基于Verilog和FPGA/CPLD的多功能秒表设计 - 实验报告

- 学号：517021910653
- 姓名：王祖来

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。

实验仪器和平台

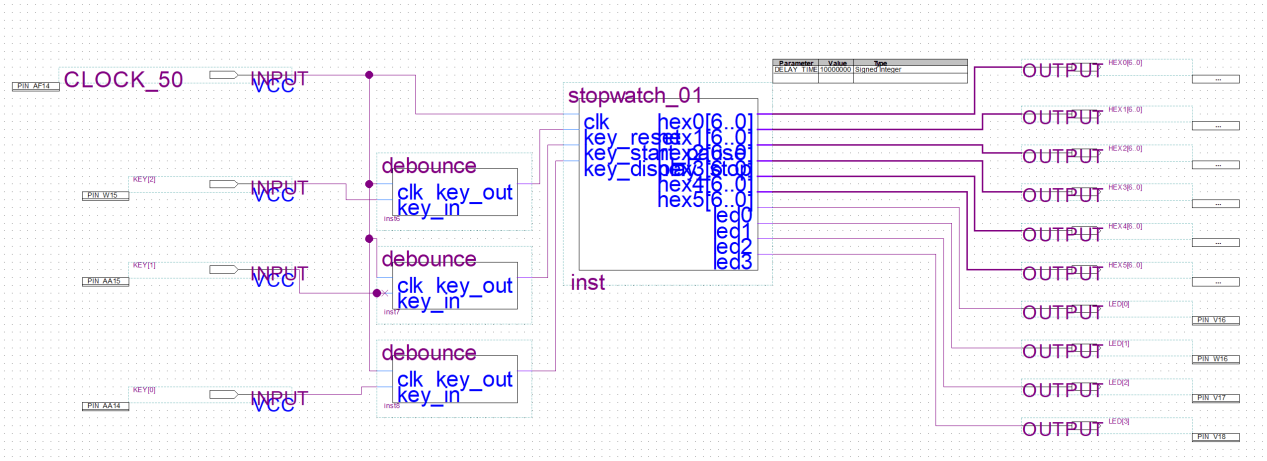
- DE1-SoC实验板
- QuartusII13.1

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。

设计过程

顶层设计



板载50MHz时钟接入秒表主模块。3个按键分别接入秒表主模块，作为秒表的控制信号。主模块将秒表状态输出到6个七段数码管和4个发光二极管

主模块

```
module stopwatch_01(clk,key_reset,key_start_pause,key_display_stop,
// 时钟输入+ 3个按键；按键按下为0 。板上利用施密特触发器做了一定消抖，效果待测试。
                hex0,hex1,hex2,hex3,hex4,hex5,
// 板上的6个7段数码管，每个数码管有7位控制信号。
                led0,led1,led2,led3 );
// LED发光二极管指示灯，用于指示/测试程序按键状态，若需要，可增加。高电平亮。
      input clk,key_reset,key_start_pause,key_display_stop;
      output [6:0] hex0,hex1,hex2,hex3,hex4,hex5;
      output led0,led1,led2,led3;
      reg led0,led1,led2,led3;

      reg display_work;
// 显示刷新，即显示寄存器的值实时 更新为计数寄存器的值。
      reg counter_work;
// 计数（计时）工作状态，由按键“计时/暂停” 控制。
      parameter DELAY_TIME = 10000000;
// 定义一个常量参数。 10000000 ->200ms；

// 定义6个显示数据（变量）寄存器：

      reg [3:0] minute_display_high;
      reg [3:0] minute_display_low;
      reg [3:0] second_display_high;
      reg [3:0] second_display_low;
      reg [3:0] msecond_display_high;
      reg [3:0] msecond_display_low;

// 定义6个计时数据（变量）寄存器：

      reg [3:0] minute_counter_high;
      reg [3:0] minute_counter_low;
      reg [3:0] second_counter_high;
```

```
      reg [3:0] second_counter_low;
      reg [3:0] msecond_counter_high;
      reg [3:0] msecond_counter_low;
      reg [31:0] counter_50M; // 计时用计数器， 每个50MHz的clock 为20ns。
// DE1-SOC板上有4个时钟， 都为 50MHz，所以需要500000次20ns之后，才是10ms。

      reg reset_1_time; // 消抖动用状态寄存器-- for reset KEY
      reg [31:0] counter_reset; // 按键状态时间计数器
      reg start_1_time; //消抖动用状态寄存器-- for counter/pause KEY
      reg [31:0] counter_start; //按键状态时间计数器
      reg display_1_time; //消抖动用状态寄存器-- for KEY_display_refresh/pause
      reg [31:0] counter_display; //按键状态时间计数器

      reg start; // 工作状态寄存器
      reg display; // 工作状态寄存器

      // init variable
      initial begin
            display_work <= 0;
            counter_work <= 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;

            minute_counter_high <= 0;
            minute_counter_low <= 0;
            second_counter_high <= 0;
            second_counter_low <= 0;
            msecond_counter_high <= 0;
            msecond_counter_low <= 0;

            reset_1_time <= 0;
            counter_reset <= 0;
            start_1_time <= 0;
            counter_start <= 0;
            display_1_time <=0;
            counter_display <= 0;

            start <= 0;
            display <= 0;
      end

// sevenseg模块为4位的BCD码至7段LED的译码器，
//下面实例化6个LED数码管的各自译码器。
      sevenseg LED8_minute_display_high ( minute_display_high, hex5 );
```

```
sevenseg LED8_minute_display_low ( minute_display_low, hex4 );

sevenseg LED8_second_display_high( second_display_high, hex3 );
sevenseg LED8_second_display_low ( second_display_low, hex2 );

sevenseg LED8_msecond_display_high( msecond_display_high, hex1 );
sevenseg LED8_msecond_display_low ( msecond_display_low, hex0 );

always @ (negedge key_start_pause) // change counter state
      begin
            counter_work <= ~counter_work;
      end

always @ (negedge key_display_stop) // change display state
      begin
            display_work <= ~display_work;
      end

always @ (posedge clk) // 每一个时钟上升沿开始触发下面的逻辑,
                  // 进行计时后各部分的刷新工作
      begin
            // reset state
            if (~key_reset) begin
                  minute_display_high <= 0;
                  minute_display_low <= 0;
                  second_display_high <= 0;
                  second_display_low <= 0;
                  msecond_display_high <= 0;
                  msecond_display_low <= 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;
            end

            // update time counter
            if (counter_work) begin
                  counter_50M <= counter_50M + 1;

                  if (counter_50M == 500000) begin
                        counter_50M <= 0;
                        msecond_counter_low <= msecond_counter_low + 1;

                        if (msecond_counter_low == 9) begin
                              msecond_counter_low <= 0;
```

```
                                    msecond_counter_high <=
msecond_counter_high + 1;

                                    if (msecond_counter_high == 9) begin
                                          msecond_counter_high <= 0;
                                          second_counter_low <=
second_counter_low + 1;

                                          if (second_counter_low == 9) begin
                                                second_counter_low <= 0;
                                                second_counter_high <=
second_counter_high + 1;

                                                if (second_counter_high == 5)
begin
                                                      second_counter_high <=
0;
                                                      minute_counter_low <=
minute_counter_low + 1;

                                                      if (minute_counter_low
== 9) begin
                                                            minute_counter_low
<= 0;
minute_counter_high <= minute_counter_high + 1;

                                                            if
(minute_counter_high == 9) begin

minute_counter_high <= 0;
                                                            end
                                                      end
                                                end
                                          end
                                    end
                              end
                        end

                        // display current time
                        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;
                        end
```

```
            end
endmodule
```

主模块中维护一个计数器，实现更新毫秒低位的数码管的周期为10ms。秒和分的各个数码管也将小一位的数码管作为计数器，实现不同周期进行数码管显示的更新。

主模块采用以下变量记录秒表当前状态：

```
// 定义6个显示数据（变量）寄存器：

      reg [3:0] minute_display_high;
      reg [3:0] minute_display_low;
      reg [3:0] second_display_high;
      reg [3:0] second_display_low;
      reg [3:0] msecond_display_high;
      reg [3:0] msecond_display_low;

// 定义6个计时数据（变量）寄存器：

      reg [3:0] minute_counter_high;
      reg [3:0] minute_counter_low;
      reg [3:0] second_counter_high;
      reg [3:0] second_counter_low;
      reg [3:0] msecond_counter_high;
      reg [3:0] msecond_counter_low;
      reg [31:0] counter_50M; // 计时用计数器， 每个50MHz的clock 为20ns。
// DE1-SOC板上有4个时钟， 都为 50MHz，所以需要500000次20ns之后，才是10ms。
```

按键功能如下：

- key[2]: key_reset，重置所有状态
- key[1]: key_start_pause，开始/暂停计时
- key[0]: key_display_stop，继续/暂停时间显示更新

七段译码模块

```
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
```

该模块将输入值译码显示在七段数码管上

消抖器

```
// implement key debounce
module debounce(clk, key_in, key_out);
  input   clk, key_in;
  output  key_out;
  reg   key_out;

  // counter during debounce procedure
  reg [31:0] cnt;
  reg key_valid;
  reg key_invalid;
  reg key_temp;

  // debounce for 500000 * 20ns = 10ms
  localparam DEBOUNCE_TIME = 500_000;

  initial begin
    key_valid <= 1;
    key_invalid <= 0;
    cnt <= 0;
    key_temp <= 1;
  end

  // output key value after debounce finished
  always @ (posedge clk)
    begin
      if (key_invalid) begin
        key_valid <= 0;
      end
      if (key_valid)
        begin
          key_out <= key_in;
          key_temp <= key_in;
        end
      else
```

```
        begin
          cnt <= cnt + 1;
          key_out <= key_temp;
          if (cnt > 500000) begin
            key_valid <= 1;
            cnt <= 0;
          end
        end
    end
  always @ (negedge key_in)
    begin
      if (key_valid) begin
        key_invalid <= 1;
      end
      else begin
        key_invalid <= 0;
      end
    end
endmodule
```

该消抖器中设计了一个周期为10ms的计时器。当按键按下时计时器启动；10ms期间，输出结果为计时开始时的按键状态；10ms结束时，将按键结果输出。在10ms以内的抖动结果不会被输出，达到消抖效果。

实验总结

实验结果

实验代码经过编译，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. verilog的阻塞赋值语句(=)与非阻塞赋值语句(<=)应区分清楚。不能在同一个always语句块中同时使用=和<=进行赋值。
2. 由于同一个always语句块的多个非阻塞赋值语句没有先后之分，并行执行。在always语句块中进行条件判断的时候不应用串行执行语句的方式写逻辑，而应该考虑上一次结束always语句块的状态。
3. 在本次秒表的设计中，将板载50MHz时钟转化为周期10ms的时钟的逻辑写在main模块中，存在一定耦合。实际上应该把时钟周期的转换单独列出一个模块，在顶层设计中进行连接，逻辑上更加合理。
4. 设计消抖器时，一开始只用一个key_valid在按键按下的always语句块和clock的always语句块进行消息传递，导致编译错误。之后引入了两个寄存器作为两个always语句的消息传递寄存器，解决了该问题。由此可见不能在两个模块同时对一个寄存器进行写操作。

感受

作为一名软件工程专业大三的学生，在本次试验中我亲身体会到让自己设计的逻辑直接运行在硬件上的过程。对硬件的结构，模块的设计，硬件与软件的接口有了更加清晰的了解。同时在实验过程中查阅用户手册、verilog语法、调试硬件需要很多耐心，在此过程中我定位问题、解决问题的能力得到了锻炼和提升。必要时和同学、老师的交流也带来了很多宝贵的经验和有效的思路。