姓名 卢 班级 ******* 学号 **********

实验日期 _____节次 _____教师签字 _____成绩_____

基于 BASYS2 的简易数字秒表

1. 实验目的

①熟悉 Verilog 语言的语法和使用。
②理解并运用模块化设计思想进行设计。
③掌握 Moore 型状态机的设计方法。
④能够利用 ISE 软件开发简单的 FPGA 应用。

2. 总体设计方案或技术路线

1) 确定设计要求

设计一个数字秒表，计时范围为 0~9min59.9s，通过四位数码管显示示值，具有复位和启停按钮，实现复位和计时/停止功能，且计时溢满时自动停止。

2) 制定设计方案

总体采用模块化设计方法，将任务按功能特性分解，让每个模块负责一个相对独立的功能：

0.主模块：对接口进行包装，并将各个子模块连接起来。

1.时钟分频模块：对内部时钟进行分频，提供一个较低频的时钟用于驱动各时序模块。

2.控制模块：核心部分，响应操作并控制计时。

控制模块的基本行为可以使用一个 Moore 型状态机描述。状态机有四个状态：RESET（复位）、STOP（停止）、RUN（计时进行）和 OVERFLOW（溢出），并通过三个输入控制：rst（复位）、toggle（启停）和 overflow（溢出标志）。控制模块的状态转换示意图如图 1。

3.数码管显示模块：将输入转换为数码管显示。

4.按键检测模块：检测按键输入响应，主要是消抖，以及保证启停按键只在按下时响应（边沿触发）。

3) 编写程序并下载验证。

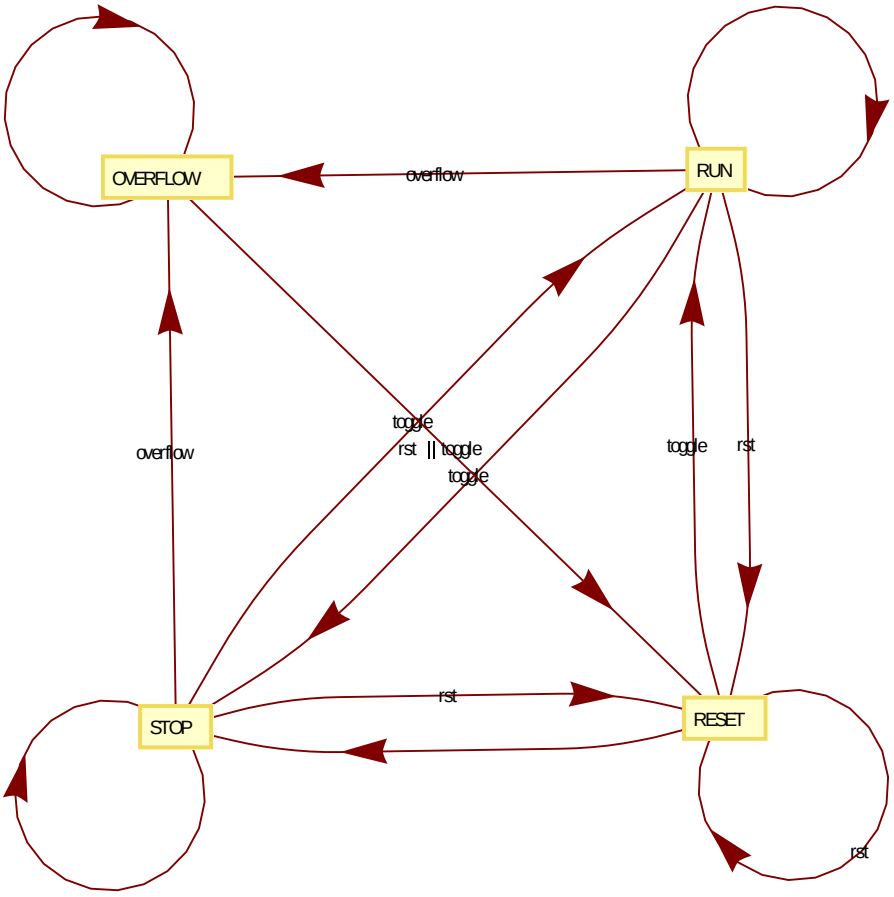

图 1：控制模块的状态转换示意图

3. 实验电路图

因采用 Verilog 语言进行编写的程序，故没有实验电路图。

4. 仪器设备名称、型号

Xilinx BASYS2 开发板

5. 理论分析或仿真分析结果

主要对作为核心的控制模块进行仿真验证。

为了便于显示仿真结果，仿真所用的时钟输入周期为 5μs，而模块设计所期待的时钟输入周期为 5ms（实际由时钟分频模块产生），故仿真显示的时间尺度比实际小了 10^3。

图 2 是对复位、启停功能的验证。

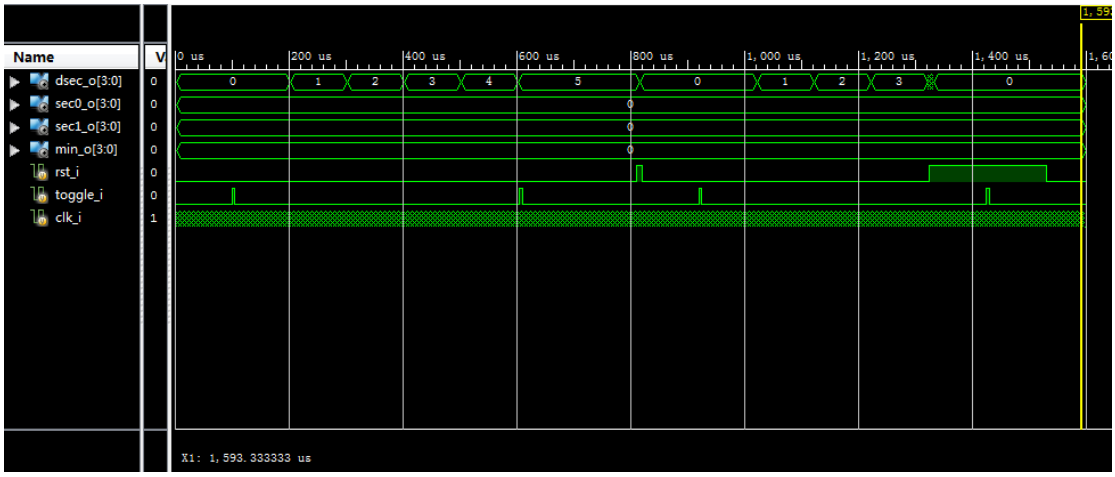

图 2：复位、启停功能仿真波形

图 3 是对计时功能和特性的验证。

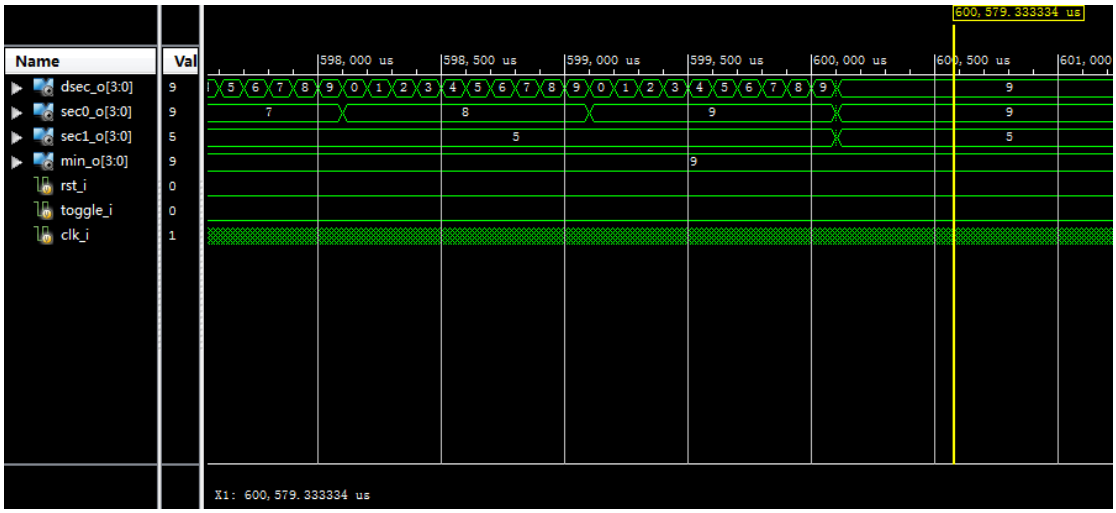

图 3：计时功能仿真波形

6. 详细实验步骤及实验结果数据记录（包括各仪器、仪表量程及内阻的记录）

(1)根据设计方案，使用 Verilog 语言编写程序（程序源码见附录）。

(2)将编写好的程序下载进开发板中，按设计要求对功能进行验证。

7. 实验结论

成功利用 BASYS2 开发板按实现了一个符合预期设计要求的简易秒表。

8. 实验中出现的问题及解决对策

①问题：开始设计时分频驱动时钟频率太低，导致数码管扫描周期过长，目视有明显闪烁。

解决对策：调整参数，提高分频模块输出的时钟频率。

②问题：编写程序时，想当然地将一个模块的输出连接到 reg 类型的信号上，导致综合失败。
解决对策：引入 wire 类型的中间信号，在 always 语句中对 reg 类型的信号进行赋值。

9.本次实验的收获和体会、对电路实验室的意见或建议

锻炼了使用 FPGA 进行开发的能力，加深了对模块化设计、有限状态机等思想的理解。

10．参考文献

[1] 杨春玲,王淑娟主编. 数字电子技术基础[M]. 2 版. 北京:高等教育出版社, 2017

附录：程序源码

主模块 - main.v

```
module main(
    input clk,          // 50MHz
    input [1:0] btn,    // 按钮
    output [3:0] an,    // 数码管位选
    output ca,          // 数码管段选
    output cb,
    output cc,
    output cd,
    output ce,
    output cf,
    output cg,
    output dp
    );

    wire clk5ms;
    wire rst, toggle;
    wire [3:0] dsec;
    wire [3:0] sec0;
    wire [3:0] sec1;
    wire [3:0] min;

    clkGen u1(
    .clk50MHz_i (clk),
    .clk5ms_o   (clk5ms)
    );

    controller u2(
    .rst_i      (rst),
    .toggle_i   (toggle),
    .clk_i      (clk5ms),
    .dsec_o     (dsec),
    .sec0_o     (sec0),
    .sec1_o     (sec1),
    .min_o      (min)
    );

    digitDisplay u3(
    .bcd_i0     (dsec),
    .bcd_i1     (sec0),
    .bcd_i2     (sec1),
    .bcd_i3     (min),
```

```
    .dp_i       (4'b0101),
    .clk_i      (clk5ms),
    .sel_o      (an),
    .a_o        (ca),
    .b_o        (cb),
    .c_o        (cc),
    .d_o        (cd),
    .e_o        (ce),
    .f_o        (cf),
    .g_o        (cg),
    .dp_o       (dp)
    );

    keyPress u4(
    .key_i      (btn[1]),
    .pressed_o  (rst),
    .clk_i      (clk5ms)
    );

    keyClick u5(
    .key_i      (btn[0]),
    .clicked_o  (toggle),
    .clk_i      (clk5ms)
    );

endmodule
```

时钟分频模块 - clkGen.v

```
module clkGen(
    input clk50MHz_i,
    output clk5ms_o
    );

    parameter TIMES = 250_000; // 5ms

    reg [18:0] cnt; // see alse TIMES
    reg out;

    initial
    begin
        cnt <= 0;
        out <= 0;
    end
```

```
    always@(posedge clk50MHz_i)
    begin
        if(cnt < (TIMES-1))
            cnt <= cnt +1;
        else
            cnt <= 0;
    end

    always@(posedge clk50MHz_i)
    begin
        if(cnt == 0)
            out <= 1;
        else if(cnt == (TIMES/2))
            out <= 0;
        else
            out <= out;
    end

    assign clk5ms_o = out;

endmodule
```

控制模块 – controller.v

```
module controller(
    input rst_i,
    input toggle_i,
    input clk_i,        // 5ms
    output [3:0] dsec_o,
    output [3:0] sec0_o,
    output [3:0] sec1_o,
    output [3:0] min_o
    );

    parameter RESET = 0, STOP = 1, RUN = 2, OVERFLOW = 3;
    parameter BASE = 20; // 100ms

    reg [3:0] dsec_reg;
    reg [3:0] sec0_reg;
    reg [3:0] sec1_reg;
    reg [3:0] min_reg;
    reg [1:0] current_state, next_state;

    wire overflow;
```

```
    initial
    begin
        current_state <= RESET;
    end

    always@(posedge clk_i)
    begin
        current_state <= next_state;
    end

    always@(current_state or rst_i or toggle_i or overflow)
    begin
        case(current_state)
        RESET:
        begin
            if(rst_i)
                next_state <= RESET;
            else if(toggle_i)
                next_state <= RUN;
            else
                next_state <= STOP;
        end
        STOP:
        begin
            if(rst_i)
                next_state <= RESET;
            else if(toggle_i)
                next_state <= RUN;
            else if(overflow)
                next_state <= OVERFLOW;
            else
                next_state <= STOP;
        end
        RUN:
        begin
            if(rst_i)
                next_state <= RESET;
            else if(toggle_i)
                next_state <= STOP;
            else if(overflow)
                next_state <= OVERFLOW;
            else
                next_state <= RUN;
        end
```

```
        OVERFLOW:
        begin
            if(rst_i || toggle_i)
                next_state <= RESET;
            else
                next_state <= OVERFLOW;
        end
        default:
            next_state <= RESET;
        endcase
    end

    reg [4:0] basecnt; // see also BASE
    always@(posedge clk_i)
    begin
        case(current_state)
        RESET:
            basecnt <= 0;
        STOP:
            basecnt <= basecnt;
        RUN:
        begin
            if(basecnt == (BASE-1))
                basecnt <= 0;
            else
                basecnt <= basecnt + 1;
        end
        OVERFLOW:
            basecnt <= 0;
        endcase
    end

    wire carry_dsec; // 0.1s
    assign carry_dsec = ((dsec_reg == 9) && (basecnt == (BASE-1)));
    always@(posedge clk_i)
    begin
        case(current_state)
        RESET:
            dsec_reg <= 0;
        STOP:
            dsec_reg <= dsec_reg;
        RUN:
        begin
            if(carry_dsec)
```

```
                dsec_reg <= 0;
            else if(basecnt == (BASE-1))
                dsec_reg <= dsec_reg + 1;
            else
                dsec_reg <= dsec_reg;
        end
        OVERFLOW:
            dsec_reg <= 9;
        endcase
    end

    wire carry_sec0; // 1s
    assign carry_sec0 = (carry_dsec && (sec0_reg == 9));
    always@(posedge clk_i)
    begin
        case(current_state)
        RESET:
            sec0_reg <= 0;
        STOP:
            sec0_reg <= sec0_reg;
        RUN:
        begin
            if(carry_sec0)
                sec0_reg <= 0;
            else if(carry_dsec)
                sec0_reg <= sec0_reg + 1;
            else
                sec0_reg <= sec0_reg;
        end
        OVERFLOW:
            sec0_reg <= 9;
        endcase
    end

    wire carry_sec1; // 10s
    assign carry_sec1 = (carry_sec0 && (sec1_reg == 5));
    always@(posedge clk_i)
    begin
        case(current_state)
        RESET:
            sec1_reg <= 0;
        STOP:
            sec1_reg <= sec1_reg;
        RUN:
```

```
        begin
            if(carry_sec1)
                sec1_reg <= 0;
            else if(carry_sec0)
                sec1_reg <= sec1_reg + 1;
            else
                sec1_reg <= sec1_reg;
        end
        OVERFLOW:
            sec1_reg <= 5;
        endcase
    end

    // min
    assign overflow = (carry_sec1 && (min_reg == 9));
    always@(posedge clk_i)
    begin
        case(current_state)
        RESET:
            min_reg <= 0;
        STOP:
            min_reg <= min_reg;
        RUN:
        begin
            if(overflow)
                min_reg <= 9; // hold on
            else if(carry_sec1)
                min_reg <= min_reg + 1;
            else
                min_reg <= min_reg;
        end
        OVERFLOW:
            min_reg <= 9;
        endcase
    end

    assign dsec_o = dsec_reg;
    assign sec0_o = sec0_reg;
    assign sec1_o = sec1_reg;
    assign min_o  = min_reg;

endmodule
```

数码管显示模块 – digitDisplay.v

```
module digitDisplay(
    input [3:0] bcd_i0, // 输入 BCD8421 码
    input [3:0] bcd_i1,
    input [3:0] bcd_i2,
    input [3:0] bcd_i3,
    input [3:0] dp_i,   // 小数点控制
    input clk_i,        // 5ms
    output [3:0] sel_o, // 数码管位选输出，低电平有效
    output a_o,         // 数码管段选输出
    output b_o,
    output c_o,
    output d_o,
    output e_o,
    output f_o,
    output g_o,
    output dp_o
    );

    reg [3:0] sel_reg;
    reg dp_reg;
    reg [3:0] bcd_reg;
    reg [1:0] tubeSel;

    always@(posedge clk_i)
    begin
        case(tubeSel)
        2'd0:
        begin
            sel_reg <= 4'b0111;
            dp_reg <= dp_i[0];
            bcd_reg <= bcd_i0;
        end
        2'd1:
        begin
            sel_reg <= 4'b1011;
            dp_reg <= dp_i[1];
            bcd_reg <= bcd_i1;
        end
        2'd2:
        begin
            sel_reg <= 4'b1101;
            dp_reg <= dp_i[2];
            bcd_reg <= bcd_i2;
```

```
        end
        2'd3:
        begin
            sel_reg <= 4'b1110;
            dp_reg <= dp_i[3];
            bcd_reg <= bcd_i3;
        end
        endcase

        tubeSel <= tubeSel +1;
    end

    assign sel_o = sel_reg;
    digitSeg u1(
    .bcd_i      (bcd_reg),
    .a_o        (a_o),
    .b_o        (b_o),
    .c_o        (c_o),
    .d_o        (d_o),
    .e_o        (e_o),
    .f_o        (f_o),
    .g_o        (g_o)
    );
    assign dp_o = dp_reg;

endmodule
```

数码管段码译码模块 – digitSeg.v

```
module digitSeg(
    input [3:0] bcd_i,  // 输入 BCD8421 码
    output a_o,         // 数码管段选，低电平有效
    output b_o,
    output c_o,
    output d_o,
    output e_o,
    output f_o,
    output g_o
    );

    assign {a_o,b_o,c_o,d_o,e_o,f_o,g_o} =
        (bcd_i == 4'd0) ? 7'b0000001 :
        (bcd_i == 4'd1) ? 7'b1001111 :
        (bcd_i == 4'd2) ? 7'b0010010 :
        (bcd_i == 4'd3) ? 7'b0000110 :
```

```
        (bcd_i == 4'd4) ? 7'b1001100 :
        (bcd_i == 4'd5) ? 7'b0100100 :
        (bcd_i == 4'd6) ? 7'b0100000 :
        (bcd_i == 4'd7) ? 7'b0001111 :
        (bcd_i == 4'd8) ? 7'b0000000 :
        (bcd_i == 4'd9) ? 7'b0000100 :
        7'bx;

endmodule
```

按键按下检测模块（电平触发） – keyPress.v

```
module keyPress(
    input key_i,        // 高电平有效
    output pressed_o,   // 高电平有效
    input clk_i         // 5ms
    );

    parameter DURATION = 10; // 50ms

    reg [3:0] cnt; // see also: DURATION

    initial
    begin
        cnt <= 0;
    end

    always@(posedge clk_i)
    begin
        if(key_i == 1)
        begin
            if(cnt < DURATION)
                cnt <= cnt + 1;
            else
                cnt <= DURATION;
        end
        else
            cnt <= 0;
    end

    assign pressed_o = (cnt == DURATION);

endmodule
```

按键点击检测模块（边沿触发） – keyClick.v

```
module keyClick(
    input key_i,
    output clicked_o,
    input clk_i
    );

    wire pressed;
    reg current, post;

    always@(posedge clk_i)
    begin
        current <= pressed;
    end

    always@(posedge clk_i)
    begin
        post <= current;
    end

    keyPress u1(
    .key_i      (key_i),
    .pressed_o  (pressed),
    .clk_i      (clk_i)
    );

    assign clicked_o = (post == 0) && (current == 1);

endmodule
```

约束文件 port.ucf

```
NET "clk" TNM_NET =clk;
TIMESPEC TS_CLK = PERIOD "clk" 20ns high 50%;

NET "clk" LOC = B8;

NET "btn<0>" LOC = G12;
NET "btn<1>" LOC = C11;

NET "ca" LOC = L14;
NET "cb" LOC = H12;
NET "cc" LOC = N14;
NET "cd" LOC = N11;
NET "ce" LOC = P12;
```

```
NET "cf" LOC = L13;
NET "cg" LOC = M12;
NET "dp" LOC = N13;

NET "an<0>" LOC = K14;
NET "an<1>" LOC = M13;
NET "an<2>" LOC = J12;
NET "an<3>" LOC = F12;
```