基于 Verilog 和 FPGA/CPLD 的多功能秒表设计 - 实验报告

- 学号：
- 姓名：

实验目的

1. 初步掌握利用 Verilog 硬件描述语言进行逻辑功能设计的原理和方法。
2. 理解和掌握运用大规模可编程逻辑器件进行逻辑设计的原理和方法。
3. 理解硬件实现方法中的并行性，联系软件实现方法中的并发性。
4. 理解硬件和软件是相辅相成、并在设计和应用方法上的优势互补的特点。
5. 本实验学习积累的 Verilog 硬件描述语言和对 FPGA/CPLD 的编程操作，是进行后续《计算机组成原理》 部分课程实验，设计实现计算机逻辑的基础。

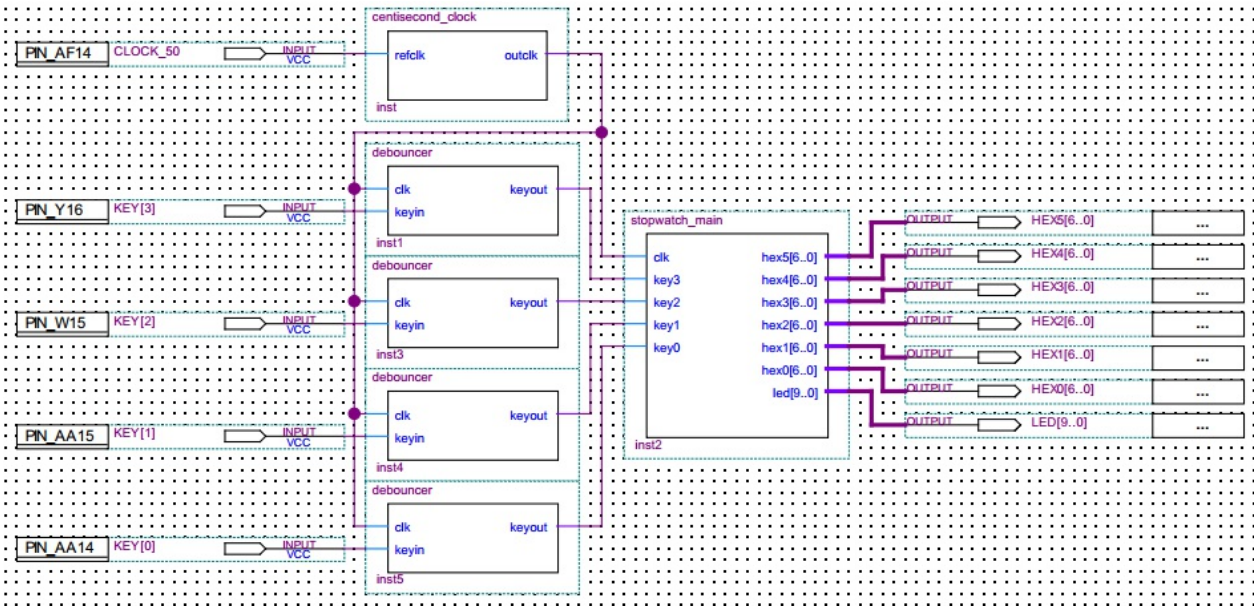
实验仪器与平台

- 硬件：DE1-SoC 实验板
- 软件：Altera Quartus Ⅱ 13.1

实验内容和任务

1. 运用 Verilog 硬件描述语言，基于 DE1-SoC 实验板，设计实现一个具有较多功能的计时秒表。
2. 要求将 6 个数码管设计为具有“分：秒：毫秒”显示，按键的控制动作有：“计时复位”、“计数/暂停”、“显示暂停/显示继续”等。功能能够满足马拉松或长跑运动员的计时需要。
3. 利用示波器观察按键的抖动，设计按键电路的消抖方法。
4. 在实验报告中详细报告自己的设计过程、步骤及 Verilog 代码。

顶层设计



板载 50 MHz 时钟通过 `centisecond_clock` 模块转化为周期 10 ms 的时钟供秒表主模块（`stopwatch_main`）使用。4 个按键通过消抖器（`debouncer`）消抖之后接入秒表主模块，作为秒表的控制信号。主模块将秒表状态输出到 6 个七段数码管（HEX）和 10 个发光二极管（LED）。

```
// Generate a 100Hz clock with the on-board 50MHz clock.
module centisecond_clock(refclk, outclk);
    input      refclk;
    output reg outclk;
    reg [24:0] counter;
    parameter half_max = 500000 / 2 - 1;

    // Initial block is only used for simulation, but the initial value here does not matter.
    initial begin
        counter <= 0;
        outclk <= 0;
    end

    always @(posedge refclk) begin
        if (counter >= half_max) begin
            counter <= 0;
            outclk <= ~outclk;
        end
        else
            counter <= counter + 1;
    end
endmodule
```

主模块

```
// Main module.
// Key usage:
//     key3: Reset all states (will stop current counting).
//     key2: Start / Pause / Resume counting.
//     key1: Pause display updating (but counting is still in process), display current time value and freeze.
//     key0: Resume display updating.
//     When stopped or paused, key1 and key0 will be disabled.
module stopwatch_main(clk, key3, key2, key1, key0, hex5, hex4, hex3, hex2, hex1, hex0, led);
    input         clk, key3, key2, key1, key0;
    output [6:0]  hex5, hex4, hex3, hex2, hex1, hex0;
    output [9:0]  led;
    reg    [18:0] time_counter, time_display;
    reg           counting, paused, freeze_display, key1_last;
    parameter max_time = 360000 - 1;

    initial begin
        time_counter <= 0;
        time_display <= 0;
        counting <= 0;
        paused <= 0;
        freeze_display <= 0;
        key1_last <= 1;
    end

    show_time(time_display, hex5, hex4, hex3, hex2, hex1, hex0);
    show_counting_status(time_counter, counting, led);

    // State change of time_counter.
    always @(posedge clk or negedge key3) begin
        if (!key3) begin // Match sensitive signal list in if tests.
            time_counter <= 0;
        end
        else begin // clk posedge
            if (counting && !paused)
                time_counter <= time_counter == max_time ? 0 : time_counter + 1; // Increment counter.
        end
    end

    // State change of time_display.
    always @(posedge clk or negedge key3 or negedge key1) begin
        if (!key3) begin
            time_display <= 0;
        end
        else if (!key1) begin
```

```
            if (key1 != key1_last && counting && !paused) // Make sure state of key1 changes.
                time_display <= time_counter; // Update display.
        end
        else begin // clk posedge
            if (counting && !paused && !freeze_display)
                time_display <= time_counter; // Update display.
        end
    end

    // State change of counting and paused.
    always @(negedge key3 or negedge key2) begin
        if (!key3) begin
            counting <= 0;
            paused <= 0;
        end
        else begin // key2 pressed
            if (!counting)
                counting <= 1; // Start counting.
            else
                paused <= ~paused; // Toggle pause / resume.
        end
    end

    // State change of freeze_display.
    always @(negedge key3 or negedge key1 or negedge key0) begin
        if (!key3) begin
            freeze_display <= 0;
        end
        else if (!key1) begin // Note: Key priority: when key1 is long pressed, key0 press will not be responded.
            if (counting && !paused)
                freeze_display <= 1;
        end
        else begin // key0 pressed
            if (counting && !paused)
                freeze_display <= 0;
        end
    end

    // State change of key1_last.
    always @(posedge clk)
        key1_last <= key1;
endmodule
```

主模块采用如下的几个状态变量记录秒表的当前状态：

```
reg [18:0] time_counter; // 时间计数器
reg [18:0] time_display; // 时间显示
reg        counting; // 当前是否为计时状态（计时状态/停止状态）
reg        paused; // 当前计时是否暂停（进行状态/暂停状态）
reg        freeze_display; // 是否暂停更新时间显示（显示更新/显示不更新）
```

各按键的功能如下:

- key3: 重置所有状态。（若当前为计时状态，将停止计时。）
- key2: 开始/暂停/继续计时。
- key1: 暂停时间显示更新。若显示更新已经暂停，会获取当前时刻计数器的值并显示出来。（个人认为这样的设计更符合长跑运动员的计时需要，按第一次显示第一名的成绩，按第二次显示第二名的成绩，以此类推。）
- key0: 恢复时间显示更新。
- 在计时停止或计时暂停状态下，key1 与 key0 被禁用。

时间显示

```
// Display time on sevensegs.
module show_time(time_display, hex5, hex4, hex3, hex2, hex1, hex0);
```

```
    input [18:0] time_display;
    output [6:0] hex5, hex4, hex3, hex2, hex1, hex0;

    // Minutes.
    sevenseg_decimal(time_display / 60000, hex5);
    sevenseg_decimal(time_display / 6000 % 10, hex4);

    // Seconds.
    sevenseg_decimal(time_display % 6000 / 1000, hex3);
    sevenseg_decimal(time_display / 100 % 10, hex2);

    // Centiseconds.
    sevenseg_decimal(time_display % 100 / 10, hex1);
    sevenseg_decimal(time_display % 10, hex0);
endmodule
```

主模块通过调用子模块 `show_time` 在七段数码管上显示时间。该模块使用 `time_display` 变量分别计算出时、分、秒的各位数值，并调用 `sevenseg_decimal` 七段译码模块。

```
// Show a decimal digit on a sevenseg.
module sevenseg_decimal(data, ledsegments);
    input      [3:0] data;
    output reg [6:0] ledsegments;

    always @(data)
        case (data)
            // gfe_dcba -> 654_3210
            // 1 -> off, 0 -> on
            0: ledsegments = 7'b100_0000;
            1: ledsegments = 7'b111_1001;
            2: ledsegments = 7'b010_0100;
            3: ledsegments = 7'b011_0000;
            4: ledsegments = 7'b001_1001;
            5: ledsegments = 7'b001_0010;
            6: ledsegments = 7'b000_0010;
            7: ledsegments = 7'b111_1000;
            8: ledsegments = 7'b000_0000;
            9: ledsegments = 7'b001_0000;
            default: ledsegments = 7'b111_1111; // All off on other conditions.
        endcase
endmodule
```

当前计数状态

```
// Show current status (counting / stopped) on LED.
module show_counting_status(time_counter, counting, led);
    input [18:0] time_counter;
    input        counting;
    output [9:0] led;

    // When stopped, all LEDs off.
    // When counting, LED light spot moves to the right every second.
    // When paused, LED light spot stays at the original position.
    assign led = counting ? 1 << (9 - time_counter / 100 % 10) : 0;
endmodule
```

主模块通过调用子模块 `show_counting_status` 在 LED 上显示当前的计数状态。若当前计数进行中，则 10 个 LED 中有一个亮起，且亮点每秒右移一次；若当前计数处于暂停状态，则亮点不动；若当前未在计数，则 LED 全灭。

按键消抖器

```
// Debounce a key on its posedge and negedge.
```

```
module debouncer(clk, keyin, keyout);
    input      clk, keyin;
    output reg keyout;
    reg        keyp;

    always @(posedge clk) begin // Check on every clk posedge (10ms).
        if (keyp == keyin) // Propagate value when last value and current value is consistent.
            keyout <= keyin;
        keyp <= keyin;
    end
endmodule
```

该消抖器工作需要一个 10 ms 的时钟，把一个可能抖动的按钮信号 `keyin` 转化为一个不抖的信号 `keyout`。逻辑是每 10 ms 检查一次按钮状态，若当前状态与上次相同，则输出当前状态，否则维持原状。若我们认为按钮的抖动持续时间不会超过 10 ms，并且按钮未被按下及被按住时不会发生抖动（即抖动仅发生在按下和抬起过程中），则此消抖逻辑可以正确消除抖动。

实验总结

实验结果

实验代码经过编译综合，载入到开发板后，能正常完成预期的秒表功能。按键消抖效果良好，未出现按键不响应或响应多次的现象。

经验教训

1. `initial` 语句和 `#` 时延语句只能用于仿真验证，是不可综合的。不能使用 `initial` 语句给寄存器的状态设定硬件上有效的初值（应当使用显式的 reset 信号去置入正确的初值），也不能使用 `#` 时延语句产生硬件上有效的时延。
2. 多个 `always` 语句块不能给同一个变量赋值，这样在逻辑上是可能存在冲突的，实际编译也无法通过。
3. `always` 语句块若判定多个（N 个）敏感信号的沿，主体中的 `if` 语句判断条件应与敏感信号沿列表中的 N - 1 个匹配。
4. 在本次秒表的设计中，实质上时钟信号已经比按键信号要高频得多，一个 `always @(posedge clk)` 的同步（轮询）设计实际上能处理好所有问题，设计简洁并且避免了很多麻烦。而我使用多个 `always` 语句块再检测按键的下降沿的异步（中断）设计方式其实并不是那么好。

感受

作为一名软件学院的学生，在本次实验中我亲身体验了让自己设计的逻辑直接在硬件电路上运行的过程，更清晰地理解了硬件的结构、原理及与软件的联系。同时，调试硬件也需要很多耐心，想办法定位问题之所在，必要时和老师、同学沟通能更有效地解决问题。