

# 数电实验EXP5拓展篇——时钟（秒表+闹铃）

刘永鹏 191220070 [693901492@qq.com](mailto: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：

```
1  module clock(  
2      //六个七段数码管刚好作为时分秒的显示  
3      //该显示是通用的，即时钟/秒表/闹铃都使这些变量显示  
4      //构造一多路选择器  
5      output reg[6:0] HEX0_sec1,  
6      output reg[6:0] HEX1_sec2,  
7      output reg[6:0] HEX2_min1,  
8      output reg[6:0] HEX3_min2,  
9      output reg[6:0] HEX4_h1,  
10     output reg[6:0] HEX5_h2,  
11     //output reg LEDR0_alarm,//LEDR[0], 闹铃输出  
12     output LEDR1_clock,//LEDR[9],used to test clk, 该LED输出等于时钟周期  
13     output test,//LEDR[8],used to test second[0], 测试输出，不重要  
14     output alarm,//LEDR[1],//LEDR[1], 闹铃输出  
15  
16     input clk,//直接连在开发板上的输入  
17     input [1:0]mode,//SW[1:0], 调整模式  
18     //00: 时钟, 01: 调时间, 10: 秒表, 11: 闹铃  
19     //按钮调整时间hour/min/sec  
20     //由于我们的开发板按钮坏了两个，因此干脆使用开关代替  
21     input key0_secadj,//SW[9]  
22     input key1_minadj,//SW[8]  
23     input key2_houradj,//SW[7]  
24
```

```

25 //秒表
26 input s_p,//SW[5], 暂停键
27 input clr,//SW[6], 清零键
28
29 //闹铃
30 input off//SW[4],when 0,turn off the alarm
31 );

```

然后是顶层文件的内部信号：

```

1 //inner signal
2 reg [17:0]clock;//通用时钟
3 wire [17:0]normal_clock;//普通时钟, wire
4 wire [17:0]sw_clock;//秒表stopwatch, wire
5 wire [17:0]alarm_clock;//闹钟, wire
6 reg [17:0]normal_clock_reg;//普通时钟, reg
7 reg [17:0]sw_clock_reg;//秒表stopwatch, reg
8 reg [17:0]alarm_clock_reg;//闹钟, reg
9
10 //下面的变量用作分频器
11 reg [25:0]count_clk;
12 reg [22:0]count_sw;
13 reg clk_t;
14 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：

```

1 //clock赋值,多路选择
2 always @ (mode)
3 begin
4     normal_clock_reg = normal_clock;
5     sw_clock_reg = sw_clock;
6     alarm_clock_reg = alarm_clock;
7     case(mode)
8     2'b00: clock = normal_clock_reg;
9     2'b01: clock = normal_clock_reg;
10    2'b10: clock = sw_clock_reg;
11    2'b11: clock = alarm_clock_reg;
12    endcase
13 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。

```

1 //分频器1, 每秒
2 always @ (posedge clk)
3 begin
4     if(count_clk == 25000000)
5     begin
6         clk_t <= ~clk_t;
7         count_clk <= 0;
8     end
9     else count_clk <= count_clk + 1;

```

```

10 end
11
12 //分频器2,每0.1秒
13 always @ (posedge clk)
14 begin
15     if(count_sw == 2500000)
16     begin
17         clk_sw <= ~clk_sw;
18         count_sw <= 0;
19     end
20     else count_sw <= count_sw + 1;
21 end

```

## 时钟模块myclock.v

输入输出信号与内部信号：

```

1 module myclock(
2     input [1:0]mode,//when 00,set to ordinary clock mode;when 01,set to
    reset mode
3     input clk,
4     input hourkey,
5     input minkey,
6     input seckey,//adjust by pushbutton
7     output reg[17:0]clock//时，分，秒
8 );
9 //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
10 //inner signal
11 reg [5:0]hour;
12 reg [5:0]min;
13 reg [5:0]sec;

```

逻辑模块：

```

1 always @ (posedge clk)
2 begin
3     if(mode == 2'b01) //重设时间
4     begin
5         if(hourkey)
6             if(hour < 23)
7                 hour <= hour + 1;
8             else hour <= 0;
9         else hour <= hour;
10
11         if(minkey)
12             if(min < 59)
13                 min <= min + 1;
14             else begin
15                 min <= 0;
16                 if(hour < 23)
17                     hour <= hour + 1;
18                 else hour <= 0;
19             end
20         else min <= min;
21
22         if(seckey)

```

```

23         if(sec < 59)
24             sec <= sec + 1;
25         else begin
26             sec <= 0;
27             if(min < 59)
28                 min <= min + 1;
29             else begin
30                 min <= 0;
31                 if(hour < 23)
32                     hour <= hour + 1;
33             else hour <= 0;
34             end
35         end
36     else sec <= sec;
37 end
38
39 else begin//正常时钟
40
41     if(sec < 59)
42         sec <= sec + 1;
43     else begin
44         sec <= 0;
45         if(min < 59)
46             min <= min + 1;
47         else begin
48             min <= 0;
49             if(hour < 23)
50                 hour <= hour + 1;
51             else hour <= 0;
52         end
53     end
54 end
55
56 //返回值
57 cclock <= {hour,min,sec};
58 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：

```

1  module stopwatch(
2      input [1:0]mode,//when 10,set to stop watch mode
3      input clk,
4      input start_pause,// 0->start,1->pause
5      input clr,//1->clear to zero, while keep counting
6      output reg[17:0]clock
7  );
8  //inner signal
9  reg [5:0]min;
10 reg [5:0]sec;
11 reg [5:0]msec;

```

逻辑模块:

```

1  always @ (posedge clk)
2  begin
3      if(mode == 2'b10)
4          if(clr == 1)
5              begin
6                  min <= 0;
7                  sec <= 0;
8                  msec <= 0;//清零
9              end
10         else if(start_pause == 0)
11             begin//正常时钟
12                 if(msec < 59)
13                     msec <= msec + 1;
14                 else begin
15                     msec <= 0;
16                     if(sec < 59)
17                         sec <= sec + 1;
18                     else begin
19                         sec <= 0;
20                         if(min < 23)
21                             min <= min + 1;
22                         else min <= 0;
23                     end
24                 end
25             end
26         else if(start_pause == 1)
27             begin
28                 min <= min;
29                 sec <= sec;
30                 msec <= msec;//保持原数据
31             end
32         else begin
33             min <= min;
34             sec <= sec;
35             msec <= msec;//保持原数据
36         end
37         //返回值
38         clock <= {min,sec,msec};
39     end
40

```

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

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

逻辑模块：

```
1 always @ (posedge clk)//这里负责调闹钟时间  
2 begin  
3     //该部分语句负责调整参数count  
4     //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  
5     100s  
6     if(count == 1000)  
7     begin  
8         count <= 0;  
9     end  
10    else if(en && alarm_on==1)  
11        count <= count + 1;  
12    else count <= count;  
13  
14    if(mode == 2'b11) //重设时间  
15    begin  
16        if(hourkey)  
17            if(hour < 23)  
18                hour <= hour + 1;  
19            else hour <= 0;  
20        else hour <= hour;  
21  
22        if(minkey)  
23            if(min < 59)  
24                min <= min + 1;
```

```

24         else begin
25             min <= 0;
26             if(hour < 23)
27                 hour <= hour + 1;
28             else hour <= 0;
29         end
30     else min <= min;
31
32     if(seckey)
33         if(sec < 59)
34             sec <= sec + 1;
35         else begin
36             sec <= 0;
37             if(min < 59)
38                 min <= min + 1;
39             else begin
40                 min <= 0;
41                 if(hour < 23)
42                     hour <= hour + 1;
43                 else hour <= 0;
44             end
45             end
46         else sec <= sec;
47     end
48     //返回值
49     clock <= {hour,min,sec};
50 end
51
52 always @ (en)//alarming, 这里负责开启“闹铃alarming”
53 begin
54     if(en == 1 && cur_clock == {hour,min,sec})
55     begin
56         alarm_on <= 1;
57     end
58     else alarm_on <= 0;
59     alarming = alarm_on;
60 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。