《数字逻辑与数字系统》实验报告

学院 智算学部 年级 2019 级 班级 计科 1 班 姓名 李润泽 学号 3019244266

课程名称 数字逻辑与数字系统 实验日期 2021.5.27 成绩_____

同组实验者_____

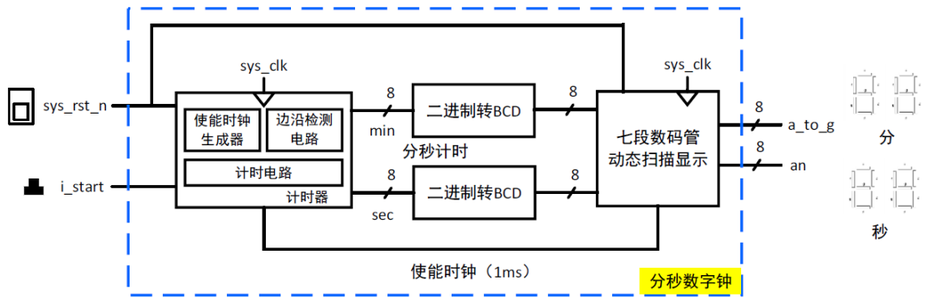
实验项目名称 分秒数字钟的设计与实现

一. 实验目的

1、掌握基于 SystemVerilog HDL 的时序逻辑电路建模方法；
2、掌握计数器设计方法，并能够使用计数器设计使能时钟（用于时钟分频）；
3、掌握移位寄存器设计方法，并能够利用移位寄存器设计边沿检测电路；
4、掌握 7 段数码管的动态显示。

二. 实验内容

基于 SystemVerilog HDL 设计并实现一个分秒数字钟。整个工程的顶层模块如下图所示，输入/输出端口如右表所示。使用 4 个七段数码管显示当前的计时。其中，两个数码管对应“分”，另两个数码管对应“秒”。用过 1 个拨动开关对数字钟进行复位控制。使用 1 个按键对数字中进行“暂停/计时”控制，按键每按下一次，进行暂停和计时的切换，即暂停时，按下按键启动计时；计时过程中，按下按键暂停计时。



分秒数字钟顶层模块

分秒数字钟由 3 部分构成：

第一部分是数字钟的核心——计时器模块，该模块又由 3 个子模块构成，分别是计时电路、使能时钟生成电路和边沿检测电路。

计时电路通过计数器实现计时功能，产生二进制的“分”（min）和“秒”（sec）输出；

使能时钟生成电路用于产生控制七段数码管动态显示的使能时钟，使能时钟高电平出现的周期为 1ms；

天津大学本科生实验报告专用纸

边沿检测电路模块对按键输入进行上升沿检测，产生控制计时器暂停和启动的信号。

第二部分是 8 位二进制转 BCD 模块，它将二进制的“分”、“秒”值转化为 BCD 编码，即 10 进制数。其中，分、秒各使用一个。本实验中，该模块不需要实现，有教师直接提供 IP 使用。

第三部分是 7 段数码管动态扫描显示模块，它实现“分”、“秒”值的最终显示。“分”、“秒”值各使用两个 7 段数码管进行显示。

输入/输出端口

端口名	方向	宽度（位）	作用
sys_clk	输入	1	系统基准时钟，主频 25MHz。
sys_rst_n	输入	1	连接拨动开关，对数字中进行复位。复位信号对低电平有效。
i_start	输入	1	连接按键，用于控制暂停/计时。每按下按键进行暂停和计时切换。按键按下为高电平。
a_to_g	输出	8	连接七段数码管的数据输入端 CA~CG 和 DP，用于显示秒表的分和秒。采用共阳极控制，数码管低电平点亮。DP 段永远不点亮。
an	输出	4	连接 4 个七段数码管的使能端 AN0~AN3，其中 AN0 和 AN1 显示秒，AN2 和 AN3 显示分。使能信号高电平有效。

完成上述分秒数字钟的设计，需要有以下几点需要注意：

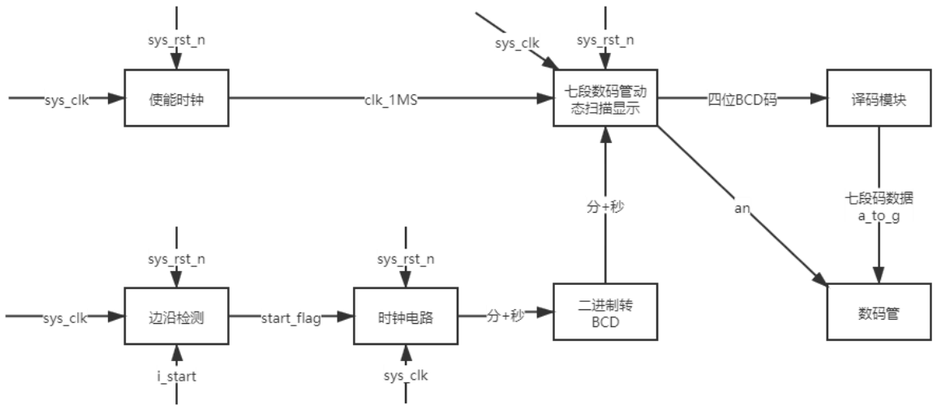
1、7 段数码管动态扫描必须采用使能时钟实现，扫描频率为 1KHz（1ms）。
2、必须通过边沿检测电路识别“暂停/计时”按键按下产生的上升沿，以边进行后续处理。
3、用于计时的时钟频率为 25MHz（40ns）。
4、本实验在 xdc 约束文件中添加时钟约束，所对应的语句如下：

```
set_property -dict {PACKAGE_PIN F5 IOSTANDARD LVCMOS33} [get_ports {sys_clk}];
create_clock -add -name sys_clk_pin -period 40.00 [get_ports {sys_clk}];
```

5、由于分秒数字钟计时单位为 1 秒，7 段数码管扫描周期是 1ms，从而造成仿真时间过长。为了加快仿真速度，可以在仿真的时候使用较大的计时单位和扫描速度。

三．实验原理与步骤（注：步骤不用写工具的操作步骤，而是设计步骤）

1. 画出分秒数字钟电路的原理图



2. 分秒数字钟电路中一共使用了几个计数器，作用分别是什么

共使用了 7 个计数器。
（我的想法以自己的代码为准）
（1）七段数码管动态扫描显示（bcd_scanner.sv）中使用了 1 个计数器，用于分频；
（2）时钟电路（timer_function.sv）中使用了 4 个计数器，用于计时；
（3）使能时钟（enable_clock.sv）中使用了 2 个计数器，用于根据计数值按固定时间间隔产生一个周期的高电平使能信号。

3. 写出分秒数字钟的 SystemVerilog HDL 代码。

（1）dig_clock.sv

```
`timescale 1ns / 1ps
module dig_clock(
    input               sys_clk,
    input               sys_rst_n,
    input               i_start,
    output logic [3 : 0]an,
    output logic [7 : 0]a_to_g
    );

    logic [7:0]sec,min,sec_bcd,min_bcd;
    logic clk_flag;
    timer timer1(sys_clk, sys_rst_n, i_start, sec, min, clk_flag);
    bin2bcd_0 bcd_sec(sec, sec_bcd);
    bin2bcd_0 bcd_min(min, min_bcd);

    bcd_scanner bcd_scanner1(sys_rst_n, clk_flag, sys_clk, min_bcd, sec_bcd, an, a_to_g);

endmodule
```

（2）timer.sv

```
`timescale 1ns / 1ps
module timer(
    input logic sys_clk,
    input logic sys_rst_n,
    input logic i_start,
    output logic[7:0] sec,
    output logic[7:0] min,
    output logic clk_flag
    );

    logic pos_edge;
    logic min_temp;

    enable_clock enable(sys_clk,sys_rst_n,clk_flag);
    edge_detection detect(
        .sys_clk(sys_clk),
        .sys_rst_n(sys_rst_n),
        .i_btn(i_start),
        .pos_edge(pos_edge)
    );
    timer_function clock(sys_clk,sys_rst_n,pos_edge,i_start,sec,min);

endmodule
```

（3）enable_clock.sv

```
`timescale 1ns / 1ps
module enable_clock #(parameter SYS_CLK_FREQ=100_000_000, TARGET_CLK_PREQ=10_000_000, N=3) (
    input logic sys_clk,
    input logic sys_rst_n,
    output logic clk_flag
    );
    localparam CNT_MAX = 24999;
```

```
    logic [14:0]r_cnt;
    always_ff@(posedge sys_clk) begin
        if(!sys_rst_n) r_cnt <= 0;
        else if(r_cnt == CNT_MAX) r_cnt <= 0;
        else r_cnt <= r_cnt+1;
    end
    always_ff@(posedge sys_clk) begin
        if(!sys_rst_n) clk_flag <= 0;
        else if(r_cnt == CNT_MAX) clk_flag <= 1;
        else clk_flag <= 0;
    end

endmodule
```

（4）edge_detection.sv

```
`timescale 1ns / 1ps
module edge_detection(
    input logic sys_clk,
    input logic sys_rst_n,
    input logic i_btn,
    output logic pos_edge
    );

    logic [1:0]dff_Q;

    always_ff@(posedge sys_clk) begin
        if(!sys_rst_n) dff_Q <= 2'b00;
        else dff_Q <= {dff_Q[0], i_btn};
    end

    assign pos_edge = ~dff_Q[1] & dff_Q[0];

endmodule
```

（5）timer_function.sv

```
`timescale 1ns / 1ps
module timer_function(
    input logic sys_clk,
    input logic sys_rst_n,
    input logic pos_edge,
    input logic i_start,
    output logic[7:0] sec,
    output logic[7:0] min
    );

    logic [24:0]temp;
    logic temp2;
    localparam temp_MAX = 25'b1011111010111100000111111;

    always_ff@(posedge sys_clk) begin
        if(!sys_rst_n) temp2 <= 0;
        else if(pos_edge) temp2 <= temp2+1'b1;
        else temp2 <= temp2;
    end
    always_ff@(posedge sys_clk) begin
        if(!sys_rst_n) temp <= 0;
        else if(temp2 == 1) begin
            if(temp != temp_MAX) temp <= temp+1;
            else temp <= 0;
        end
        else temp <= temp;
    end

    always_ff@(posedge sys_clk) begin
        if(!sys_rst_n) sec <= 0;
        else if(temp == temp_MAX) begin
            if(sec != 59) sec <= sec+1'b1;
            else sec <= 0;
        end
    end
    always_ff@(posedge sys_clk) begin
        if(!sys_rst_n) min <= 0;
        else if(sec == 59 & temp == temp_MAX) min <= min+1'b1;
        else min <= min;
    end

endmodule
```

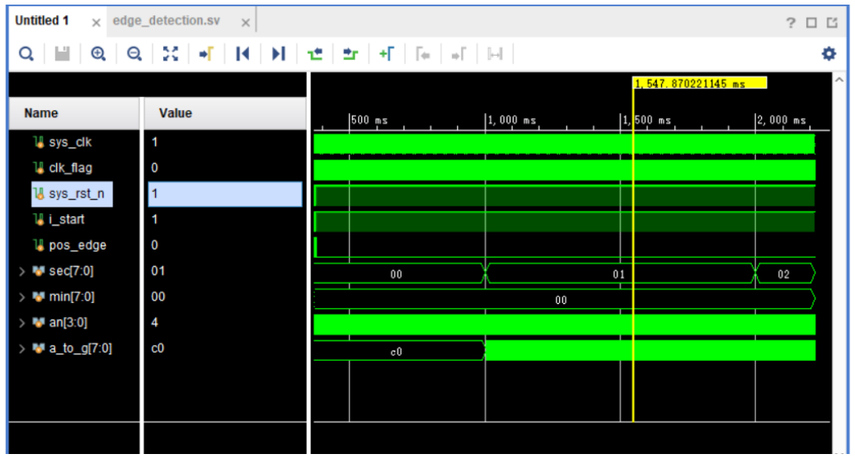
（6）bcd_scanner.sv

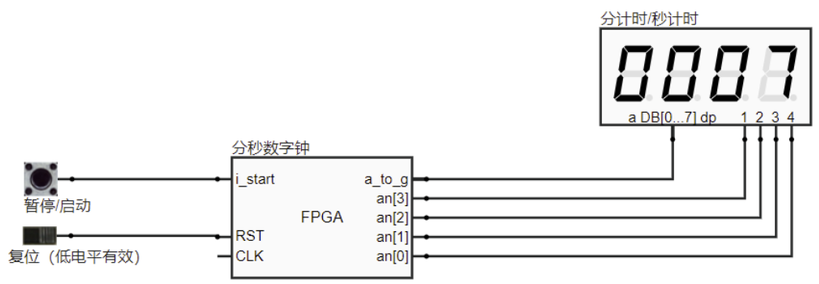
```
`timescale 1ns / 1ps
module bcd_scanner(
    input logic sys_rst_n,
    input logic clk_flag,
    input logic sys_clk,
    input logic[7:0] min,
```

```
input logic[7:0] sec,
output logic[3:0] an,
output logic[7:0] a_to_g
);

logic [1:0]temp;
logic [3:0]data;

always_ff@(posedge sys_clk) begin
    if(!sys_rst_n) temp <= 0;
    else if(clk_flag) begin
        if(temp == 3) temp <= 2'b00;
        else temp <= temp+1;
    end
end
always_ff@(posedge clk_flag) begin
    if(!sys_rst_n) begin
        data <= 0;
        a_to_g <= ~8'b00111111;
        an <= 4'b1111;
    end
    else begin
        case(temp)
            0 : begin an<=4'b0001; end
            1 : begin an<=4'b0010; end
            2 : begin an<=4'b0100; end
            3 : begin an<=4'b1000; end
        endcase
        case(an)
            4'b0001 : begin data<= min[3:0]; end
            4'b0010 : begin data<= min[7:4]; end
            4'b0100 : begin data<= sec[3:0]; end
            4'b1000 : begin data<= sec[7:4]; end
        endcase
        case(data)
            4'b0000: a_to_g<=~8'b00111111;
            4'b0001: a_to_g<=~8'b00000110;
            4'b0010: a_to_g<=~8'b01011011;
            4'b0011: a_to_g<=~8'b01001111;
            4'b0100: a_to_g<=~8'b01100110;
            4'b0101: a_to_g<=~8'b01101101;
            4'b0110: a_to_g<=~8'b01111101;
            4'b0111: a_to_g<=~8'b00000111;
            4'b1000: a_to_g<=~8'b01111111;
            4'b1001: a_to_g<=~8'b01101111;
            default: a_to_g<=~8'b00111111;
        endcase
    end
end

endmodule
```

四．仿真与实验结果（注：仿真需要给出波形图截图，截图要清晰，如果波形过长，可以分段截取；实验结果为远程 FPGA 硬件云平台的截图）

注：远程 FPGA 硬件云平台截图只需要一个测试激励即可





五．实验中遇到的问题和解决办法

Vivado 无法生成比特流：

在进行 Vivado 实验中，其中一个步骤：Generate Bitstream 时发生报错，其中，重点句如下：

NOTE: When using the Vivado Runs infrastructure (e.g. launch_runs Tcl command), add this command to a .tcl file and add that file as a pre-hook for write_bitstream step for the implementation run.

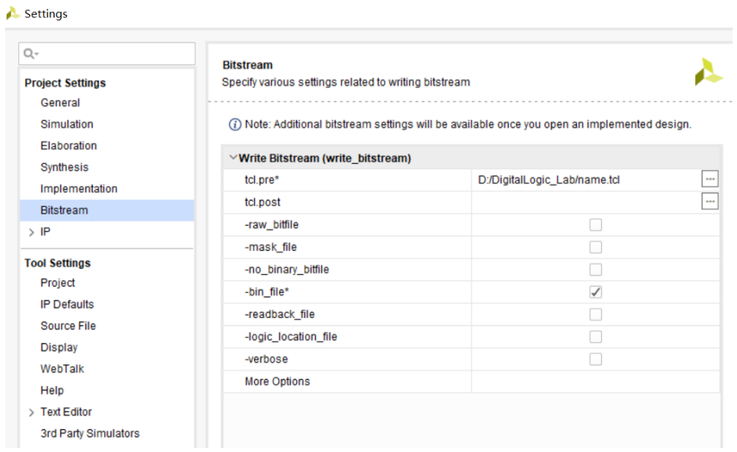
在进行网络查询之后，我发现缺少一个文件，因此，我们可以将如下三行代码：

set_property SEVERITY {Warning} [get_drc_checks NSTD-1]

set_property SEVERITY {Warning} [get_drc_checks UCIO-1]

set_property SEVERITY {Warning} [get_drc_checks RTSTAT-1]

写到一个.tcl 文件中并保存，再将保存文件导入，便可正确生成比特流，操作如下：



六．附加题（若实验指导书无要求，则无需回答）

教师签字：

年　月　日