

实验十一单周期 CPU

姓名：张云策 学号：3200105787 学院：计算机科学与技术

课程名称：计算机系统 1 同组学生姓名：/

实验时间：2021.7.4 实验地点：紫金港东 4-509 指导老师：吴磊

一、实验目的和要求

Lab 10_1:

- 了解 CPU 设计原理
- 设计 CPU 数据通路
- 为之后搭起单周期 CPU 打下基础

Lab 10_2:

- 了解 CPU 设计的基本原理
- 设计 CPU 控制单元模块
- 结合上次实验中的数据通路模块，搭建单周期 CPU

二、实验内容和原理

2.1 实验内容

搭建单周期 CPU 的核心部件，并且按照不同指令进行数据通路搭建，同时设计一个控制模块进行模块调用。

2.2 设计模块

Datapath:

```
datapath : Datapath (Datapath.v) (5)
  regs : Regs (Regs.v)
  IG : ImmGen (ImmGen.v)
  alu : ALU (ALU.v)
  mux : MUX2T1_32 (MUX2T1_32.v)
  pc : PC (PC.v)
```

Regs: registers, 寄存器组

```
module Regs(  
    input clk,  
    input rst,  
    input we,  
    input [4:0] read_addr_1,  
    input [4:0] read_addr_2,  
    input [4:0] write_addr,  
    input [31:0] write_data,  
    output [31:0] read_data_1,  
    output [31:0] read_data_2  
);  
  
    integer i;  
    reg [31:0] register [1:31]; // x1 - x31, x0 keeps zero  
  
    assign read_data_1 = (read_addr_1 == 0) ? 0 : register[read_addr_1]; // read  
    assign read_data_2 = (read_addr_2 == 0) ? 0 : register[read_addr_2]; // read  
  
    always @(posedge clk or posedge rst) begin  
        if (rst == 1) for (i = 1; i < 32; i = i + 1) register[i] <= 0; // reset  
        else if (we == 1 && write_addr != 0) register[write_addr] <= write_data;  
    end  
  
endmodule
```

(按照实验手册编写)

IMMGEN:立即数扩展, 由于指令类型的不同, imm 取 inst 中的数值的位置也不同

```
module ImmGen(  
    input [31:0] inst,  
    output [31:0] imm  
);  
    reg [31:0] imm_1;  
    always@(*)  
    case(inst[6:0])  
        7'b0010011: imm_1 = {{20{inst[31]}}, inst[31:20]}; //addi_andi_slti_ori  
        7'b0000011: imm_1 = {{20{inst[31]}}, inst[31:20]}; //lw  
        7'b0100011: imm_1 = {{20{inst[31]}}, inst[31:25], inst[11:7]}; //sw  
        7'b0110111: imm_1 = {inst[31:12], 12'b0}; //lui  
        7'b1100011: imm_1 = {{20{inst[31]}}, inst[7], inst[30:25], inst[11:8], 1'b0}; //beq_bne  
        7'b1101111: imm_1 = {{12{inst[31]}}, inst[19:12], inst[20], inst[30:21], 1'b0}; //jal  
        default: imm_1 = 32'b0;  
    endcase  
    assign imm = imm_1;  
endmodule
```

ALU: 进行运算, 如加减左移右移等等, 最终结果会在 res 变量中保存, zero 变量类似 “ZF”, count 类似 “SF”

```
//
/////////////////////////////////////////////////////////////////

module ALU(
    input signed [31:0] a,
    input signed [31:0] b,
    input [3:0] alu_op,
    output [31:0] res,
    output zero,
    output count
);
    `include "AluOp.vh"
    reg [31:0] final;
    always @(*)
        case (alu_op)
            ADD: final <= a + b;
            SUB: final <= a - b;
            SLT: final <= (a < b)? 1:0;

            OR: final <= a | b;
            AND: final <= a & b;
            default: final = 0;
        endcase
    assign res = final;
    assign zero = (res==0)? 1:0;
    assign count = (res[31]==0)? 1:0;
endmodule
```

MUX2_1: 选择器，用于输入多个数据，而输出一个数据的器件，根据控制信号，输出不同的值（四选一，0~3）

```
//
/////////////////////////////////////////////////////////////////

module ALU(
    input signed [31:0] a,
    input signed [31:0] b,
    input [3:0] alu_op,
    output [31:0] res,
    output zero,
    output count
);
    `include "AluOp.vh"
    reg [31:0] final;
    always @(*)
        case (alu_op)
            ADD: final <= a + b;
            SUB: final <= a - b;
            SLT: final <= (a < b)? 1:0;

            OR: final <= a | b;
            AND: final <= a & b;
            default: final = 0;
        endcase
    assign res = final;
    assign zero = (res==0)? 1:0;
    assign count = (res[31]==0)? 1:0;
endmodule
```

PC:程序计数器，根据控制信号输入，分别将不同的地址进行输出，如 pc_src== 2 ‘b00 时，输出值即为源地址+4；

```

module PC(
    input clk,
    input rst,
    input [31:0] imm,
    input [31:0] pc_pre,
    input [1:0] pc_src,
    input branch, b_type, zero,
    output reg [31:0] pc_out
);
initial begin
    pc_out <= 0;
end
always @(posedge clk) begin
    if(rst) pc_out <= 0;
    else if (pc_src==2'b00) pc_out <= pc_pre+4;
    else if (pc_src==2'b10) begin
        if(branch==1)pc_out <= ((b_type^zero)==0)? pc_pre + imm:pc_pre + 4;
        else pc_out <= pc_pre + imm;
    end
end
endmodule

```

Datapath:该模块主要就是将上述模块以及环境连接，具体操作由不同的指令决定。

```

output [31:0] pc_out
);
reg [31:0] register [1:31];
wire ZF, CF;
wire [31:0] read_data_1;
wire [31:0] read_data_2;
wire [31:0] read_data_3;
wire [31:0] reg_write_data;
wire [31:0] pc_pre;
assign pc_pre = pc_out;
assign data_out = read_data_2;
Regs regs(
    .clk(clk),
    .rst(rst),
    .w(reg_write),
    .read_addr_1(inst_in[19:15]),
    .read_addr_2(inst_in[24:20]),
    .write_addr(inst_in[11:7]),
    .write_data(reg_write_data),
    .read_data_1(read_data_1),
    .read_data_2(read_data_2)
);
ImmGen IG(
    .inst(inst_in),
    .imm(read_data_3)
);
ALU alu(
    .a(read_data_1),
    .b((alu_src_b==1)? read_data_3:read_data_2),
    .alu_op(alu_op),
    .res(addr_out),
    .zero(ZF),
    .count(CF)
);
MUX2T1_32 mux(
    .I0(addr_out),
    .I1(read_data_3),
    .I2(pc_pre*4),
    .I3(data_in),
    .s(mem_to_reg),
    .o(reg_write_data)
);
PC pc(
    .clk(clk),
    .rst(rst),
    .imm(read_data_3),
    .pc_pre(pc_pre),
    .pc_src(pc_src),
    .branch(branch),
    .b_type(b_type),
    .zero(ZF),
    .pc_out(pc_out)
);
endmodule

```

Core.v:主要就是将 core 有 Ram, Rom 结合

```
assign rst = ~aresetn;
SCPU cpu(
    .clk(cpu_clk),
    .rst(rst),
    .inst(inst),
    .data_in(core_data_in), // data from data memory
    .addr_out(addr_out), // data memory address
    .data_out(core_data_out), // data to data memory
    .pc_out(pc_out), // connect to instruction memory
    .mem_write(mem_write)
);

always @(posedge clk) begin
    if(rst) clk_div <= 0;
    else clk_div <= clk_div + 1;
end
assign mem_clk = ~clk_div[0]; // 50mhz
assign cpu_clk = debug_mode ? clk_div[0] : step;

// TODO: 模拟Instruction Memory
Rom rom_unit (
    .a({00, pc_out[31:2]}),
    .spo(inst)
);

// TODO: 模拟Data Memory
Ram ram_unit (
    .clka(mem_clk), // 时钟
    .wea(mem_write), // 写使能
    .addra(addr_out), // 地址
    .dina(core_data_out), // 数据输入
    .douta(core_data_in) // 数据输出
);

// TODO: 模拟32位 寄存器
assign chip_debug_out0 = pc_out;
assign chip_debug_out1 = addr_out;
assign chip_debug_out2 = inst;
assign chip_debug_out3 = 32'hffffaaa;
endmodule
```

Control.v:

该模块为模拟 control 器件，按照不同指令的数据通路，输出不同的控制信号

```
`timescale 1 ps / 1 ps
module Control(
    input [8:0] op_code,
    input [2:0] funct3,
    input funct7_5,
    output reg [1:0] pc_src,
    output reg reg_write,
    output reg alu_src_b,
    output reg [3:0] alu_op,
    output reg [1:0] mem_to_reg,
    output reg mem_write,
    output reg branch,
    output reg b_type
);
    include "AluOp.vh"
    always @* begin
        pc_src = 0;
        reg_write = 0;
        alu_src_b = 0;
        alu_op = {1'b0, funct3};
        mem_to_reg = 0;
        mem_write = 0;
        branch = 0;
        b_type = 1'b0;
        case (op_code)
            7'b0101011: begin pc_src = 2'b00; reg_write = 1'b1; mem_to_reg = 2'b01; end // LUI
            7'b0100011: begin pc_src = 2'b00; reg_write = 1'b1; alu_src_b = 1'b1; mem_to_reg = 2'b00; end // addi_alu1
            7'b0101001: begin pc_src = 2'b00; reg_write = 1'b1; alu_src_b = 1'b0; mem_to_reg = 2'b00; end // s_type
            7'b0100001: begin pc_src = 2'b00; reg_write = 1'b1; alu_src_b = 1'b1; alu_op = 4'b0; mem_to_reg = 2'b11; mem_write = 1'b0; end // s
            7'b0100010: begin pc_src = 2'b00; reg_write = 1'b1; alu_src_b = 1'b1; alu_op = 4'b0; mem_write = 1'b1; end // sr
            7'b1010111: begin pc_src = 2'b00; reg_write = 1'b1; mem_to_reg = 2'b10; branch = 1'b0; end // jal
            7'b1000011: begin pc_src = 2'b00; reg_write = 1'b1; mem_to_reg = 2'b10; branch = 1'b0; end // jalr
            default: alu_op = 0;
        endcase
    end
endmodule
```

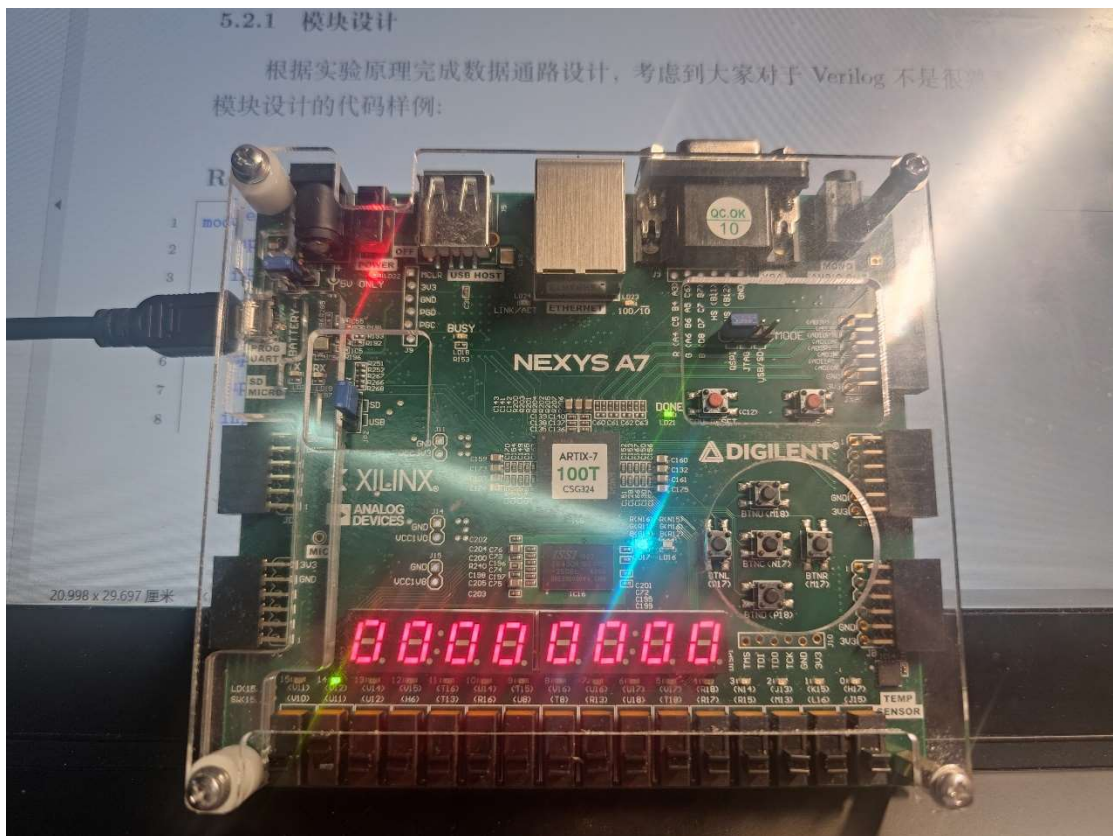
三、 主要仪器设备

Vivado 2017.4
NEXYS-A7 FPGA

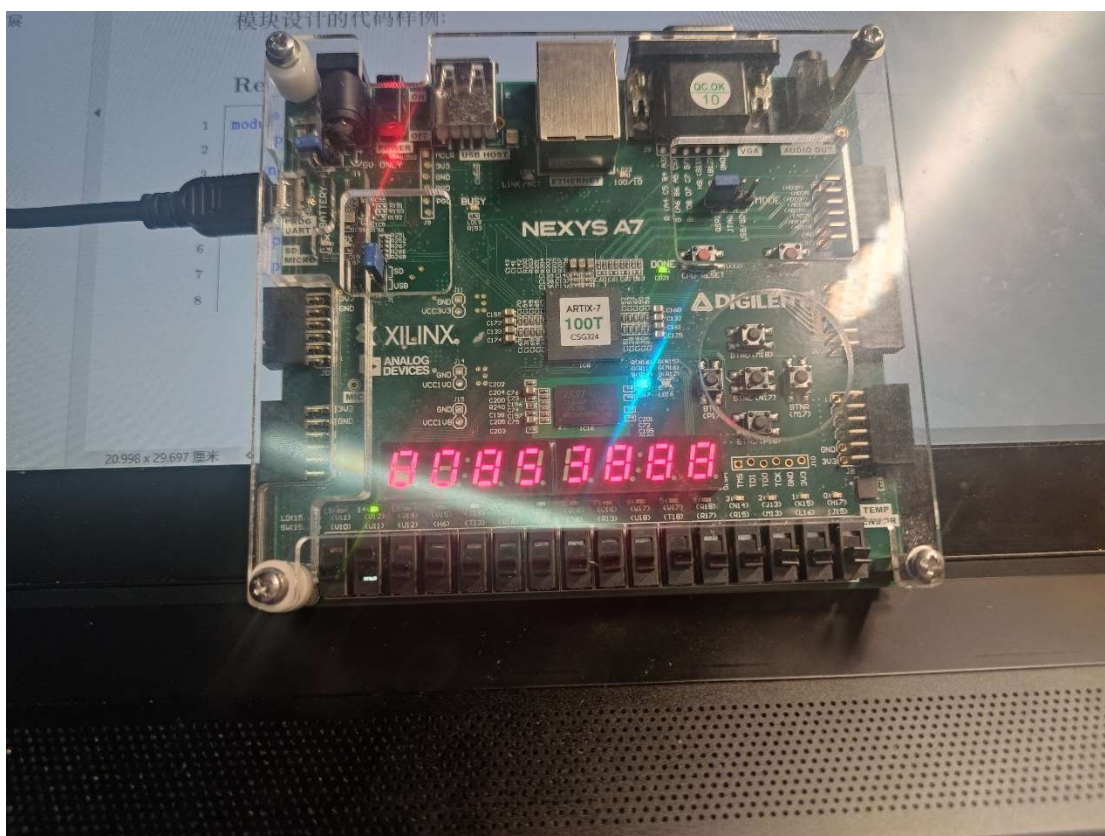
四、 操作方法与实验步骤

4.1 操作方法

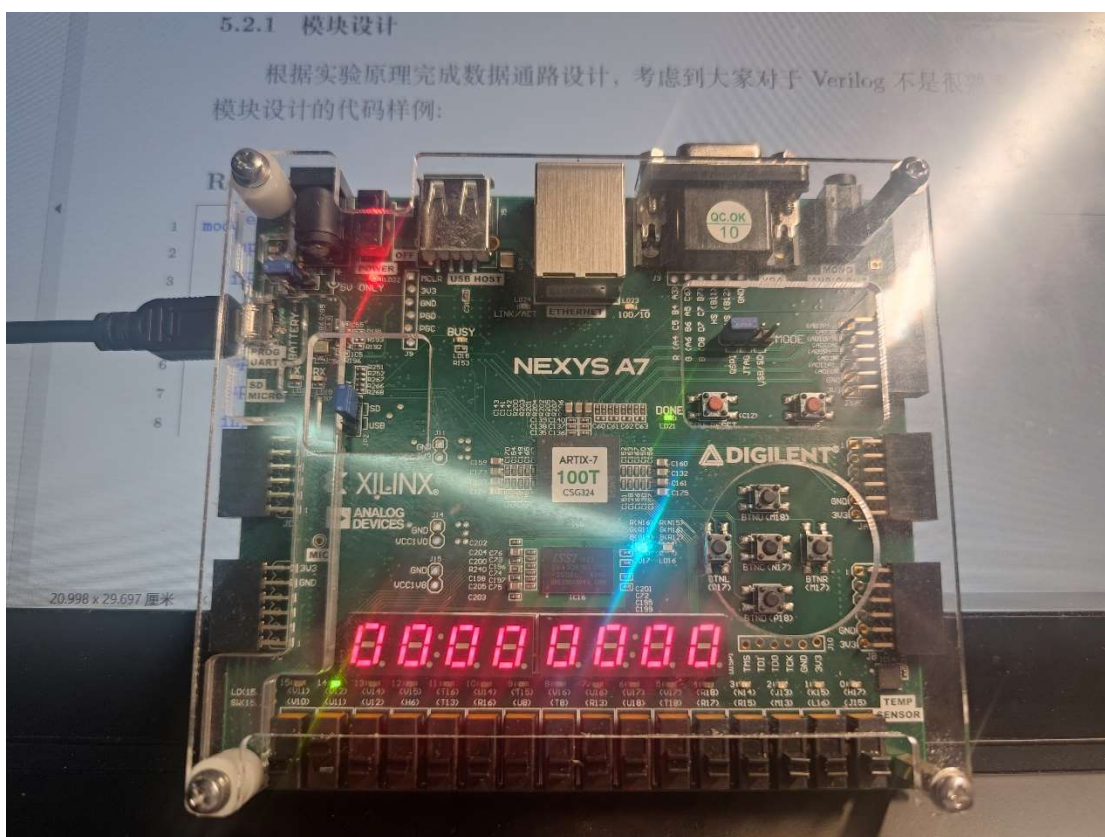
点亮七段管：可以看出 PC 内为 32'b 0000 0000h



CPU 自行运行：
(由于一直是变化状态，照片没法表述。。)



通过几次单步调试后：
(此显示为 PC 内的值)



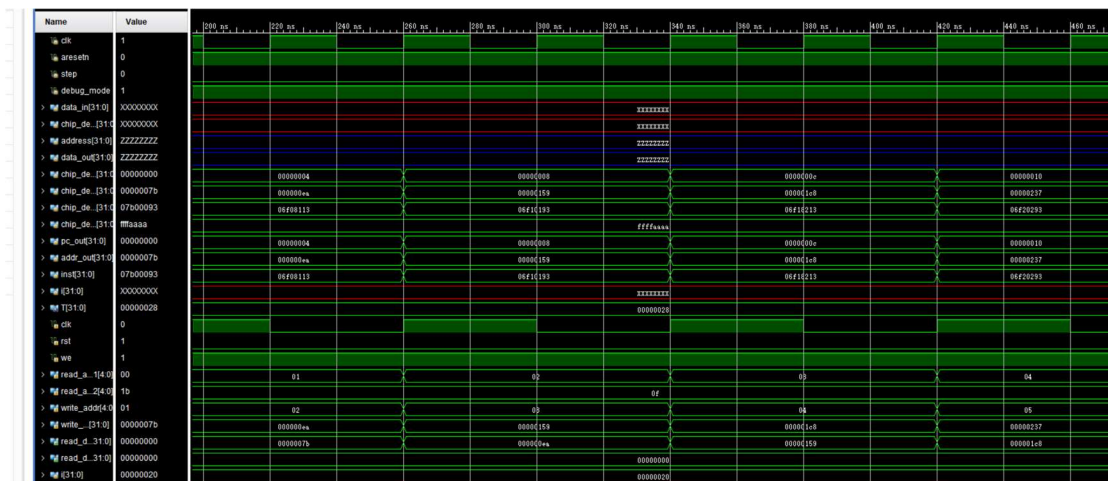
4.2 实验步骤

Lab10_1

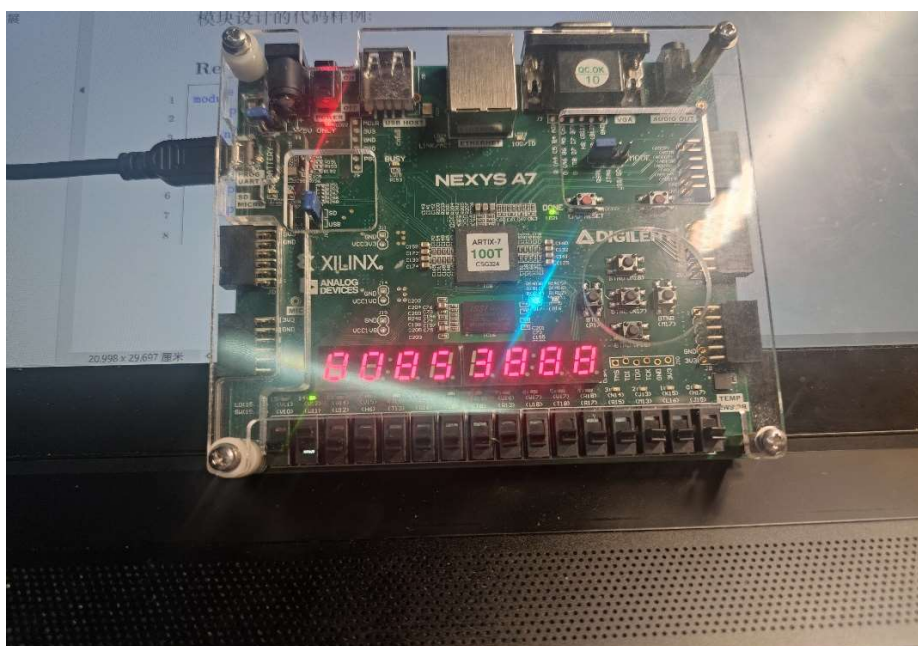
1. 设计模块，如上部分
2. IP 核生成，以及 Coe 文件的导入：

```
> rom_unit : Rom (Rom.xci)
> ram_unit : Ram (Ram.xci)
> io_manager_inst : IO_Manager (IO_Manager
> Core_tb (Core_tb.sv) (1)
  Coefficient Files (1)
    lab10_all.coe
```

3. 仿真测试：

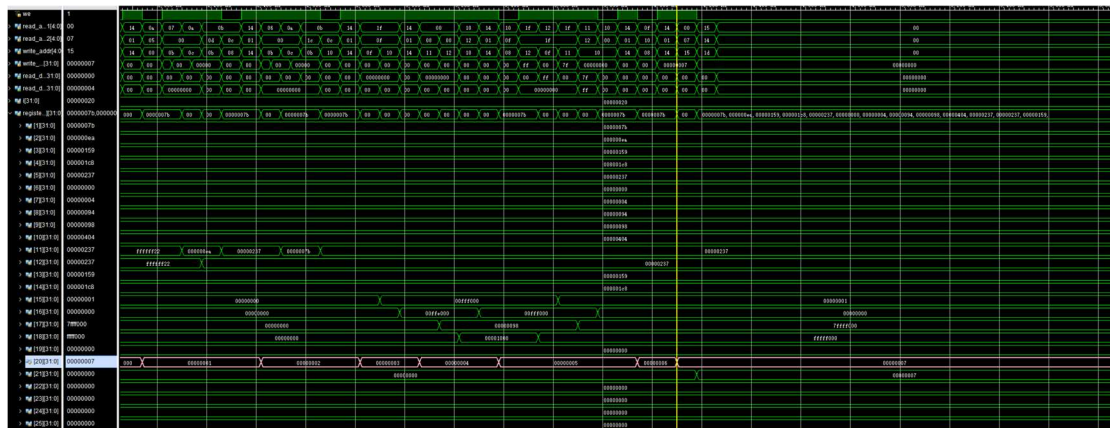


4. 上板测试：

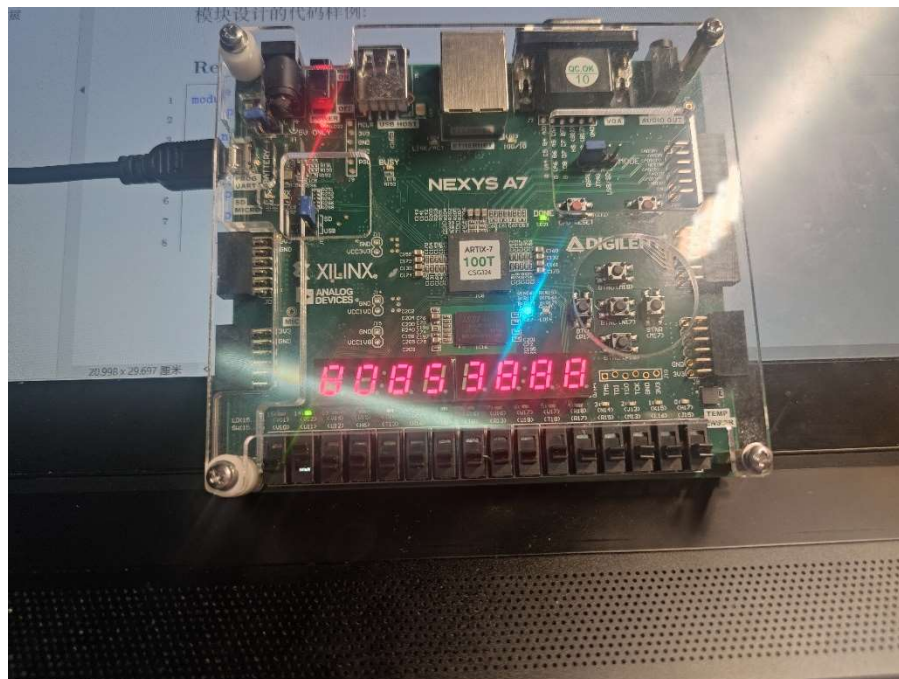


Lab10_2

1. 撰写 `control` 代码：如上部分所示
2. 仿真结果：`x20` 从 `0000 0000` 跳到 `0000 0007`，经过极长时间重新开始循环，但原因不明。。。



上板结果:



五、实验结果与分析

仿真波形:

