实验报告：基于 Verilog 和 FPGA/CPLD 的多功能秒表设计

姓名：江玓璠

学号：518021910550

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

实验仪器

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

实验详情

一、设计思路

输入部分采用了 DE1_SoC 自带的 50MHz 的时钟，在模块内部将其转换为需要的频率，每 10ms 触发一次数值的更新。另外采用了 KEY0-3 四个按钮，其功能如下：

KEY3：复位，重置所有数值；

KEY2：开始/暂停/继续计时；

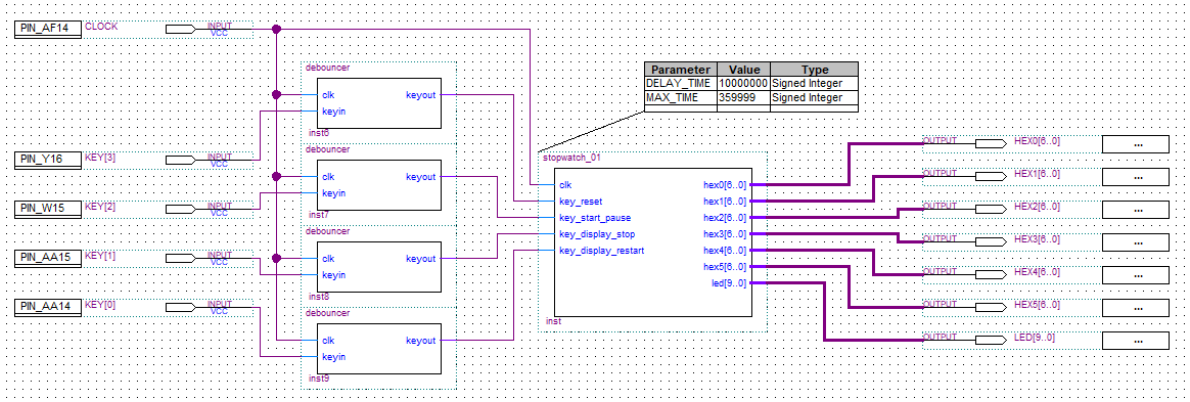
KEY1：暂停显示数值更新，但实际的计时还是在继续，连续点击会更新显示的数值，与实际应用中秒表的功能相符合（如：计时不停止，每按一次显示当前名次选手的时间）

KEY0：显示数值更新继续。

设计了消除抖动的模块，使得在 DE1_SoC 本身的防抖动的基础上进一步保证准确性，具体实现见后文；

主模块中根据按钮逻辑实时更新时间计时，并利用转换模块将累计的时间（秒数）转换为六个 LED 数码管对应显示所需要的 7 段 LED 数码管位段编号，另外还使用了 LED 发光二极管指示灯，当计时进行时，指示灯从左到右依次闪烁，暂停时暂停闪烁，停止时不闪烁，具体实现逻辑见后文和源代码。

二、顶层设计



板载 50 MHz 时钟和 4 个按钮 KEY 0-3 进入 `debouncer` 模块，消除抖动，之后输入 `stopwatch_01` 主模块，作为秒表的控制信号。主模块进行运算处理之后将时钟状态输出到 6 个 7 段 LED 数码管和 10 个 LED 发光二极管。

三、部分源代码设计

1. 时钟频率转换

```
always @(posedge clk) begin
    ...
    if (counter_50M < 500000) begin
        // 没有到 10ms
        counter_50M <= counter_50M + 1;
    end
    else
        counter_50M <= 0;
    ...
end
```

利用了简单的累加方法转换频率。由于板载时钟是 50 MHz，即每个时钟周期为 20 ns，而我们的秒表需要每 10 ms 更新一次数值，因此需要累计 500000 次累加即可转换。这一部分代码在 `debouncer` 和 `stopmatch_01` 模块中均有出现。

2. `debouncer` 消除抖动模块

```
// 按键消抖模块
module debouncer(clk, keyin, keyout);
    input clk, keyin;
    output reg keyout;
    reg keypast;
    reg [31:0] counter_50M; // 计时用计数器， 每个 50MHz 的 clock 为 20ns。
    always @(posedge clk) begin
        if (counter_50M < 500000) begin
            // 没有到 10ms
            counter_50M <= counter_50M + 1;
        end
        else begin
            // 到达 10ms
            counter_50M <= 0;
            if (keypast == keyin)
                keyout <= keyin;
            keypast <= keyin;
        end
```

```
    end
endmodule
```

为了消除抖动，使用了寄存器 `keypast` 储存 10 ms 前的按钮的值，将其与当前的输入 `keyin` 如果保持不变，则认为没有抖动，实际触发了按钮，因此改变输出 `keyout` 的值，更新 `keypast`

3. `stopwatch_01` 主模块

下面的代码都是在 `always` 语句内的逻辑；

```
always @ (posedge clk) begin
    ...
    // 按下 reset 按钮，重置所有数据
    if (!key_reset && key_reset_last != key_reset) begin
        counter_work <= 0;
        display_work <= 1;
        time_counter <= 0;
        minute_counter_high <= 0;
        minute_counter_low <= 0;
        second_counter_high <= 0;
        second_counter_low <= 0;
        msecond_counter_high <= 0;
        msecond_counter_low <= 0;
        minute_display_high <= 0;
        minute_display_low <= 0;
        second_display_high <= 0;
        second_display_low <= 0;
        msecond_display_high <= 0;
        msecond_display_low <= 0;
        led = 0;
    end
    ...
end
```

按下 `key_reset` 之后（按下后的值为 0 ）重置所有内部变量和输出变量的值，并且停止计时。

`key_reset_last != key_reset` 是为了保证一次按钮只触发一次。下面的类似逻辑的目的都是一样的。

```
if (time_counter <= MAX_TIME) begin
    time_counter <= time_counter + 1;
end
else
    time_counter <= 0;
minute_counter_high <= time_counter / 60000;
minute_counter_low <= time_counter / 6000 % 10;
second_counter_high <= time_counter % 6000 / 1000;
second_counter_low <= time_counter / 100 % 10;
msecond_counter_high <= time_counter % 100 / 10;
msecond_counter_low <= time_counter % 10;
```

`time_counter` 是累计计时数，每 10 ms 加 1；

下面的几行语句根据不同位置的数码管计算不同的数值，范围为 0-9 ；

```
// 计时期间 LED 持续闪烁
led = (1 << (9 - time_counter / 100 % 10));
```

`led` 是输出到 10 个 LED 发光二极管的值，每秒向右移动一位；

```
// 如果显示更新开启
if (display_work) begin
    minute_display_high <= minute_counter_high;
    minute_display_low <= minute_counter_low;
    second_display_high <= second_counter_high;
    second_display_low <= second_counter_low;
    msecond_display_high <= msecond_counter_high;
    msecond_display_low <= msecond_counter_low;

    // // 按下“显示暂停”暂停到当前时间
    if (!key_display_stop && key_display_stop_last != key_display_stop)
begin
        display_work <= ~display_work;
        key_display_stop_last <= key_display_stop;
    end
end
```

带有 `xxx_display_xxx` 的是指实际显示在 LED 数码管的数值，通过 `key_display_stop` 和 `key_display_restart` 可以控制其变化，当正常计时时，按下 `key_display_stop` 会暂停显示数值的变化，但是实际上的计时是还在继续的，此时按下 `key_display_restart` 可以将显示数值重新变成实际的时间；如果暂停后再次按下 `key_display_stop` 则将暂停显示的时间更新为最新的时间并保持显示暂停。

```
key_start_pause_last <= key_start_pause;
key_display_restart_last <= key_display_restart;
key_display_stop_last <= key_display_stop;
key_reset_last <= key_reset;
```

这几行代码是为了保证每一次按键只触发一次相应的功能。

```
// 4bit 的 BCD 码至 7 段 LED 数码管译码器模块
// 可供实例化共 6 个显示译码模块
module sevenseg (data, ledsegments);
    input [3:0] data;
    output ledsegments;
    reg [6:0] ledsegments;
    always @ (*)
        case(data)
        // gfe_dcba // 7 段 LED 数码管的位段编号
        // 654_3210 // DE1-SOC 板上的信号位编号
        0: ledsegments = 7'b100_0000; // DE1-SOC 板上的数码管为共阳极接法。
        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; // 其它值时全灭。
    endcase
endmodule
```

模块 `sevenseg` 是主模块调用的子模块，是用于将每一位数码管的数值转化为 7 段 LED 数码管位段编号的形式。

实验总结

实验结果

实验代码能够正常通过编译，正常载入开发板，且运行效果符合预期，按钮消除抖动效果良好，未出现按键不响应或响应多次的现象。

经验总结与反思

1. 主模块部分代码比较庞杂，实际上可以拆分成多个子模块，简化模块逻辑，使得代码可读性更高，耦合性更低。
2. 有些重复利用的代码，例如时钟频率转换的代码实际的逻辑与模块本身的逻辑无关，可以抽出来作为单独的模块，这样可以使得每个模块专注于各自的逻辑，便于调试和管理。
3. 需要考虑到按钮的信号与时钟的频率差距很大，一开始没有注意到这个问题，没有关于 `key_xxx_past` 部分的代码，导致不能正常工作；需要注意到每按下一次按钮，即使消除了抖动，高低电平都是以一段的形式出现的，因此最好是将电平变化的时刻视为按钮按下（弹起）。