数字电路实验 Lab 5 实验报告

奚项正 PB23000020

必做内容

题目 1：寄存器堆（3 分）

设计

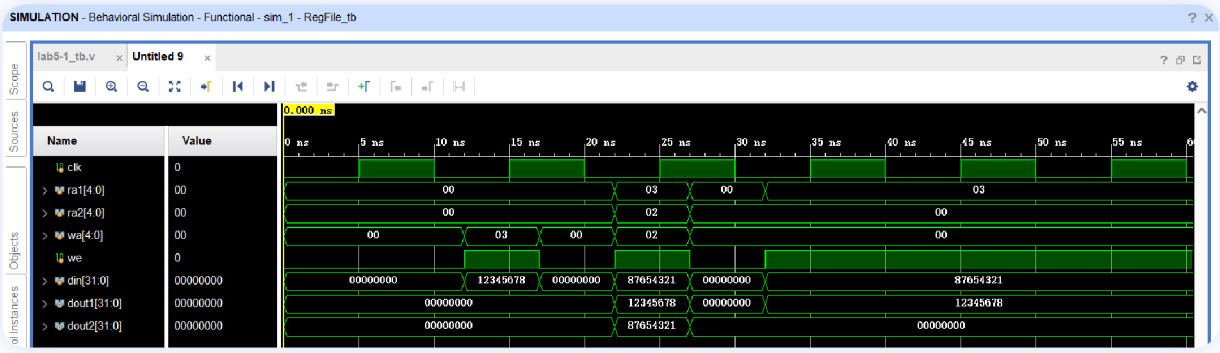
```
module RegFile (
    input                       clk,          // 时钟信号
    input         [4:0]         ra1,          // 读端口 1 地址
    input         [4:0]         ra2,          // 读端口 2 地址
    input         [4:0]         wa,           // 写端口地址
    input                       we,           // 写使能信号
    input         [31:0]        din,          // 写数据
    output  reg   [31:0]        dout1,        // 读端口1数据输出
    output  reg   [31:0]        dout2         // 读端口2数据输出
);

reg [31:0] reg_file [31:0]; // 32 个 32 位寄存器，规模为 32×32 bits

// 写端口
always @(posedge clk) begin //上升沿触发
    reg_file[0] <= 0;
    if (we && wa != 0) begin     //写使能为1且写地址非0时
        reg_file[wa] <= din;
    end
    else begin
        reg_file[wa] <= reg_file[wa];
    end
end

// 读端口
always @(*) begin
    if (ra1 == 0) begin //0号寄存器始终为0
        dout1 = 0;
    end
    else if (we && ra1 == wa) begin //同时读写时，能读到正在写入的最新数据
        dout1 = din;
    end
```

```
    else begin
        dout1 = reg_file[ra1];
    end
end

always @(*) begin    //端口2与端口1完全类似
    if (ra2 == 0) begin
        dout2 = 0;
    end
    else if (we && ra2 == wa) begin
        dout2 = din;
    end
    else begin
        dout2 = reg_file[ra2];
    end
end
endmodule
```

仿真

```
module RegFile_tb ();
reg             clk;
reg     [ 4 : 0] ra1;
reg     [ 4 : 0] ra2;
reg     [ 4 : 0] wa;
reg              we;
reg     [31 : 0] din;
wire    [31 : 0] dout1;
wire    [31 : 0] dout2;

initial begin
    clk = 0;
    ra1 = 5'H0; ra2 = 5'H0; wa = 5'H0; we = 1'H0; din = 32'H0;

    #12;
    ra1 = 5'H0; ra2 = 5'H0; wa = 5'H3; we = 1'H1; din = 32'H12345678;

    #5;
    ra1 = 5'H0; ra2 = 5'H0; wa = 5'H0; we = 1'H0; din = 32'H0;

    #5;
    ra1 = 5'H3; ra2 = 5'H2; wa = 5'H2; we = 1'H1; din = 32'H87654321;

    #5;
    ra1 = 5'H0; ra2 = 5'H0; wa = 5'H0; we = 1'H0; din = 32'H0;

    #5;
    ra1 = 5'H3; ra2 = 5'H0; wa = 5'H0; we = 1'H1; din = 32'H87654321;
end

always #5 clk = ~clk;

RegFile regfile(
```

```
    .clk(clk),
    .ra1(ra1),
    .ra2(ra2),
    .wa(wa),
    .we(we),
    .din(din),
    .dout1(dout1),
    .dout2(dout2)
);
endmodule
```



题目 2：数码管时钟（3 分）

设计

```
module TIMER (
    input           clk,
    input           rst,
    output  [3:0]   out,
    output  [2:0]   select
);

reg [3:0] out_h;    //时
reg [7:0] out_m;    //分
reg [7:0] out_s;    //秒
reg [39:0] cnt;     //计数器，此4个量作为现态

reg [3:0] next_h;
reg [7:0] next_m;
reg [7:0] next_s;
reg [39:0] next_cnt;    //此4个量作为次态

wire [31:0] data;   //将4个量拼接以供数码管显示模块调用
assign data = {12'b000000000000, out_h, out_m, out_s};

always @(posedge clk) begin //使用同步时序进行状态更新
    if (rst) begin  //按动按钮可将时钟复位到95830
        out_h <= 4'h9;
        out_m <= 8'h58;
        out_s <= 8'h30;
        cnt <= 1;
```

```
    end
    else begin
        out_h <= next_h;
        out_m <= next_m;
        out_s <= next_s;
        cnt <= next_cnt;
    end
end

always @(*) begin    //使用组合逻辑判断状态跳转逻辑，计数器部分，cnt=100000000时为1s
    next_cnt = cnt;
    if (cnt >= 100000000) begin //为提高仿真速度，仿真时改为10000
        next_cnt = 1;
    end
    else begin
        next_cnt = cnt + 1;
    end
end

always @(*) begin    //使用组合逻辑判断状态跳转逻辑，“秒”部分
    next_s = out_s;      //进行默认赋值，防止出现遗漏
    if(cnt == 1) begin       //cnt每次循环为1s，需要更新out_s值
        case(out_s)
            8'h09:   next_s = 8'h10;     //采用16进制，用case单独列出进位的情形
            8'h19:   next_s = 8'h20;
            8'h29:   next_s = 8'h30;
            8'h39:   next_s = 8'h40;
            8'h49:   next_s = 8'h50;
            8'h59:   next_s = 8'h00;
            default: next_s = out_s + 1;     //非进位情形直接+1
        endcase
    end
end

always @(*) begin    //使用组合逻辑判断状态跳转逻辑，“分”部分
    next_m = out_m;
    if(cnt == 1 && out_s == 8'h59) begin
        case(out_m)
            8'h09:   next_m = 8'h10;
            8'h19:   next_m = 8'h20;
            8'h29:   next_m = 8'h30;
            8'h39:   next_m = 8'h40;
            8'h49:   next_m = 8'h50;
            8'h59:   next_m = 8'h00;
            default: next_m = out_m + 1;
        endcase
    end
end

always @(*) begin    //使用组合逻辑判断状态跳转逻辑，“时”部分
    next_h = out_h;
    if(cnt == 1 && out_s == 8'h59 && out_m == 8'h59) begin //out_s更新后，若out_s与out_m同时进
位，则out_h需要+1
        next_h = out_h + 1;      //out_h超出15后首位自动溢出
```

```
    end
end

Segment segment(
    .clk(clk),
    .rst(rst),
    .output_data(data),
    .output_valid(8'b11111111),
    .seg_data(out),
    .seg_an(select)
);

endmodule
```

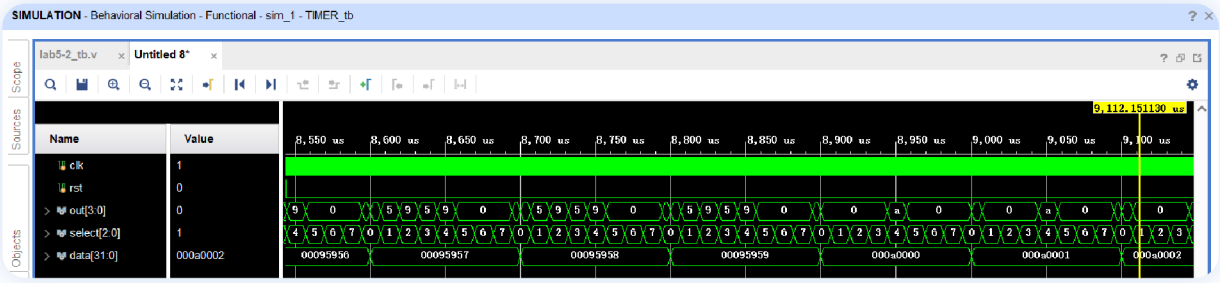
```
module Segment( //此代码来自lab3数码管显示模块
    input                       clk,
    input                       rst,
    input       [31:0]          output_data,
    input       [ 7:0]          output_valid,
    output reg  [ 3:0]          seg_data,
    output reg  [ 2:0]          seg_an
);

reg [31:0]  counter;
reg [2:0]   seg_id;

always @(posedge clk) begin
    if (rst) begin
        counter <= 0;
    end
    else if (counter >= 250000) begin    //为提高仿真速度，仿真时改为1250
        counter <= 0;
    end
    else begin
        counter <= counter + 1;
    end
end

always @(posedge clk) begin
    if (rst) begin
        seg_id <= 0;
    end
    else if (counter == 1) begin
        if (seg_id >= 8) begin
            seg_id <= 0;
        end
        else begin
            seg_id <= seg_id + 1;
        end
    end
    else begin
        seg_id <= seg_id;
    end
```

```
end

always @(*) begin
    seg_data = 0;
    if (output_valid[seg_id] == 1) begin
        seg_an = seg_id;
    end
    else begin
        seg_an = 0;
    end

    case (seg_an)
        0: seg_data = output_data[3:0];
        1: seg_data = output_data[7:4];
        2: seg_data = output_data[11:8];
        3: seg_data = output_data[15:12];
        4: seg_data = output_data[19:16];
        5: seg_data = output_data[23:20];
        6: seg_data = output_data[27:24];
        7: seg_data = output_data[31:28];
        default: seg_data = 0;
    endcase
end
endmodule
```

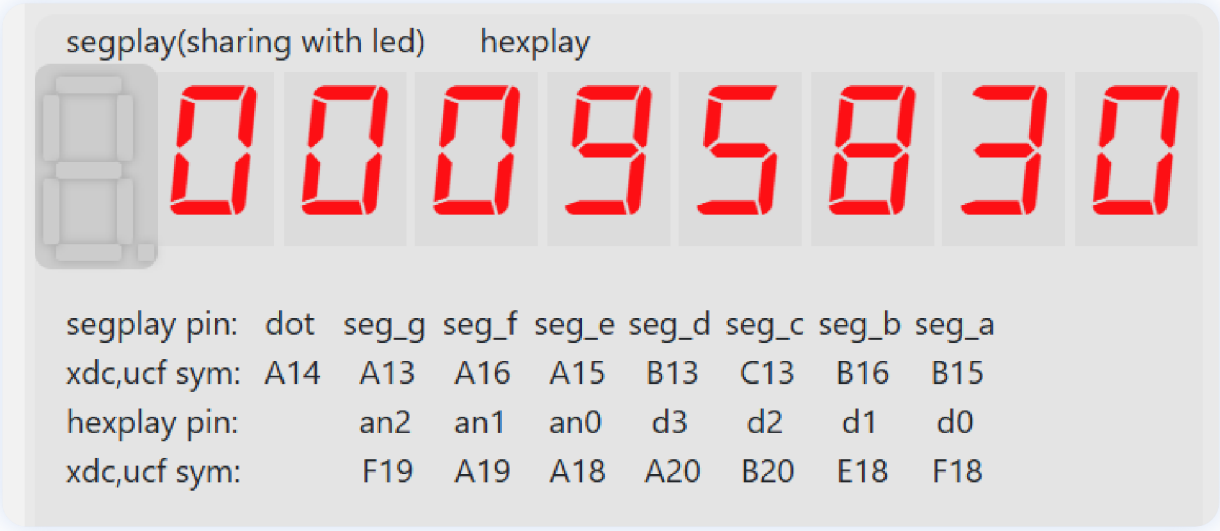
仿真

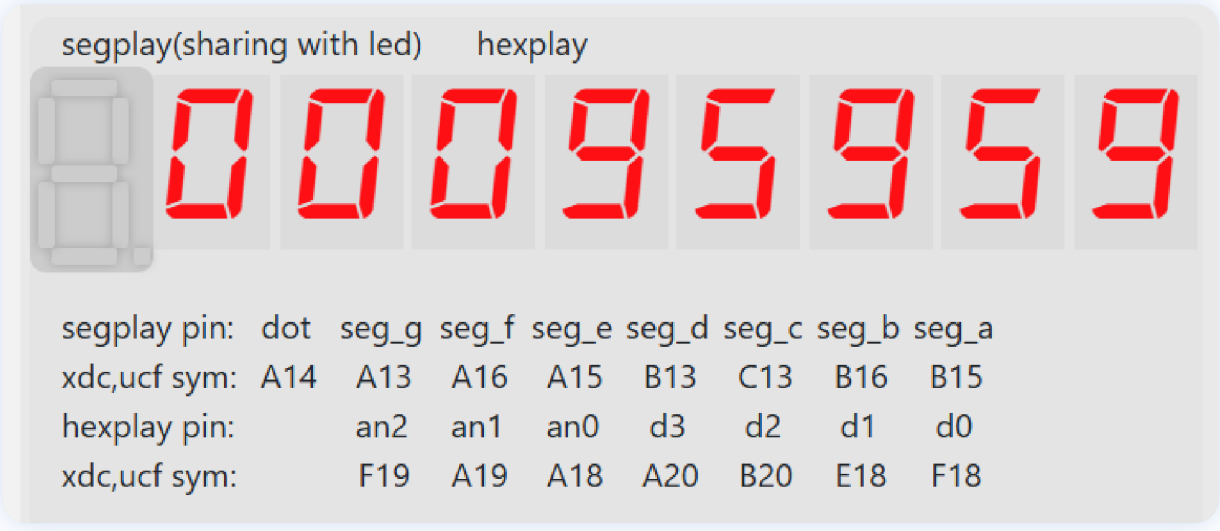
```
module TIMER_tb();
reg             clk;
reg             rst;
wire    [3:0]   out;
wire    [2:0]   select;

initial begin
    clk = 1;
    rst = 1;
    #10;
    rst = 0;
end

always #5 clk = ~clk;

TIMER timer(
    .clk(clk),
    .rst(rst),
    .out(out),
    .select(select)
);
endmodule
```

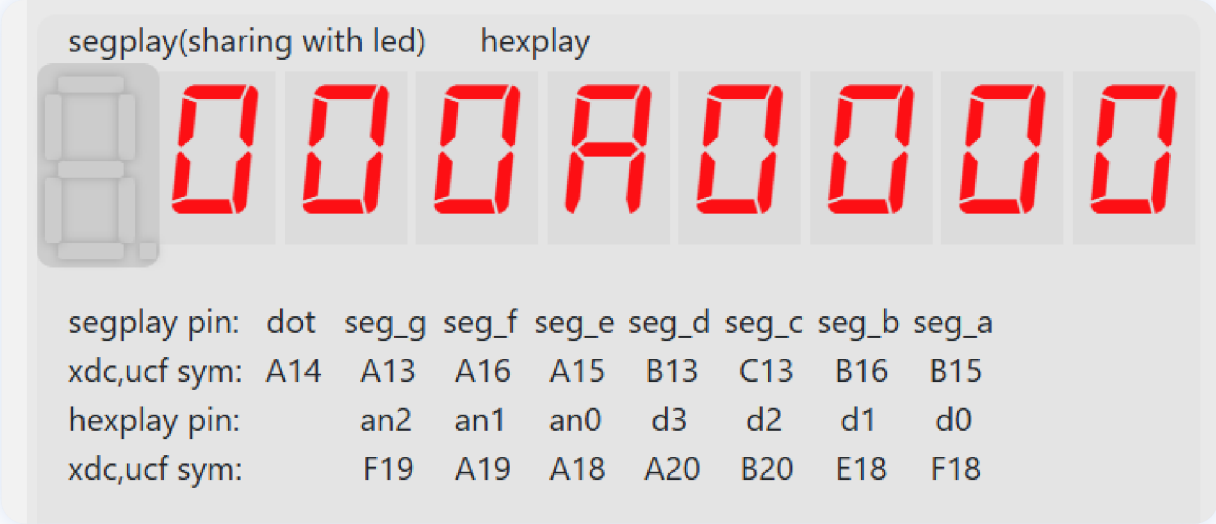
> 注：为提高仿真速度，进行仿真时对部分时间参数进行修改，可参见代码注释


SIMULATION - Behavioral Simulation - Functional - sim_1 - TIMER_tb
lab5-2_tb.v
Untitled 8*
Scope
Sources
Objects
Name
Value
clk
rst
out[3:0]
select[2:0]
data[31:0]
000a0002
9,112.151130 us
00095956
00095957
00095958
00095959
000a0000
000a0001
000a0002

实现


segplay(sharing with led)
hexplay
00095830
segplay pin: dot seg_g seg_f seg_e seg_d seg_c seg_b seg_a
xdc,ucf sym: A14 A13 A16 A15 B13 C13 B16 B15
hexplay pin: an2 an1 an0 d3 d2 d1 d0
xdc,ucf sym: F19 A19 A18 A20 B20 E18 F18


segplay(sharing with led)
hexplay
00095959
segplay pin: dot seg_g seg_f seg_e seg_d seg_c seg_b seg_a
xdc,ucf sym: A14 A13 A16 A15 B13 C13 B16 B15
hexplay pin: an2 an1 an0 d3 d2 d1 d0
xdc,ucf sym: F19 A19 A18 A20 B20 E18 F18



选择性必做内容

题目 1：倍数检测器——再临（4 分）

设计

```
module Top(
    input           clk,            // 时钟信号
    input           rst,            // 复位信号，使状态机回到初始态
    input   [31:0]  src,            // 输入数据
    input           src_valid,      // 表明输入结果是否有效
    output  reg     ready,          // 表明是否正在检测
    output  reg     res,            // 输出结果
    output  reg     res_valid       // 表明输出结果是否有效
);

// 状态变量
reg [6:0] current_state;     //存储当前状态，33表示rst初态，32表示检测完毕态，31-0表示正在检测态且记录正在读入src的位数
reg [6:0] next_state;
reg [3:0] current_mod;  //存储当前余数，即正在检测态下，读入前若干位所构造数的模7余数
reg [3:0] next_mod;
reg [31:0] tmp;

// Part 1: 使用同步时序进行状态更新，即更新 current_state 的内容。
always @(posedge clk) begin
    if (rst) begin
        current_state <= 33;
        current_mod <= 0;
    end
    else begin
        current_state <= next_state;
        current_mod <= next_mod;
    end
end
```

```
// Part 2: 使用组合逻辑判断状态跳转逻辑，即根据 current_state 与其他信号确定 next_state。
always @(*) begin
    next_state = current_state;
    case (current_state)
        33: begin   //rst初态
            if (src_valid == 1) begin
                tmp = src;
                next_state = 31;
            end
        end
        32: begin   //检测完毕态
            if (src_valid == 1) begin
                tmp = src;
                next_state = 31;
            end
        end
        0:  begin   //正在检测且即将检测完毕态
            next_state = 32;
        end
        default: begin  //正在检测态
            next_state = next_state - 1;
        end
    endcase
end

always @(*) begin
    next_mod = current_mod;
    if (current_state <= 31 && current_state >= 0) begin    //current_state在0-31间表示正在检测
        case (current_mod)  //根据有限状态机状态转换图，次态=现态*2+读入下一位为0/1
            0: begin
                if (tmp[current_state] == 1) begin
                    next_mod = 1;
                end
                else begin
                    next_mod = 0;
                end
            end
            1: begin
                if (tmp[current_state] == 1) begin
                    next_mod = 3;
                end
                else begin
                    next_mod = 2;
                end
            end
            2: begin
                if (tmp[current_state] == 1) begin
                    next_mod = 5;
                end
                else begin
                    next_mod = 4;
                end
            end
            3: begin
```

```
                if (tmp[current_state] == 1) begin
                    next_mod = 0;
                end
                else begin
                    next_mod = 6;
                end
            end
            4: begin
                if (tmp[current_state] == 1) begin
                    next_mod = 2;
                end
                else begin
                    next_mod = 1;
                end
            end
            5: begin
                if (tmp[current_state] == 1) begin
                    next_mod = 4;
                end
                else begin
                    next_mod = 3;
                end
            end
            6: begin
                if (tmp[current_state] == 1) begin
                    next_mod = 6;
                end
                else begin
                    next_mod = 5;
                end
            end
        endcase
    end
end

// Part 3: 使用组合逻辑描述状态机的输出。
always @(*) begin
    case (current_state)
        33: begin   //rst初态
            ready = 1;
            res_valid = 0;
        end
        32: begin   //检测完毕态
            ready = 1;
            res_valid = 1;
        end
        0:  begin   //正在检测且即将检测完毕态
            ready = 0;
            res_valid = 0;
        end
        default: begin  //正在检测态
            ready = 0;
            res_valid = 0;
        end
```

```
    endcase
end

always @(*) begin
    res = 0;
    if (current_state == 32 && current_mod == 0) begin   //检测完毕态，若为7的倍数，则res=1
        res = 1;
    end
end

endmodule
```

仿真

```
module Top_tb();
reg           clk;            // 时钟信号
reg           rst;            // 复位信号，使状态机回到初始态
reg   [31:0]  src;            // 输入数据
reg           src_valid;      // 表明输入结果是否有效
wire          ready;          // 表明是否正在检测
wire          res;            // 输出结果
wire          res_valid;      // 表明输出结果是否有效

initial begin
    clk = 0;
    rst = 1;

    #10;
    rst = 0;
    src_valid = 1;
    src = 7;
    #10;
    src_valid = 0;

    #400
    src_valid = 1;
    src = 14;
    #10;
    src_valid = 0;

    #400
    src_valid = 1;
    src = 15;
    #10;
    src_valid = 0;

    #400
    rst = 1;

    #10;
    rst = 0;
    src_valid = 1;
    src = 896;
```

```
    #10;
    src_valid = 0;

    #200    //正在检测态下输入会被忽略
    src_valid = 1;
    src = 1;
    #10;
    src_valid = 0;

    #400
    src_valid = 1;
    src = 165;
    #10;
    src_valid = 0;
end

always #5 clk = ~clk;

Top top(
    .clk(clk),
    .rst(rst),
    .src(src),
    .src_valid(src_valid),
    .ready(ready),
    .res(res),
    .res_valid(res_valid)
);
endmodule
```