第六次实验报告

存储器与控制器设计实验

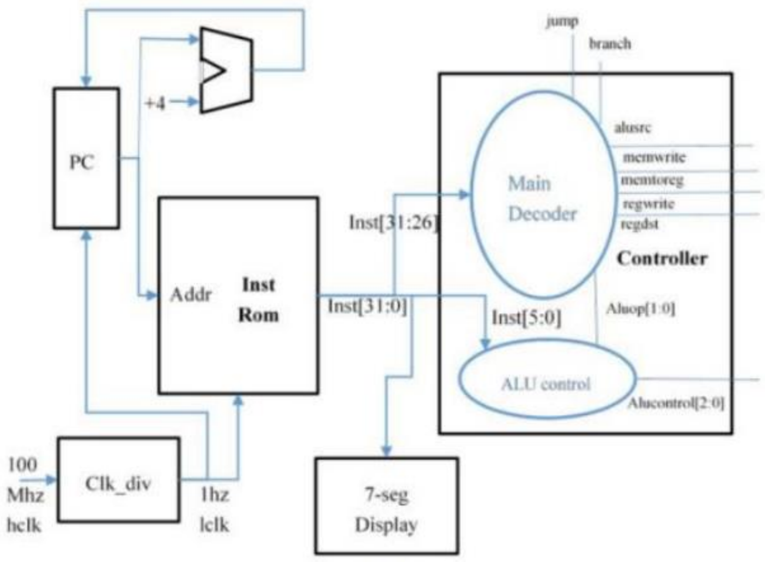
22336313 郑鸿鑫

一．实验目的

1. 掌握调用 Xlinx 库 IP（Block Memory Generator）实例化 ROM 的方法。
2. 掌握单周期 CPU 各个控制信号的作用和生成过程。
3. 掌握单周期 CPU 控制器的工作原理和设计方法。
4. 理解单周期 CPU 执行指令的过程。
5. 掌握取指、译码阶段数据通路、控制器的执行过程。

二．实验内容

下图为本次实验的原理图



本实验实现了以下模块：

PC，用于存储 PC（一个周期），并计算下一条指令的地址；Controller，其中包含两部分 a）main_decoder，负

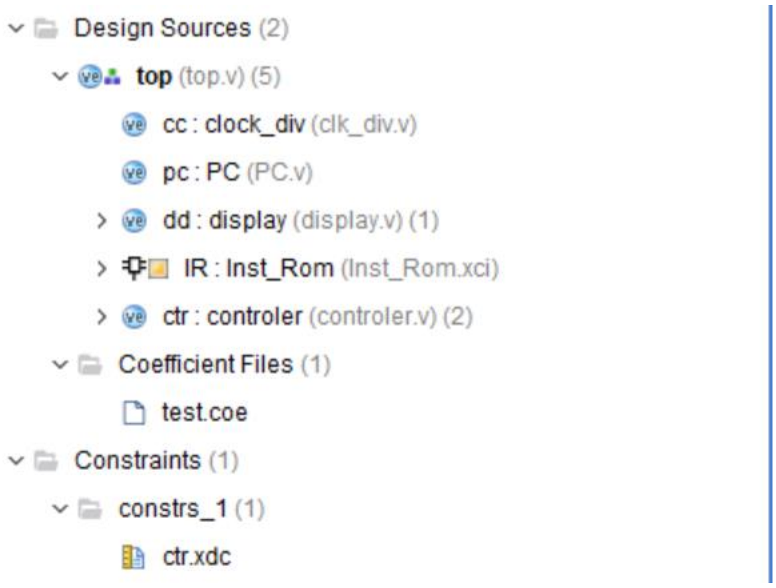
责判断指令类型，并生成相应的控制信号 b）alu_decoder，负责 ALU 模块控制信号的译码，通过接收 aluop 配合 funct 字段生成 aluctr 信号；时钟分频器，将板载的 100MHz 频率降低为 1Hz，输出分频后的时钟信号；指令存储器，使用 Block Memory Generator 构造。

三．实验过程

1. 从存储器实验中导入 display，clk_div 模块
2. 创建 PC 模块
3. 创建 main_decoder，alu_decoder 模块
4. 创建 Controller 模块，调用 3 中的两个子模块
5. 使用 Block Memory，导入 coe 文件
6. 自定义顶层文件，连接相关模块
7. 待结果无误后，上板子进行测试

以下为模块代码展示：

模块层次


Design Sources (2)
top (top.v) (5)
cc : clock_div (clk_div.v)
pc : PC (PC.v)
dd : display (display.v) (1)
IR : Inst_Rom (Inst_Rom.xci)
ctr : controler (controler.v) (2)
Coefficient Files (1)
test.coe
Constraints (1)
constrs_1 (1)
ctr.xdc

Top 文件

```
    module top(
    input clk,
    input rst,
    output memtoreg,
    output memwrite,
    // output pcsrc,
    output alusrc,
    output regdst,
    output regwrite,
    output jump,
    output branch,
    output [2:0] aluctr,
    output [6:0]seg,
    output [3:0]ans
    );
    wire  CLK;
    wire [31:0] addre;
    wire [31:0] douta;
 //   wire [31:0] naddre;
    clock_div cc(
        .clk(clk),
        .clk_sys(CLK)
    );
    PC pc(
        .clk(CLK),
        .rst(rst),
        .addre(addre)
    );
    display dd(
        .clk(clk),
        .reset(rst),
        .addra(douta[15:0]),
        .seg(seg),
        .ans(ans)
    );
    Inst_Rom IR (
    .clka(CLK),    // input clka
    .addra(addre[9:2]),  // input [7 : 0] addra
    .douta(douta)  // output [31 : 0] douta
    );
    controler ctr(
        .op(douta[31:26]),
        .funct(douta[5:0]),
        .memtoreg(memtoreg),
        .memwrite(memwrite),
    // output pcsrc,
        .alusrc(alusrc),
        .regdst(regdst),
        .regwrite(regwrite),
        .jump(jump),
        .branch(branch),
        .aluctr(aluctr)
    );
Endmodule
```

时钟分频器代码:

```
module clock_div(
    input clk,
    output reg clk_sys = 0
    );
    reg [25:0] div_counter = 0;
    always @(posedge clk) begin
        if(div_counter>=50000000) begin//上板子使用
        //if(div_counter>=50) begin//仿真使用
            clk_sys <= ~clk_sys;
            div_counter <= 0;
            end
        else begin
            div_counter <= div_counter + 1;
        end
    end
endmodule
```

PC 模块代码:

```
module PC (
    input clk,
    input rst,
    output reg [31:0] addre
);
    always @(posedge clk or negedge rst) begin
        if(rst)begin
            addre <= 32'b0;
        end
        else begin
            addre <= addre + 32'h4;
        end
    end
endmodule
```

display 模块代码，在数码管中展示指令的低 16 位

```
module display(
    input wire clk,reset,
    input wire [15:0] addra,
    output wire [6:0]seg,
    output reg [3:0]ans
    );
    reg [20:0]count;
    reg [4:0]digit;
    wire [15:0] s;
    assign s = addra[15:0];
    always@(posedge clk,posedge reset)
    if(reset)
        count = 0;
    else
        count = count + 1;
    always @(posedge clk)
    case(count[20:19])
    0:begin
        ans = 4'b1110;
        digit = s[3:0];
    end
    1:begin
```

```
        ans = 4'b1101;
        digit = s[7:4];
    end
    2:begin
        ans = 4'b1011;
        digit =s[11:8];
    end
    3:begin
        ans = 4'b0111;
        digit = s[15:12];
    end
    endcase
    seg U4(digit,seg);
endmodule
```

seg 模块代码（用于将 16 进制数译码到数码管显示）：

```
module seg(
    input [3:0]din,
    output reg [6:0]dout
    );
    always@(*)
    begin
    case(din)
        4'h0:dout = 7'b000_0001;
        4'h1:dout = 7'b100_1111;
        4'h2:dout = 7'b001_0010;
        4'h3:dout = 7'b000_0110;
        4'h4:dout = 7'b100_1100;
        4'h5:dout = 7'b010_0100;
        4'h6:dout = 7'b010_0000;
        4'h7:dout = 7'b000_1111;
        4'h8:dout = 7'b000_0000;
        4'h9:dout = 7'b000_0100;
        4'ha:dout = 7'b000_1000;
        4'hb:dout = 7'b110_0000;
        4'hc:dout = 7'b011_0001;
        4'hd:dout = 7'b100_0010;
        4'he:dout = 7'b011_0000;
        4'hf:dout = 7'b011_1000;
        default:dout = 7'b000_0000;
    endcase
    end
endmodule
```

controller 模块代码：

```
module controler(
    input [5:0] op,
    input [5:0] funct,
    output wire memtoreg,
    output wire memwrite,
    // output reg pcsrc,
    output wire alusrc,
    output wire regdst,
    output wire regwrite,
    output wire jump,
    output wire branch,
    output wire [2:0] aluctr
    );
    main_decoder mm(
        .op(op),
        .memtoreg(memtoreg),
        .memwrite(memwrite),
    //    .pcsrc(pcsrc),
        .alusrc(alusrc),
        .regdst(regdst),
        .regwrite(regwrite),
        .jump(jump),
        .branch(branch),
        .aluop(aluop)
    );
    alu_decoder aa(
        .funct(funct),
        .aluop(aluop),
        .aluctr(aluctr)
    );
endmodule
```

main_decoder 模块代码（识别指令的 opcode 在生成控制信号）：

```
module main_decoder(
    input [5:0]op,
    output reg memtoreg,
    output reg memwrite,
  //  output reg pcsrc,
    output reg alusrc,
    output reg regdst,
    output reg regwrite,
    output reg jump,
    output reg branch,
    output reg [1:0] aluop
    );
    always@(*)begin
        case(op)
        6'b000000:begin
            regwrite = 1;
            regdst = 1;
            alusrc = 0;
            branch = 0;
            memwrite = 0;
            memtoreg = 0;
            aluop = 2'b10;
        end
        6'b000010:begin
            regwrite = 0;
            memwrite = 0;
        end
        6'b100011:begin
            regwrite = 1;
            regdst = 0;
            alusrc = 1;
            branch = 0;
            memwrite = 0;
            memtoreg = 1;
            aluop = 2'b00;
        end
        6'b101011:begin
            regwrite = 0;
            alusrc = 1;
            branch = 0;
            memwrite = 1;
            aluop = 2'b00;
        end
        6'b000100:begin
            regwrite = 0;
            alusrc = 0;
            branch = 1;
            memwrite = 0;
            aluop = 2'b01;
        end
        6'b001000:begin
            regwrite = 1;
            regdst = 0;
            alusrc = 1;
            branch = 0;
            memwrite = 0;
            memtoreg = 0;
            aluop = 2'b00;
        end
        default: begin
            regwrite = 0;
            regdst = 0;
            alusrc = 0;
            branch = 0;
            memwrite = 0;
            memtoreg = 0;
            aluop = 2'b00;
        end
    endcase
    end
endmodule
```

alu_decoder 模块代码:

```
module alu_decoder (
    input [5:0]funct,
    input [1:0]aluop,
    output reg [2:0] aluctr
);
    always @(*) begin
        case(aluop)
        2'b00: begin
            aluctr = 3'b010;
        end
        2'b01: begin
            aluctr = 3'b110;
        end
        2'b10: begin
            case(funct)
            6'b100000: aluctr = 3'b010;
            6'b100010: aluctr = 3'b110;
            6'b100100: aluctr = 3'b000;
            6'b100101: aluctr = 3'b001;
            6'b101010: aluctr = 3'b111;
            endcase
        end
        default: aluctr = 3'b000;
        endcase
    end
endmodule
```

引脚分配 xdc 文件

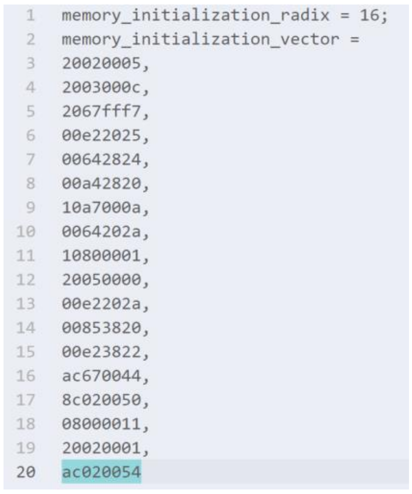
```
# clk signal
set_property PACKAGE_PIN W5 [get_ports clk]
    set_property IOSTANDARD LVCMOS33 [get_ports clk]
    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports clk]
# leds
set_property PACKAGE_PIN U16 [get_ports memwrite]
    set_property IOSTANDARD LVCMOS33 [get_ports memwrite]
set_property PACKAGE_PIN E19 [get_ports memtoreg]
    set_property IOSTANDARD LVCMOS33 [get_ports memtoreg]
set_property PACKAGE_PIN U19 [get_ports alusrc]
    set_property IOSTANDARD LVCMOS33 [get_ports alusrc]
set_property PACKAGE_PIN V19 [get_ports regwrite]
    set_property IOSTANDARD LVCMOS33 [get_ports regwrite]
set_property PACKAGE_PIN W18 [get_ports regdst]
    set_property IOSTANDARD LVCMOS33 [get_ports regdst]
set_property PACKAGE_PIN U15 [get_ports branch]
    set_property IOSTANDARD LVCMOS33 [get_ports branch]
set_property PACKAGE_PIN U14 [get_ports jump]
    set_property IOSTANDARD LVCMOS33 [get_ports jump]
set_property PACKAGE_PIN V14 [get_ports aluctr[0]]
    set_property IOSTANDARD LVCMOS33 [get_ports aluctr[0]]
set_property PACKAGE_PIN V13 [get_ports aluctr[1]]
    set_property IOSTANDARD LVCMOS33 [get_ports aluctr[1]]
set_property PACKAGE_PIN V3 [get_ports aluctr[2]]
    set_property IOSTANDARD LVCMOS33 [get_ports aluctr[2]]

# 7 segment display
set_property PACKAGE_PIN W7 [get_ports {seg[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]
set_property PACKAGE_PIN W6 [get_ports {seg[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U8 [get_ports {seg[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V5 [get_ports {seg[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U7 [get_ports {seg[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property PACKAGE_PIN U2 [get_ports {ans[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {ans[0]}]
set_property PACKAGE_PIN U4 [get_ports {ans[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {ans[1]}]
set_property PACKAGE_PIN V4 [get_ports {ans[2]}]
```

```
    set_property IOSTANDARD LVCMOS33 [get_ports {ans[2]}]
set_property PACKAGE_PIN W4 [get_ports {ans[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {ans[3]}]
##Buttons
set_property PACKAGE_PIN U18 [get_ports rst]
    set_property IOSTANDARD LVCMOS33 [get_ports rst]
```

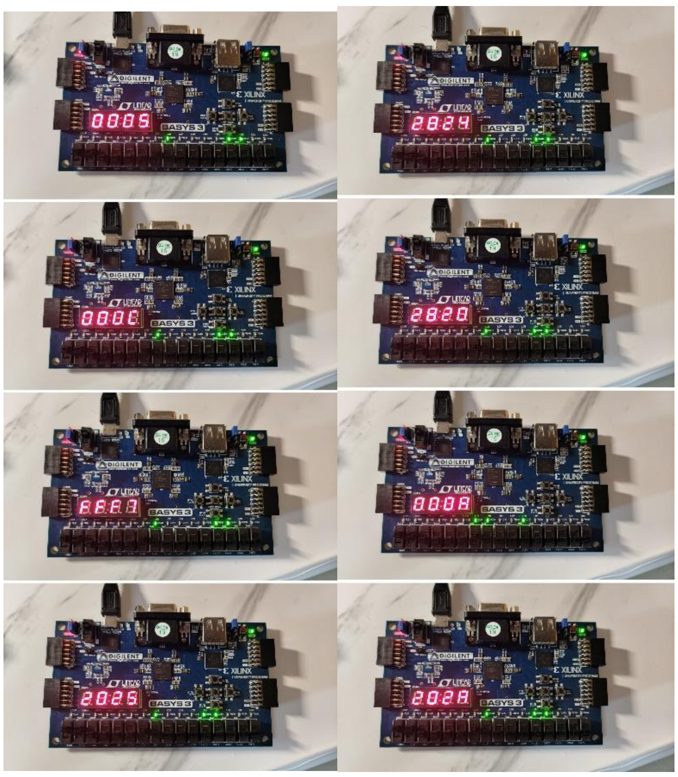
四．实验结果分析

Coe 代码如下：



```
memory_initialization_radix = 16;
memory_initialization_vector =
20020005,
2003000c,
2067fff7,
00e22025,
00642824,
00a42820,
10a7000a,
0064202a,
10800001,
20050000,
00e2202a,
00853820,
00e23822,
ac670044,
8c020050,
08000011,
20020001,
ac020054
```

模块综合后上板结果：（以前几条指令为例，且只展示低 16 位）



五．实验总结

实验结果符合预期，数码管显示与 coe 文件符合良好，在按下 rst 按钮后，Pc 会清零后重新从 0 地址开始读出指令。实验中遇到的问题：实现过程中数码管总是显示乱码，检查后发现，是引脚分配文件中 seg 0~6 配置反了；数码管总是只显示一位，而且本身 display 代码没有问题，检查后发现是接入的时钟是分频后的时钟导致时钟频率过低，无法进行动态扫描，改接分频前的时钟后则正常