0, 0x404(x0)    #rdy=0
l2:
    lw t0, 0x410(x0)    #wait vld=0
    #   addi a7, x0, 5    #for debug begin
    #   ecall
    #   mv t0, a0        #for debug end

    beq t0, x1, l2

#### input f1
    sw x1, 0x404(x0)    #rdy=1
l3:
    lw t0, 0x410(x0)    #wait vld=1
    #   addi a7, x0, 5    #for debug begin
    #   ecall
    #   mv t0, a0        #for debug end

    beq t0, x0, l3
    lw s1, 0x40c(x0)    #s1=vin
    #   addi a7, x0, 5    #for debug begin
    #   ecall
    #   mv s1, a0        #for debug end

    sw s1, 0x408(x0)    #out1=f1
    sw s1, 0(t1)        #store f1
    addi t1, t1, 4

    sw x0, 0x404(x0)    #rdy=0
l4:
    lw t0, 0x410(x0)    #wait vld=0
    #   addi a7, x0, 5    #for debug begin
    #   ecall
    #   mv t0, a0        #for debug end
```

```

    beq t0, x1, 14

#### comput fi = fi-2 + fi-1
next:
    add t0, s0, s1    #fi
    sw t0, 0x408(x0)  #out1=fi
    sw t0, 0(t1)      #store fi
    addi t1, t1, 4

    add s0, x0, s1
    add s1, x0, t0

    sw x1, 0x404(x0)  #rdy=1
15:
    lw t0, 0x410(x0)  #wait vld=1
    # addi a7, x0, 5   #for debug begin
    # ecall
    # mv t0, a0        #for debug end

    beq t0, x0, 15
    sw x0, 0x404(x0)  #rdy=0
16:
    lw t0, 0x410(x0)  #wait vld=0
    # addi a7, x0, 5   #for debug begin
    # ecall
    # mv t0, a0        #for debug end

    beq t0, x1, 16
    jal x0, next

```

上板测试结果如图所示：

**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

uart pins: cts rts rxd txd

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

hexplay pin: an2 an1 an0 d3 d2 d1 d0

xdc,ucf sym: A18 B16 B17 A15 A16 A13 A14

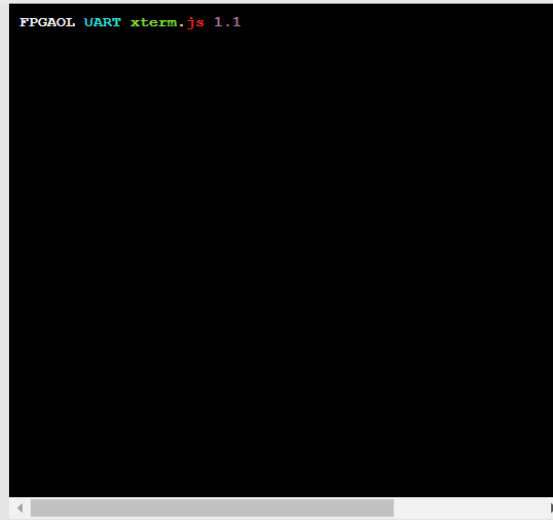
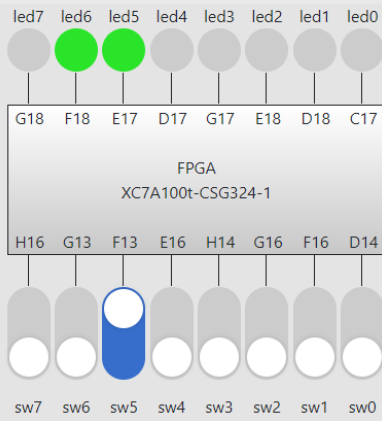
soft clock button

None

clk btn pins: clk\_btn

xdc,ucf sym: B18

## FPGA interface



uart pins: cts rts rxd txd

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
hexplay pin:		an2	an1	an0	d3	d2	d1	d0
xdc,ucf sym:		A18	B16	B17	A15	A16	A13	A14

soft clock

button

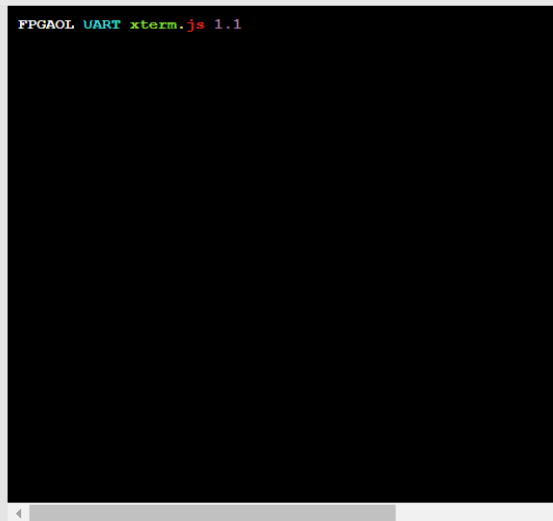
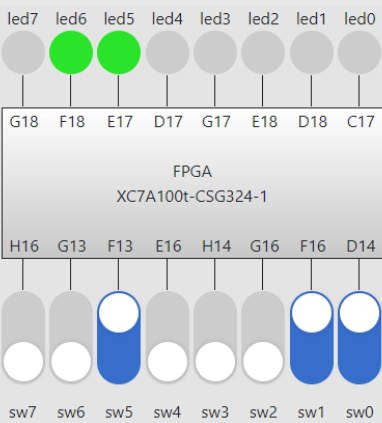
None



clk btn pins: clk\_btn

xdc,ucf sym: B18

## FPGA interface



uart pins: cts rts rxd txd

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
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

测试成功。

## 总结与思考：

本次实验对于我而言还是比较困难的，希望可以更新一下PPT，不然不利于理解。时间可以稍微延长一点。