数电实验EXP5拓展篇——时钟（秒表+闹铃）

刘永鹏 191220070 693901492@qq.com 2020.9.29

这次的实验报告的是个人写的最值的一次

用户手册在实验报告底部

1. 实验目的

在 DE-10 Standard 开发板上实现一个电子时钟，时钟要求能够显示时、 分、秒；还可以有以下功能：调整时间；闹铃（在特定时间 LED 闪烁）；秒表；等。

2. 实验原理

1. 时钟与EXP5-1类似，但是改为按时/分/秒进行划分。
2. 使用**[1:0]mode**来进行功能选择。
3. **分模块编写**：顶层文件**Clock.v**调用一下三个文件：
    1. myclock.v : 附加调时功能的时钟。
    2. alarm.v : 闹钟。
    3. stopwatch : 秒表。

3. 实验代码

顶层模块Clock.v

首先，在顶层模块中，我们定义了如下输入/输出变量：

```
module clock(
    //六个七段数码管刚好作为时分秒的显示
    //该显示是通用的，即时钟/秒表/闹铃都使这些变量显示
    //构造一多路选择器
    output reg[6:0] HEX0_sec1,
    output reg[6:0] HEX1_sec2,
    output reg[6:0] HEX2_min1,
    output reg[6:0] HEX3_min2,
    output reg[6:0] HEX4_h1,
    output reg[6:0] HEX5_h2,
    //output reg LEDR0_alarm,//LEDR[0]，闹钟输出
    output LEDR1_clock,//LEDR[9],used to test clk，该LED输出等于时钟周期
    output test,//LEDR[8],used to test second[0]，测试输出，不重要
    output alarm,//LEDR[1],//LEDR[1],闹钟输出

    input clk,//直接连在开发板上的输入
    input [1:0]mode,//SW[1:0]，调整模式
//00：时钟，01：调时间，10：秒表，11：闹钟
    //按钮调整时间hour/min/sec
    //由于我们的开发板按钮坏了两个，因此干脆使用开关代替
    input key0_secadj,//SW[9]
    input key1_minadj,//SW[8]
    input key2_houradj,//SW[7]

```

```
    //秒表
    input s_p,//SW[5]，暂停键
    input clr,//SW[6]，清零键

    //闹铃
    input off//SW[4],when 0,turn off the alarm
);
```

然后是顶层文件的内部信号：

```
//inner signal
reg [17:0]clock;//通用时钟
wire [17:0]normal_clock;//普通时钟，wire
wire [17:0]sw_clock;//秒表stopwatch，wire
wire [17:0]alarm_clock;//闹钟，wire
reg [17:0]normal_clock_reg;//普通时钟，reg
reg [17:0]sw_clock_reg;//秒表stopwatch，reg
reg [17:0]alarm_clock_reg;//闹钟，reg

//下面的变量用作分频器
reg [25:0]count_clk;
reg [22:0]count_sw;
reg clk_t;
reg clk_sw;
```

由于**模块的输出必须是wire型**，这里使用wire型数据接受模块给出的时间数据，然后将它们存到**顶层模块的寄存器中**。在always模块中完成这步操作。

根据上面所说的mode[1:0]模式选择变量，**构建多路选择器**，从三个信号中选择一个读入到通用时钟，统一在显示模块中通过七段数码管进行输出，七段数码管的模块不再放入报告中，下面是多路选择器：

```
//clock赋值,多路选择
always @ (mode)
begin
    normal_clock_reg = normal_clock;
    sw_clock_reg = sw_clock;
    alarm_clock_reg = alarm_clock;
    case(mode)
    2'b00: clock = normal_clock_reg;
    2'b01: clock = normal_clock_reg;
    2'b10: clock = sw_clock_reg;
    2'b11: clock = alarm_clock_reg;
    endcase
end
```

构建两个分频器，时钟使用1s的分频器，而秒表使用0.1s的。

```
//分频器1，每秒
always @ (posedge clk)
begin
    if(count_clk == 25000000)
    begin
        clk_t <= ~clk_t;
        count_clk <= 0;
    end
    else count_clk <= count_clk + 1;
```

```
end

//分频器2,每0.1秒
always @ (posedge clk)
begin
    if(count_sw == 2500000)
    begin
        clk_sw <= ~clk_sw;
        count_sw <= 0;
    end
    else count_sw <= count_sw + 1;
end
```

时钟模块myclock.v

输入输出信号与内部信号：

```
module myclock(
    input [1:0]mode,//when 00,set to ordinary clock mode;when 01,set to reset mode
    input clk,
    input hourkey,
    input minkey,
    input seckey,//adjust by pushbutton
    output reg[17:0]clock//时，分，秒
);
//在任何情况下计时器都应正常工作，除非在时间重设时，暂停计时
//inner signal
reg [5:0]hour;
reg [5:0]min;
reg [5:0]sec;
```

逻辑模块：

```
always @ (posedge clk)
begin
    if(mode == 2'b01) //重设时间
    begin
        if(hourkey)
            if(hour < 23)
                hour <= hour + 1;
            else hour <= 0;
        else hour <= hour;

        if(minkey)
            if(min < 59)
                min <= min + 1;
            else begin
                min <= 0;
                if(hour < 23)
                    hour <= hour + 1;
                else hour <= 0;
            end
        else min <= min;

        if(seckey)
```

```
            if(sec < 59)
                sec <= sec + 1;
            else begin
                sec <= 0;
                if(min < 59)
                    min <= min + 1;
                else begin
                    min <= 0;
                    if(hour < 23)
                        hour <= hour + 1;
                else hour <= 0;
                end
            end
        else sec <= sec;
    end

    else begin//正常时钟

        if(sec < 59)
            sec <= sec + 1;
        else begin
            sec <= 0;
            if(min < 59)
                min <= min + 1;
            else begin
                min <= 0;
                if(hour < 23)
                    hour <= hour + 1;
                else hour <= 0;
            end
        end
    end

    //返回值
    clock <= {hour,min,sec};
end
```

简要解释一下此种逻辑：

1. 时钟上升沿触发。
2. 如果mode == 2'b01，判断为调时模式，暂时暂停时钟，每一个时钟周期当前时钟根据三个开关的输入改变一次。
3. 否则，时钟正常运行。根据常理，**时钟应当在秒表计时和闹钟调时时仍然正常运行，只是不再显示**，本代码即根据此功能要求编写。
4. 问题：采用1s的时钟周期进行调时有点慢，按钮调时比较合理（但我们的开发板按钮坏了，所以用这种方法）。

秒表模块stopwatch.v

输入输出信号和内部信号：

```
module stopwatch(
    input [1:0]mode,//when 10,set to stop watch mode
    input clk,
    input start_pause,// 0->start,1->pause
    input clr,//1->clear to zero, while keep counting
    output reg[17:0]clock
);
//inner signal
reg [5:0]min;
reg [5:0]sec;
reg [5:0]msec;
```

逻辑模块：

```
always @ (posedge clk)
begin
    if(mode == 2'b10)
        if(clr == 1)
        begin
            min <= 0;
            sec <= 0;
            msec <= 0;//清零
        end
        else if(start_pause == 0)
        begin//正常时钟
            if(msec < 59)
                msec <= msec + 1;
            else begin
                msec <= 0;
                if(sec < 59)
                    sec <= sec + 1;
                else begin
                    sec <= 0;
                    if(min < 23)
                        min <= min + 1;
                    else min <= 0;
                end
            end
        end
        else if(start_pause == 1)
        begin
            min <= min;
            sec <= sec;
            msec <= msec;//保持原数据
        end
    else begin
        min <= min;
        sec <= sec;
        msec <= msec;//保持原数据
    end
    //返回值
    clock <= {min,sec,msec};
end

```

1. 秒表的逻辑类似时钟的正常计时，只是将时钟周期改为0.1s。(编写代码时，正是写好时钟后直接把代码搬过来用了，稍作修改)
2. 由清零端和暂停端两个输入信号进行控制。
3. 在调至其他模式后，秒表的**当前计时将保存**。再次打开秒表时，仍保存着上次的进度。
4. mode == 10时，进入秒表模式。

闹钟模块alarm.v

输入输出信号与内部信号

```
module alarm(
    input [1:0]mode,//when 11, set to alarm reset mode
    input clk,//when 1,alarm on
    input [17:0]cur_clock,
    input hourkey,
    input minkey,
    input en,//when 0,turn off
    input seckey,
    output reg [17:0]clock,
    output reg alarming
);
//闹钟多了一个使能（开启）开关，用于控制闹铃的开启和关闭
//该开关只控制闹铃，不会清除当前的闹钟时间
//inner signal
reg [5:0]hour;
reg [5:0]min;
reg [5:0]sec;
reg [9:0]count;
reg alarm_on;//开启该信号时，闹钟响起（LED亮）
```

逻辑模块：

```
always @ (posedge clk)//这里负责调闹钟时间
begin
    //该部分语句负责调整参数count
    //count负责定义闹钟“响起”的时间，即在手动关闭（en置为0）情况下，要响多久，此处是100s
    if(count == 1000)
    begin
        count <= 0;
    end
    else if(en && alarm_on==1)
        count <= count + 1;
    else count <= count;

    if(mode == 2'b11) //重设时间
    begin
        if(hourkey)
            if(hour < 23)
                hour <= hour + 1;
            else hour <= 0;
        else hour <= hour;

        if(minkey)
            if(min < 59)
                min <= min + 1;
```

```
            else begin
                min <= 0;
                if(hour < 23)
                    hour <= hour + 1;
                else hour <= 0;
            end
        else min <= min;

        if(seckey)
            if(sec < 59)
                sec <= sec + 1;
            else begin
                sec <= 0;
                if(min < 59)
                    min <= min + 1;
                else begin
                    min <= 0;
                    if(hour < 23)
                        hour <= hour + 1;
                else hour <= 0;
                end
            end
        else sec <= sec;
    end
    //返回值
    clock <= {hour,min,sec};
end

always @ (en)//alarming，这里负责开启“闹铃alarming”
begin
    if(en == 1 && cur_clock == {hour,min,sec})
    begin
        alarm_on <= 1;
    end
    else alarm_on <= 0;
    alarming = alarm_on;
end
```

1. 也是用0.1秒的时钟周期，当然，闹钟模块的时钟周期影响不大。
2. 额外定义alarm_on，为了解决多个always语句无法重复赋值的问题。
3. 当mode == 11时，进入调整闹钟时间模式，在其他模式下，只要en == 1，闹钟也正常运行。

附：简易用户手册

1. 七段数码管：负责显示多个模式下的时间。
2. SW[9:7]，调整时间时/分/秒，在时钟模式和闹钟模式下使用。
3. SW[6]: 秒表清零
4. SW[5]: 秒表暂停
5. SW[4]: 闹钟开启/关闭
6. LEDR[1]: 闹钟
7. LEDR[9]: 时钟周期显示

不同模式间切换不会影响数据。