2020-9-28

数字电路实验五 计数器和时钟

秦嘉余

191220088

1348288404@qq.com

目录

1.实验目的 1
2.实验原理 1
3.实验环境/器材 1
4. 基础实验 2
    1.设计思路 2
    2. 实验代码 2
    3.RTL 视图 3
    4.引脚分配图 3
    5.实验结果 3
5. 拓展实验 4
    1.设计思路 4
    2.实验代码 4
    3.RTL 视图 6
    4.引脚分配图 7
    5.实验结果 7
6.实验实验中遇到的问题及解决办法 7

1.实验目的

本次实验的目的是利用开发板上的时钟完成一个计数器与多功能时钟，同时复习时序电路的特点，熟悉 verilog 语言的多模块写法

2.实验原理

加法计数器和减法计数器都是数字电路中非常常见的原件，由多个 D 触发器组成，而在 verilog 中，这种器件很好实现，只需要写出逻辑代码，大部分任务都由编译器完成。同时，开发板上带有时钟信号，利用这个是时钟作为我们的初始时钟信号即可完成很多功能。开发板上的时钟器件为 50MHz 的时钟，所以我们需要使用分频器变为我们需要的时钟频率的时钟信号。有了我们需要的时钟信号，即可不断拓展完成我们的实验。

3.实验环境/器材

本次实验的环境为 Quartus17.1 版本
本次实验的器材为 DE10 Standard 开发板

4. 基础实验

1.设计思路

我们的开发板上的时钟信号为 50MHz 的时钟，所以我们需要使用分频器来将该时钟信号变为 1Hz 的时钟信号。Always 语句来检测 clk 时钟的信号，不断在计数 count 上加一作为秒数，当最大为 99 时变为 0，将 count 显示在 2 个 7 段显示管上。除此之外，设置一个使能端 en，由 str，pas 控制，完成计数器的开始与暂停，最后在 clk 的 always 语句中加入 clr 为敏感信号，来为 coun 清零

2. 实验代码

模块名为 Timer，包含输入 clk，str，pas，clr 分别为时钟信号，开始，暂停，清零，输出有 clk_1s，of，HEX0，HEX1 分别为输出的 1s 的时钟信号，计数到 99 的标志，两个七段显示管。同时还有中间变量 count，count_clk，en 分别为计数，50MHz 的计数，暂停开始使能端

```
module Timer(clk,str,pas,clr,of,clk_1s,HEX0,HEX1);
    input clk;
    input str,pas,clr;
    output reg clk_1s,of;
    output reg [6:0] HEX0;
    output reg [6:0] HEX1;
    reg [7:0] count;
    reg [24:0] count_clk;
    reg en;
```

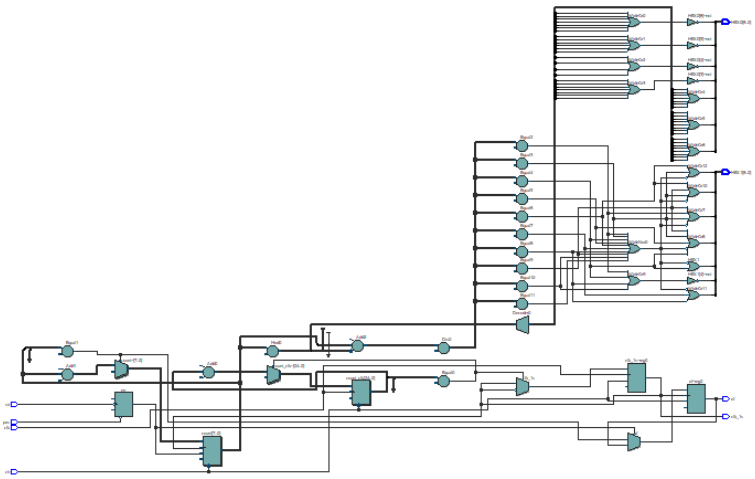
clr 与 clk 都会改变 count_clk 由于 verilog 的限制，必须在同一 always 语句中

```
  always @(posedge clk or negedge clr)
  begin
        if(!clr)
           count_clk<=0;
        else if(count_clk==25000000)
        begin
           clk_1s<=~clk_1s;
           count_clk<=0;
        end
        else
           count_clk<=count_clk+1;
  end
```

pas 与 str 会改变 en 的大小

```
  always @(negedge str or negedge pas)
  begin
     if(!pas)
        en<=0;
     else
        en<=1;
  end
```

3.RTL 视图



4.引脚分配图

HEX0[6]	Output	PIN_AH18
HEX0[5]	Output	PIN_AG18
HEX0[4]	Output	PIN_AH17
HEX0[3]	Output	PIN_AG16
HEX0[2]	Output	PIN_AG17
HEX0[1]	Output	PIN_V18
HEX0[0]	Output	PIN_W17
HEX1[6]	Output	PIN_V17
HEX1[5]	Output	PIN_AE17
HEX1[4]	Output	PIN_AE18
HEX1[3]	Output	PIN_AD17
HEX1[2]	Output	PIN_AE16
HEX1[1]	Output	PIN_V16
HEX1[0]	Output	PIN_AF16
clk	Input	PIN_AF14
clk_1s	Output	PIN_AA24
clr	Input	PIN_AA14
of	Output	PIN_AB23
pas	Input	PIN_AK4
str	Input	PIN_AJ4

5.实验结果

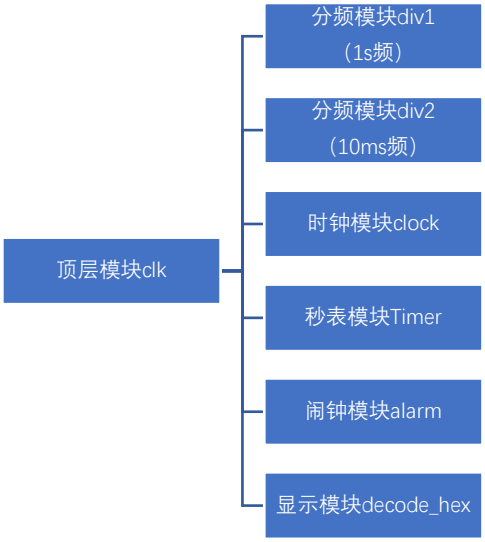
已在开发板上验收

5. 拓展实验

1.设计思路

拓展试验的原理其实大致上与基础实验一致，都是先分频，利用这个时钟来拓展一系列操作，但在本实验中，需要拓展的部分较多，首先，数码显示管需要显示的内容有三种，分别为时钟，秒表与闹钟，这就需要用开关来切换，同时每个功能都是同时运行的的，互不干扰，只是显示哪一个的问题，但是调时或者开始暂停清零按钮却只在显示该功能时有效，所以为了完成这个复杂的功能，分模块是很有必要的。

模块如下图所示：



其中 1s 频供时钟模块与闹钟模块使用，1ms 模块供秒表使用；时钟，秒表，闹钟频产生的 count 给 decode_hex 使用来输出

2.实验代码

顶层模块如下：

```
//wire clk_1s;
wire [19:0] count_1,count_2,count_3,count;
wire [6:0] hex_num;
wire [20:0] d;


    div #25000000 div1(CLOCK2_50,LEDR[0]);
    div #250000 div2(CLOCK2_50,LEDR[1]);
    clock clock1(LEDR[0],SW[1:0],KEY[0],KEY[1],KEY[2],count_1);
    Timer timer1(LEDR[1],SW[1:0],KEY[0],KEY[1],KEY[2],count_2);
    alarm alarm1(LEDR[0],SW[1:0],KEY[0],KEY[1],KEY[2],count_3);


assign count=(SW[1])?count_3:((SW[0])?count_2:count_1);
assign hex_num=(SW[1])?60:((SW[0])?100:60);

assign d=count_1-count_3;
assign LEDR[2]=(((d>=0&&d<=10)||(d>=-86400&&d<=-86390))&&SW[2]);

decode_hex hex1(hex_num,count,HEX0,HEX1,HEX2,HEX3,HEX4,HEX5);
```

其中：

count_1,count_2,count_3 分别为三个功能产生的计数，count 为最终计数，由 assign 语句来根据选择端来赋值

hex_num 为 hex 显示的低两位起端显示管的进位，因为时钟与闹钟最低位为 60 进位，而秒表为 100 进位

d 为时钟时间与闹钟时间的差，用来使得闹钟触发灯亮显示一段时间

SW[2]决定闹钟是否开启

div 模块的参数分别赋值为不同值产生两个不同的时钟信号 1s 与 10ms 给时钟闹钟与秒表使用

clock，timer，alarn 模块传入时钟与选择端，控制端，给出计数输出 counr_x

其中 SW[1:0]为 0 时显示时钟，1 时显示秒表，2 时显示闹钟，但这只是显示的功能，所有功能在选择端不为当前功能时仍然在运行，这符合多功能时钟的特性

显示模块如下:

```
module decode_hex(hex_num,count,HEX0,HEX1,HEX2,HEX3,HEX4,HEX5);
   input [6:0] hex_num;
   input [19:0] count;
   output reg [6:0] HEX0,HEX1,HEX2,HEX3,HEX4,HEX5;

   always @(count)
   begin
      case (count%10)
         0: HEX0=7'b1000000;
         1: HEX0=7'b1111001;
         2: HEX0=7'b0100100;
         3: HEX0=7'b0110000;
         4: HEX0=7'b0011001;
         5: HEX0=7'b0010010;
         6: HEX0=7'b0000010;
         7: HEX0=7'b1111000;
```

分频模块如下:

```
module div(clk,clk_1s);
    parameter max=25000000;
    input clk;
    output reg clk_1s;
    reg [24:0] count_clk;

    initial begin
        count_clk=0;
        clk_1s=0;
    end

    always @(posedge clk)
    begin
        if(count_clk==max)
        begin
            clk_1s<=~clk_1s;
            count_clk<=0;
        end
        else
            count_clk<=count_clk+1;
    end


endmodule
```

其余模块与计数器类似，只不过拓展不同的功能，由于篇幅限制，只展示 alarm 模块

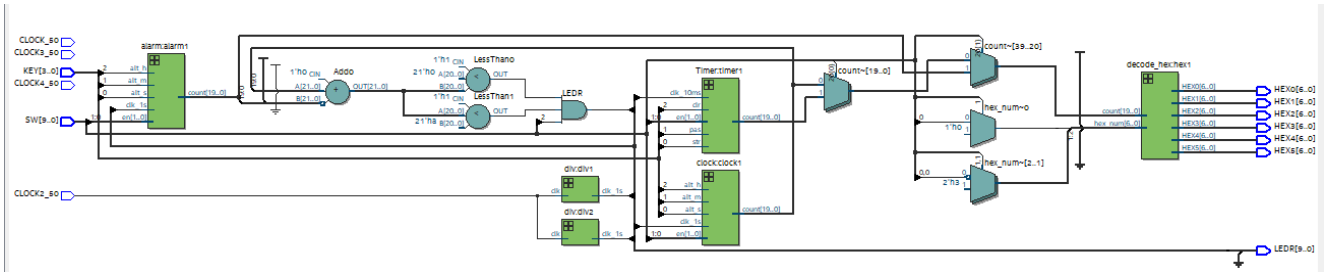
```
module alarm(clk_1s,en,alt_s,alt_m,alt_h,count);
    input  clk_1s;
    input [1:0] en;
    input alt_s,alt_m,alt_h;
    output reg [19:0] count;

    initial begin
    count=0;
    end

    always @(posedge clk_1s)
    begin
        if(!alt_h)
        begin
            if(en==2||en==3)
                count<=count+3600;
        end
        else  if(!alt_m)
        begin
            if(en==2||en==3)
                count<=count+60;
        end
        else  if(!alt_s)
        begin
            if(en==2||en==3)
                count<=count+1;
        end
        else if(count>86398)
        begin
            count<=count-86399;
        end
        else
        begin
            count<=count;
        end

    end
```

3.RTL 视图



每个模块内部的 RTL 视图由于篇幅不再展示

4.引脚分配图

本项目由 build 软件自动生成，则不再展示引脚分配图

5.实验结果

已在开发板上验收

6.实验实验中遇到的问题及解决办法

本次实验由于较为复杂，遇到了很多问题，大多为对 verilog 是一门硬件语言理解不足，许多在软件语言中显而易见的语句在 verilog 中不能成立，所以发生了很多语法问题，下列举几个：
Question1:
Always 语句中不能存在 @(clk or negedge clr)这种含有上升下降沿和下降沿的形式

Question2:
多个 always 语句不能对同一个变量赋值

Question3:
多个变化的 always 中必须出现最后一个变化沿的判断 if 语句

这些都是与硬件息息相关的内容，例如多个 always 不能对一个变量赋值，这符合硬件电路中不能给一个输出多个接多条线的原则
所以在写 verilog 文件时，需要时时刻刻注意这是一门硬件语言，避免犯语法错误