STEP-MAX10 快速入门

小脚丫 STEP FPGA

STEP

2018/9/5

STEP-MAX10 快速入门

目录

小脚丫 STEP-MAX10.....3
实验一　点亮 LED.....3
实验二　RGB LED 与光源三基色的混合.....7
实验三　3-8 译码器.....9
实验四　数码管显示.....12
实验五　时钟分频.....16
实验六　LED 流水灯.....22
实验七　按键消抖.....26
实验八　计时控制.....32
实验九　脉宽调制.....37
实验十　简易交通灯.....41

小脚丫 STEP-MAX10

本文档为 FPGA 初学者入门项目，以 STEP MAX10-08C（核心芯片 10M08SCM153）版本核心板硬件为例，使用 Verilog 作为编程语言，目的是让初学者快速了解基本的逻辑实现和 FPGA 的编程过程，我们由浅入深，从如何点亮 LED、数码管到利用状态机去完成交通灯的设计。开发板具体的硬件信息请查看文档 STEP-MAX10 硬件手册。

本文档的所有程序都是基于 STEP MAX10-08C 版本（核心芯片 10M08SCM153），其他的小脚丫 STEP MAX10 系列版本都是管脚兼容，因此只需要修改程序的芯片设置就能够使用。

实验一　点亮 LED

在这个系列教程里你将更深入学习 FPGA 的设计同时更深入了解我们的小脚丫。如果你还没有开始使用小脚丫，也可以从这里一步一步开始你的可编程逻辑学习。请先准备好软硬件文档，因为 FPGA 的设计是和硬件息息相关，会经常用到这些文档。你还必须先安装好 Quartus Prime 设计工具，这是用小脚丫 STEP-MAX10 必须用到的。 对于很多刚刚接触电子硬件的人来说，第一件事就是点亮 LED，在这个例程里，将带您实现这一功能。

1. 硬件说明

STEP-MAX10 开发板虽然很小巧，上面也集成了不少外设，在本实验里我们就看看如何用 FPGA 控制简单外设，如何用按键或者开关控制 LED 的亮和灭。

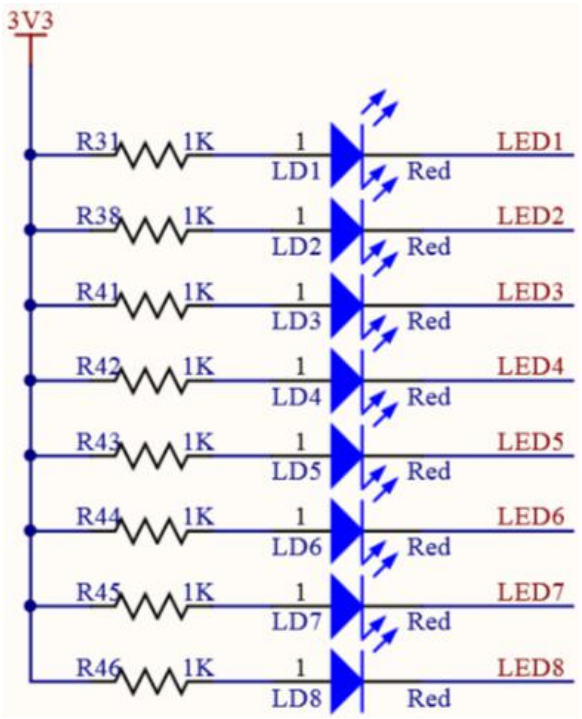


图 1-1　STEP-MAX10 板载 LED 硬件电路图

这是开发板上的 8 个红色 LED，LED1~8 信号连接到 FPGA 的引脚，作为 FPGA 输出信号控制。当 FPGA 输出低电平时 LED 变亮，当 FPGA 输出高电平时 LED 熄灭。

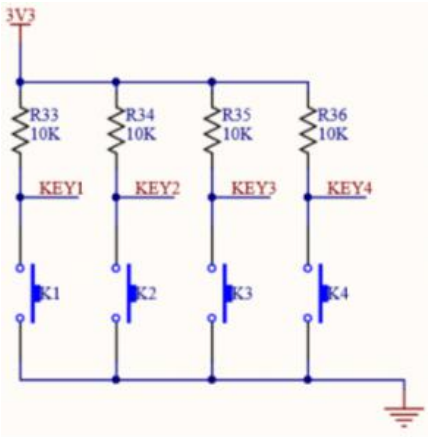


图 1-2　STEP-MAX10 板载轻触按键硬件电路图

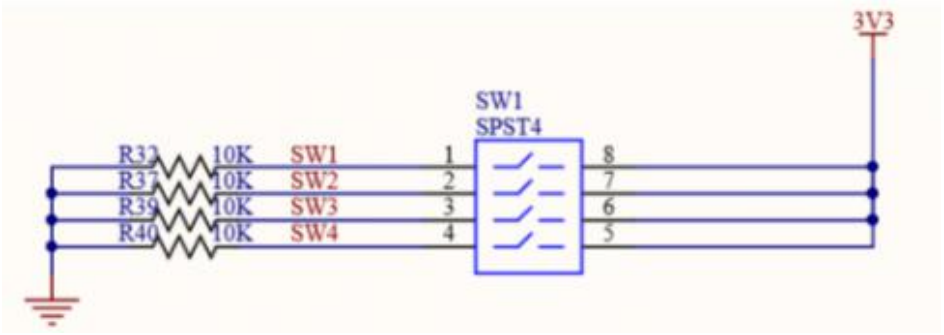


图 1-3　STEP-MAX10 板载拨码开关硬件电路图

这是开发板上 4 个按键和 4 个开关，Key1~4 是按键控制信号，SW1~4 是开关控制信号，都连接到 FPGA 的引脚，作为 FPGA 的输入信号。当按键断开时，FPGA 输入为高电平，当按键按下时，FPGA 输入为低电平；当开关断开（OFF）时，FPGA 输入为低电平，当开关合上（ON）时，FPGA 输入为高电平。

所以我们可以用开关或者按键来控制 LED 的亮灭。

2. Verilog 代码

```
module LED (key,sw,led);

    input [3:0] key;              //按键输入信号
    input [3:0] sw;               //开关输入信号
    output [7:0] led;             //输出信号到LED

    assign led = {key,sw};        //assign连续赋值。大括号是拼接符，表示把
key和sw拼接组成一个新的8位数赋值给led

endmodule
```

3. 引脚分配

综合(synthesize)完成之后一定要配置 FPGA 的引脚到相应的外设，这样下载 FPGA 程序后才能达到我们想要的效果。

信号名称	分配管脚	信号名称	分配管脚
LED[0]	N15	**SW[0]**	J12
LED[1]	N14	**SW[1]**	H11
LED[2]	M14	**SW[2]**	H12
LED[3]	M12	**SW[3]**	H13
LED[4]	L15	**KEY[0]**	J9
LED[5]	K12	**KEY[1]**	K14
LED[6]	L11	**KEY[2]**	J11
LED[7]	K11	**KEY[3]**	J14

4. 小结

下载完程序后就可以实现按键开关控制 LED 灯的亮灭。了解小脚丫 STEP-MAX10 V2 上的外设 LED、按键和开关的使用。

实验二　RGB LED 与光源三基色的混合

在这个实验里我们将学习控制小脚丫 STEP-MAX10 上的 RGB 三色 LED 的显示，基本的原理和点亮 LED 是相似的。

1. 硬件说明

STEP-MXO2 V2 开发板上面有两个三色 LED，我们也可以用按键或者开关控制三色 LED 的显示。

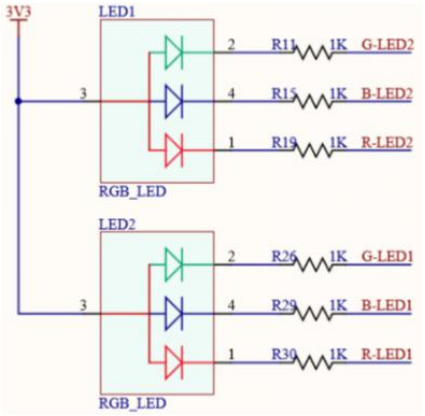


图 2-1　STEP-MAX10 板载 LED 硬件电路图

这是开发板上的 2 个三色 LED，采用的是共阳极的设计，RGB 三种信号分别连接到 FPGA 的引脚，作为 FPGA 输出信号控制。当 FPGA 输出低电平时 LED 变亮，当 FPGA 输出高电平时 LED 熄灭，当两种或者三种颜色变亮时会混合出不同颜色，一共能产生 8 种颜色。

2. Verilog 代码

```
module rgb_led (sw,led);

    input [2:0] sw;                    //开关输入信号，利用了其中3个开关
    output [2:0] led;                  //输出信号到RGB LED

    assign led = sw;                   //assign连续赋值。

endmodule
```

3. 引脚分配

综合(synthesize)完成之后一定要配置 FPGA 的引脚到相应的外设，这样下载 FPGA 程序后才能达到我们想要的效果。

信号名称	分配管脚	信号名称	分配管脚
LED[0]	G15	**SW[0]**	J12
LED[1]	E15	**SW[1]**	H11
LED[2]	E14	**SW[2]**	H12

下载完程序后就可以实现 3 个开关控制三色 LED 灯的不同颜色显示，PS：小心比较刺眼。

4. 小结

了解小脚丫 STEP-MAX10 上的外设三色 LED。之前实验都是开关和按键直接控制 LED，在下一个实验 3-8 译码器将学习如何用组合逻辑实现控制 LED 显示。

实验三　3-8 译码器

在这个实验里我们将学习如何用 Verilog 来实现组合逻辑。

1. 硬件说明

组合逻辑电路是数字电路的重要部分，电路的输出只与输入的当前状态相关的逻辑电路，常见的有选择器、比较器、译码器、编码器、编码转换等等。在本实验里以最常见的 3-8 译码器为例说明如何用 Verilog 实现。3-8 译码器的真值表如下：

A2	A1	A0	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
0	0	0	0	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	1	1
0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0

图 3-1　3-8 译码器真值表

从前面的实验可以知道，当 FPGA 输出信号到 LED 为高电平时 LED 熄灭，反之 LED 变亮。同时我们可以以开关的信号模拟 3-8 译码器的输入，这样控制开关我们就能控制特定的 LED 变亮。

2. Verilog 代码

```
module decode38 (sw,led);

    input [2:0] sw;   //开关输入信号，利用了其中3个开关作为3-8译码器的输入
    output [7:0] led; //输出信号控制特定LED
    //定义led为reg型变量，在always过程块中只能对reg型变量赋值
    reg [7:0] led;

    //always过程块，括号中sw为敏感变量，当sw变化一次执行一次always中所有语句，
否则保持不变
    always @ (sw)
    begin
        case(sw)        //case语句，一定要跟default语句
                        //条件跳转，其中"_"下划线为了阅读方便，无实际意义
                        //位宽'进制+数值是Verilog里常数的表达方法，进制可//
                        以是b、o、d、h（二、八、十、十六进制）
            3'b000:     led=8'b0111_1111;
            3'b001:     led=8'b1011_1111;
            3'b010:     led=8'b1101_1111;
            3'b011:     led=8'b1110_1111;
            3'b100:     led=8'b1111_0111;
            3'b101:     led=8'b1111_1011;
            3'b110:     led=8'b1111_1101;
            3'b111:     led=8'b1111_1110;
            default: ;
        endcase
    end
endmodule
```

3. 引脚分配

综合(synthesize)完成之后一定要配置 FPGA 的引脚到相应的外设，这样下载 FPGA 程序后才能达到我们想要的效果。

信号名称	分配管脚	信号名称	分配管脚
LED[0]	N15	**SW[0]**	J12
LED[1]	N14	**SW[1]**	H11

LED[2]	M14	**SW[2]**	H12
LED[3]	M12		
LED[4]	L15		
LED[5]	K12		
LED[6]	L11		
LED[7]	K11		

下载完程序后就可以实现 3 个开关控制不同 LED 灯的显示，3-8 译码器完成。

4. 小结

实现了一个简单的组合逻辑 3-8 译码器，在下一个数码管显示实验我们将学习如何通过译码实现控制数码管的显示。

实验四　数码管显示

本实验将会让你熟悉小脚丫上最后一种有意思的外设七段数码管。

1. 硬件说明

数码管是工程设计中使用很广的一种显示输出器件。一个 7 段数码管（如果包括右下的小点可以认为是 8 段）分别由 a、b、c、d、e、f、g 位段和表示小数点的 dp 位段组成。实际是由 8 个 LED 灯组成的，控制每个 LED 的点亮或熄灭实现数字显示。通常数码管分为共阳极数码管和共阴极数码管，结构如下图所示：

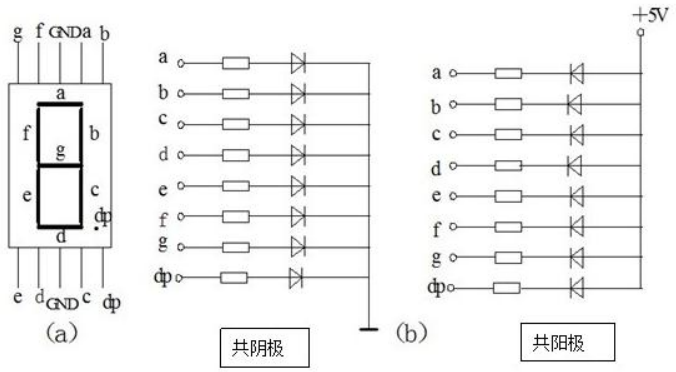


图 4-1　数码管内部结构图

共阴 8 段数码管的信号端低电平有效，而共阳端接高电平有效。当共阳端接高电平时只要在各个位段上加上相应的低电平信号就可以使相应的位段发光。比如：要使 a 段发光，则在 a 段信号端加上低电平即可。共阴极的数码管则相反。 可以看到数码管的控制和 LED 的控制有相似之处，在小脚丫 STEP-Max10 开发板上有两位共阴极数码管，

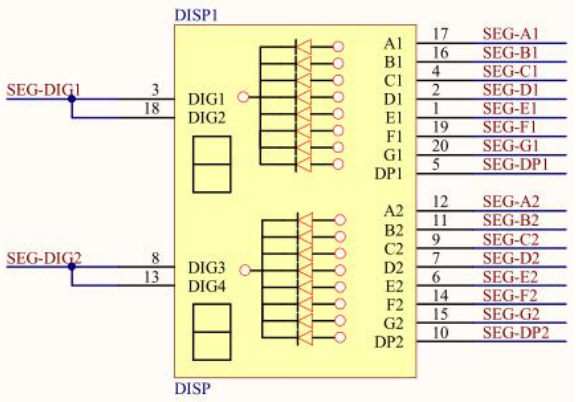


图 4-2　小脚丫 STEP MAX10 板载数码管电路图

数码管所有的信号都连接到 FPGA 的管脚，作为输出信号控制。FPGA 只要输出这些信号就能够控制数码管的那一段 LED 亮或者灭。这样我们可以通过开关来控制 FPGA 的输出，和 3-8 译码器实验一样，通过组合逻辑的输出来控制数码管显示数字，

下面是数码管显示的表格:

输入码				输出码（共阴极）							字型
A_3	A_2	A_1	A_0	g	f	e	d	c	b	a	
0	0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	0	0	0	1	1	0	1
0	0	1	0	1	0	1	1	0	1	1	2
0	0	1	1	1	0	0	1	1	1	1	3
0	1	0	0	1	1	0	0	1	1	0	4
0	1	0	1	1	1	0	1	1	0	1	5
0	1	1	0	1	1	1	1	1	0	1	6
0	1	1	1	0	0	0	0	1	1	1	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	0	1	1	1	1	9
1	0	1	0	1	1	1	0	1	1	1	A
1	0	1	1	1	1	1	1	1	0	0	B
1	1	0	0	0	1	1	1	0	0	1	C
1	1	0	1	1	0	1	1	1	1	0	D
1	1	1	0	1	1	1	1	0	0	1	E
1	1	1	1	1	1	1	0	0	0	1	F

这其实是一个 4-16 译码器，如果我们想数码管能显示 16 进制可以全译码，如果只想显示数字，可以只利用其中 10 个译码，下面看看如果用 Verilog 来实现。

2. Verilog 代码

```
module segment (seg_data_1,seg_data_2,seg_led_1,seg_led_2);

    input [3:0] seg_data_1;  //数码管需显示0~9，故需要4位输入做译码
    input [3:0] seg_data_2;  //小脚丫上第二个数码管
    output [8:0] seg_led_1;          //在小脚丫上控制一个数码管需要9个信
号 MSB~LSB=DIG、DP、G、F、E、D、C、B、A
    output [8:0] seg_led_2;          //在小脚丫上第二个数码管的控制信号
MSB~LSB=DIG、DP、G、F、E、D、C、B、A

    reg [8:0] seg [9:0];              //定义了一个reg型的数组变量，相当于
一个10*9的存储器，存储器一共有10个数，每个数有9位宽

    initial  //在过程块中只能给reg型变量赋值，Verilog中有两种过程块always
和initial
    //initial和always不同，其中语句只执行一次
        begin
          seg[0] = 9'h3f;              //对存储器中第一个数赋值
9'b00_0011_1111,相当于共阴极接地，DP点变低不亮，7段显示数字  0
          seg[1] = 9'h06;                  //7段显示数字  1
          seg[2] = 9'h5b;                  //7段显示数字  2
          seg[3] = 9'h4f;                  //7段显示数字  3
          seg[4] = 9'h66;                  //7段显示数字  4
          seg[5] = 9'h6d;                  //7段显示数字  5
          seg[6] = 9'h7d;                  //7段显示数字  6
          seg[7] = 9'h07;                  //7段显示数字  7
          seg[8] = 9'h7f;                  //7段显示数字  8
          seg[9] = 9'h6f;                  //7段显示数字  9
       end
        assign seg_led_1 = seg[seg_data_1];
//连续赋值，这样输入不同四位数，就能输出对于译码的9位输出
        assign seg_led_2 = seg[seg_data_2];

endmodule
```

3. 引脚分配

综合(synthesize)完成之后一定要配置 FPGA 的引脚到相应的外设，这样下载 FPGA 程序后才能达到我们想要的效果。小脚丫上正好有 4 路按键和 4 路开关，可以用来作为输入信

号分别控制数码管的输出。按照下面表格定义输入输出信号：

信号	引脚	信号	引脚
seg_data_1(0)	J12	**seg_data_2(0)**	J9
seg_data_1(1)	H11	**seg_data_2(1)**	K14
seg_data_1(2)	H12	**seg_data_2(2)**	J11
seg_data_1(3)	H13	**seg_data_2(3)**	J14

信号	引脚	信号	引脚
seg_led_1(0)	E1	**seg_led_2(0)**	A3
seg_led_1(1)	D2	**seg_led_2(1)**	A2
seg_led_1(2)	K2	**seg_led_2(2)**	P2
seg_led_1(3)	J2	**seg_led_2(3)**	P1
seg_led_1(4)	G2	**seg_led_2(4)**	N1
seg_led_1(5)	F5	**seg_led_2(5)**	C1
seg_led_1(6)	G5	**seg_led_2(6)**	C2
seg_led_1(7)	L1	**seg_led_2(7)**	R2
seg_led_1(8)	E2	**seg_led_2(8)**	B1

配置好以后编译下载程序。这样可以通过按键或者开关来控制相应的数码管显示数字。如果你想显示 16 进制的 AbCDeF 在数码管，可以试试修改程序。这时候一定要定义一个 16*9 的存储器来初始化。

4. 小结

了解了小脚丫数码管的工作原理，在下个实验我们将进行到有趣的时序逻辑。首先是如何控制时钟分频。

实验五　时钟分频

在之前的实验中我们已经熟悉了小脚丫的各种外设，掌握了 verilog 的组合逻辑设计，接下来我们将学习时序逻辑的设计。

1. 硬件说明

时钟信号的处理是 FPGA 的特色之一，因此分频器也是 FPGA 设计中使用频率非常高的基本设计之一。一般在 FPGA 中都有集成的锁相环可以实现各种时钟的分频和倍频设计，但是通过语言设计进行时钟分频是最基本的训练，在对时钟要求不高的设计时也能节省锁相环资源。在本实验中我们将实现任意整数的分频器，分频的时钟保持 50%占空比。

偶数分频：偶数倍分频相对简单，比较容易理解。通过计数器计数是完全可以实现的。如进行 N 倍偶数分频，那么通过时钟触发计数器计数，当计数器从 0 计数到 N/2-1 时，输出时钟进行翻转，以此循环下去。

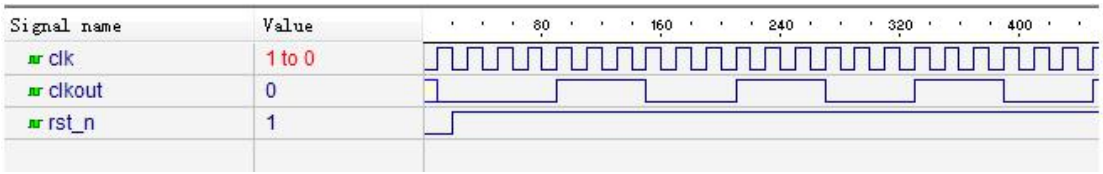


图 4-1　偶数分频时序图

奇数分频：　如果要实现占空比为 50%的奇数倍分频，不能同偶数分频一样计数器记到一半的时候输出时钟翻转，那样得不到占空比 50%的时钟。以待分频时钟 CLK 为例，如果以偶数分频的方法来做奇数分频，在 CLK 上升沿触发，将得到不是 50%占空比的一个时钟信号（正周期比负周期多一个时钟或者少一个时钟）；但是如果在 CLK 下降沿也触发，又得到另外一个不是 50%占空比的时钟信号，这两个时钟相位正好相差半个 CLK 时钟周期。通过这两个时钟信号进行逻辑运算我们可以巧妙的得到 50%占空比的时钟。

总结如下：对于实现占空比为 50%的 N 倍奇数分频，首先进行上升沿触发进行模 N 计数，计数选定到某一个值进行输出时钟翻转，然后经过（N-1）/2 再次进行翻转得到一个占空比非 50%奇数 n 分频时钟。再者同时进行下降沿触发的模 N 计数，到和上升沿触发输出时钟翻转选定值相同值时，进行输出时钟时钟翻转，同样经过（N-1）/2 时，输出时钟再次翻转生成占空比非 50%的奇数 n 分频时钟。两个占空比非 50%的 n 分频时钟进行逻辑运算（正周期多的相与，负周期多的相或），得到占空比为 50%的奇数 n 分频时钟。

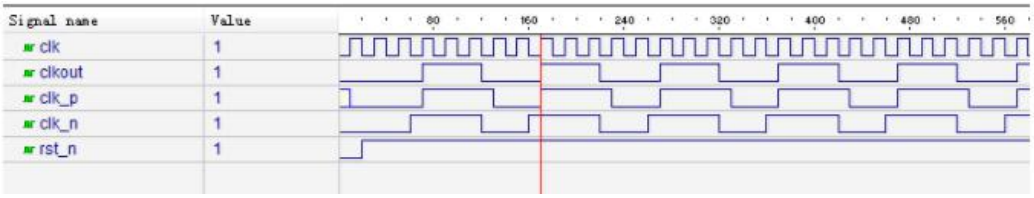

Signal name
Value
80
160
240
320
400
480
560
clk
1
clkout
1
clk_p
1
clk_n
1
rst_n
1

图 4-1　奇数分频时序图

2. Verilog 代码

```
module divide (   clk,rst_n,clkout);

       input   clk,rst_n;                        //输入信号，其中clk连接到
FPGA的C1脚，频率为12MHz
       output  clkout;                          //输出信号，可以连接到LED观
察分频的时钟
       //parameter是verilog里常数语句
    parameter  WIDTH   = 3;                 //计数器的位数，计数的最大值为
2**WIDTH-1
    parameter  N    = 5;              //分频系数，请确保 N < 2**WIDTH-1，否
则计数会溢出

    reg     [WIDTH-1:0]    cnt_p,cnt_n;     //cnt_p为上升沿触发时的计数
器，cnt_n为下降沿触发时的计数器
    reg         clk_p,clk_n;     //clk_p为上升沿触发时分频时钟，clk_n为下
降沿触发时分频时钟
    //上升沿触发时计数器的控制
    always @ (posedge clk or negedge rst_n )          //posedge和negedge
是verilog表示信号上升沿和下降沿
                                                     //当clk上升沿来临或者
rst_n变低的时候执行一次always里的语句
       begin
           if(!rst_n)
               cnt_p<=0;
           else if (cnt_p==(N-1))
               cnt_p<=0;
           else cnt_p<=cnt_p+1;              //计数器一直计数，当计数到N-1
的时候清零，这是一个模N的计数器
       end
        //上升沿触发的分频时钟输出,如果N为奇数得到的时钟占空比不是50%；如果N
为偶数得到的时钟占空比为50%
```

```
        always @ (posedge clk or negedge rst_n)
       begin
           if(!rst_n)
               clk_p<=0;
           else if (cnt_p<(N>>1))          //N>>1表示右移一位，相当于除以
2去掉余数
               clk_p<=0;
           else
               clk_p<=1;              //得到的分频时钟正周期比负周期多一个
clk时钟
       end

       //下降沿触发时计数器的控制
   always @ (negedge clk or negedge rst_n)
       begin
           if(!rst_n)
               cnt_n<=0;
           else if (cnt_n==(N-1))
               cnt_n<=0;
           else cnt_n<=cnt_n+1;
       end

       //下降沿触发的分频时钟输出，和clk_p相差半个时钟
   always @ (negedge clk)
       begin
           if(!rst_n)
               clk_n<=0;
           else if (cnt_n<(N>>1))
               clk_n<=0;
           else
               clk_n<=1;               //得到的分频时钟正周期比负周期多一
个clk时钟
       end

       assign clkout = (N==1)?clk:(N[0])?(clk_p&clk_n):clk_p;      //
条件判断表达式
                                                                   //当N=1时，
直接输出clk
                                                                   //当N为偶数
也就是N的最低位为0，N（0）=0，输出clk_p
                                                                   //当N为奇数
也就是N最低位为1，N（0）=1，输出clk_p&clk_n。正周期多所以是相与
endmodule
```

测试文件，进行功能仿真时需要编写 testbench 测试文件。verilog 里的 testbench 文件和源文件一样也是.v 文件，仿真能让我们更直观的观察信号波形，可以先阅读软件手册的使用了解如何使用仿真工具。

```
`timescale 1ns/100ps                  //仿真时间单位/时间精度，时间单位要大于或者等于时间精度

module divide_tb();                  //测试文件也是一个module，因为用于仿真所以无需输入输出信号

    reg    clk,rst_n;                 //需要产生的激励信号定义，激励信号需要过程块产生所以定义为reg型变量
    wire   clkout;                    //需要观察的输出信号定义，定义为wire型变量

    //初始化过程块
    initial
      begin
         clk = 0;
         rst_n = 0;
         #25                                  //#表示延时25个时间单位
         rst_n = 1;                           //产生了一个初始25ns低电平，然后变高电平的复位信号
       end

    always #10 clk = ~clk;                    //每隔10ns翻转一次clk信号，也就是产生一个时钟周期20ns的clk，频率为50MHz

          //module调用例化格式
     divide  #(.WIDTH(4),.N(11))  u1 (      //#后面的（）中为参数传递，如果不传递参数就是所调用模块中的参数默认值
                                          //divide表示所要例化的module名称，u1是我们定义的例化名称，必须以字母开头
                        .clk    (clk),     //输入输出信号连接。 .clk表示module本身定义的信号名称；（clk）表示我们在这里定义的激励信号
                        .rst_n  (rst_n),   //在testbench里定义的信号名称可以与所要调用module的端口信号名称不同
                        .clkout    (clkout)
                     );
         endmodule
```

3. 引脚分配

小脚丫上的系统时钟连接到 FPGA 的 C1 脚，时钟为 12MHz。你可以通过仿真波形观察分频时钟（注意仿真的时间是有限的，所以分频时钟频率需要较高）。如果我们想通过眼睛观察 LED 的闪烁，那么需要设置参数 N 和 WIDTH 得到一个频率较低的时钟（例如 N=12000000，WIDTH=24，分频时钟周期为 1 秒）。

信号	引脚
clk	J5
rst_n	J9
clkout	N15

修改程序中的分频系数和计数器位数就能够调整 LED 闪烁速度（注意计数的最大值一定要保证超过分频系数 N）。

4. 小结

在本实验学习了如何进行任意整数的分频设计，我们产生各种时钟，通过修改程序还能实验调整输出时钟的频率、相位以及占空比，非常灵活。同时学习了如何编写 testbench 文件，了解 verilog 中如何例化 module，在后面的学习中将会经常用到。在下个实验我们将进一步了解时序逻辑，如何利用时钟来进一步设计，请看最常见的 LED 流水灯。

实验六　LED 流水灯

在时钟分频实验中我们练习了如何处理时钟，接下来我们要学习如何利用时钟来完成时序逻辑。

1. 硬件说明

流水灯实现是很常见的一个实验，虽然逻辑比较简单，但是里面也包含了实现时序逻辑的基本思想。要用 FPGA 实现流水灯有很多种方法，在这里我们会用两种不同的方法实现。

(1)模块化设计

在之前的实验中我们做了 3-8 译码器和时钟分频，如果把这两个结合起来，我们就能搭建一个自动操作的流水 LED 显示。框图如下：

(2)循环赋值

这是一种很简洁的实现流水灯效果逻辑，就是定义一个 8 位的变量，在每个时钟上升沿将最低位赋值给最高位，其他位右移一位，这就实现了循环赋值。这 8 位输出到 LED 就能实现流水灯。

2. Verilog 代码

模块化设计是用硬件描述语言进行数字电路设计的精髓，代码可重复利用。而且模块化的设计使得程序的结构也很清晰。这里我们首先看看流水灯的模块化设计。利用了之前的**3-8 译码器**和**分频器**，你需要把这两个程序也拷贝到一个工程。

```
module flashled (clk,rst,led);

    input clk,rst;
    output [7:0] led;


        reg   [2:0] cnt ;                                  //定义了一个3位的计数
器，输出可以作为3-8译码器的输入

        wire clk1h;                                        //定义一个中间变量，表
示分频得到的时钟，用作计数器的触发

        //例化module decode38，相当于调用
        decode38 u1 (
            .sw(cnt),                          //例化的输入端口连接到cnt，输出端
口连接到led
            .led(led)
            );

        //例化分频器模块，产生一个1Hz时钟信号
        divide #(.WIDTH(32),.N(12000000)) u2 (         //传递参数
            .clk(clk),
            .rst_n(rst),                       //例化的端口信号都连接到定义好的
信号
            .clkout(clk1h)
            );

        //1Hz时钟上升沿触发计数器，循环计数
        always @(posedge clk1h or negedge rst)
         if (!rst)
        cnt <= 0;
         else
        cnt <= cnt +1;

        endmodule
```

模块化设计结构清晰，verilog 语言是很灵活的。对于流水灯还有一种很简洁的实现方法:

```
module flash (clk,rst,led);

    input clk,rst;
    output [7:0] led;
        wire clk1h;                                    //定义一个中间变量，表示分频得到的时钟，用作计数器的触发
        //例化分频器模块，产生一个1Hz时钟信号
        divide #(.WIDTH(32),.N(12000000)) u2 (         //传递参数
            .clk(clk),
            .rst_n(rst),                    //例化的端口信号都连接到定义好的信号
            .clkout(clk1h)
            );

        //1Hz时钟上升沿触发循环赋值
        always@(posedge clk1h or negedge rst)
    begin
        if(!rst)
            led <= 8'b11111110;            // <=为非阻塞赋值
        else
            led <= {led[0],led[7:1]};      //当时钟上升沿来一次，执行一次赋值，赋值内容是led[0]与led[7:1]重新拼接成8位赋给led，相当于循环右移
    end
endmodule
```

3.引脚分配

按照下面表格定义输入输出信号

信号	引脚	信号	引脚
clk	J5	led[3]	M12
rst	J9	led[4]	L15
led[0]	N15	led[5]	K12
led[1]	N14	led[6]	L11
led[2]	M14	led[7]	K11

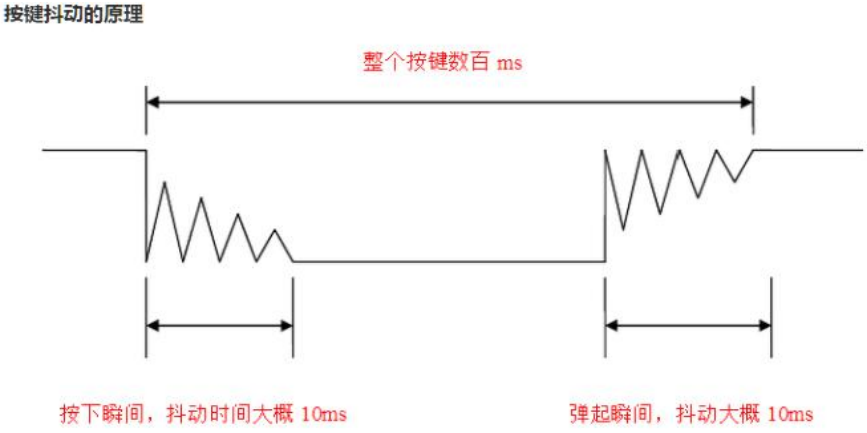
配置好以后编译下载程序。可以调整例化分频器时传递的参数来调整流水灯的速度。

实验七　按键消抖

在之前的实验中我们学习了如何用按键作为 FPGA 的输入控制，在本实验中将学习如何进行按键消抖，用按键完成更多的功能。

1. 硬件说明

按键是一种常用的电子开关，电子设计中不可缺少的输入设备。当按下时使开关导通，松开时则开关断开，内部结构是靠金属弹片来实现通断。



按键一次出现的电平变化

2. 按键去抖原理：

抖动的产生

通常的按键所用的开关为机械弹性开关，当机械触点断开、闭合时，由于机械触点的弹性作用，一个按键开关在闭合时不会马上稳定地接通，在断开时也不会一下子断开。因而在闭合及断开的瞬间均伴随有一连串的抖动，为了不产生这种现象而作的措施就是按键消抖。

消除抖动的措施

一般我们采用软件方法消抖。即检测到按键按下动作之后进行 10ms~20ms 左右的延时，当前沿的抖动消失之后再一次检测按键的状态。如果仍然是按下的电平状态，则认为这是一次真正的按键按下；同样检测到按键释放，也要做 10ms~20ms 延时，检测到后沿抖动消失后认为是一个完整的按键弹起过程。

消抖的好处

执行按键消抖有两个好处，

消除误触发：我们想通过按键来翻转信号（例如按下一次 led 亮，在按一次 led 灭），如果没有进行消抖，则会产生很多误触发造成不必要的翻转。

记录按键次数：执行按键消抖可以让我们记录按键动作的次数，在很多应用里这非

常有用。

在点亮 LED 实验中我们知道了小脚丫板子上按键的设计，当按键未被按下时，连接到 FPGA 管脚认为是高电平；当按键被按下时，连接到 FPGA 管脚认为是低电平。

要消除按键的抖动，我们需要去扫描按键，也就是不断的去采集按键的状态。软件消抖时我们一般只考虑按键按下时的抖动，而放弃对释放时抖动的消除。用系统时钟（频率较高）去采集按键状态，当检测到按下时用计数器延时 20ms，再去检测按键状态，如果这时仍为按下状态，确认是一次按下动作，否侧的话认为无按键按下。如何检测按键状态变化就需要用到脉冲边沿检测的方法。

3. 脉冲边沿检测原理

检测按键按下时要用到脉冲边沿检测的方法，捕捉信号的突变、捕捉时钟的上升下降沿等经常会用到这种方法。简单地说就是用一个频率更高的时钟去触发要检测的信号，用两个寄存器去储存相邻两个时钟采集到的值，然后进行异或运算，如果不为零，代表发生了上升沿或者下降沿。

在按键消抖的过程中，同样运用了脉冲边沿检测。用两个寄存器储存相邻时钟采集的值（例如 data_pre,data）,然后将 data 取反与前一个值相与（state=data_pre&（~data））,如果为 1，则判断有下降沿既按键按下由高到低；否则无变化。

将一个信号由连续时钟采集，相邻两个钟触发的值存入两个寄存器。理解 verilog 实现这个过程要充分了解其中的非阻塞赋值。

4. Verilog 代码

本实验主要通过按键来控制 led 的翻转，当按下一次 led 变亮，再按下一次 led 变暗。首先我们做个试验，对按键不做处理通过按键来控制 led 翻转。

```
module top(
        key,          //按键输入
        rst,          //复位输入
        led           //led输出
        );

    input key,rst;
    output reg led;

    always @(key or rst)
        if (!rst)             //复位时led熄灭
            led = 1;
        else if(key == 0)
            led = ~led;   //按键按下时led翻转
        else
            led = led;
endmodule
```

未经过消抖的程序下载到小脚丫上会发现按键有时不能够控制 led 翻转，这是因为按键的抖动造成了 led 状态变化不可控，所以我们必须将抖动消除。下面是一种延时去抖的程序。

```
module debounce (clk,rst,key,key_pulse);

        parameter       N  =  1;                        //要消除的按键的数量
        input            clk;
        input            rst;
        input   [N-1:0]   key;                          //输入的按键
        output  [N-1:0]   key_pulse;                     //按键动作产生的脉冲
        reg     [N-1:0]   key_rst_pre;                   //定义一个寄存器型变
量存储上一个触发时的按键值
        reg     [N-1:0]   key_rst;                      //定义一个寄存器变量
储存储当前时刻触发的按键值
        wire    [N-1:0]   key_edge;                     //检测到按键由高到低
变化是产生一个高脉冲

        //利用非阻塞赋值特点，将两个时钟触发时按键状态存储在两个寄存器变量中
        always @(posedge clk  or  negedge rst)
          begin
             if (!rst) begin
                 key_rst <= {N{1'b1}};                  //初始化时给key_rst
赋值全为1，{}中表示N个1
                 key_rst_pre <= {N{1'b1}};
             end
             else begin
                 key_rst <= key;                       //第一个时钟上升沿触发之
后key的值赋给key_rst,同时key_rst的值赋给key_rst_pre
                 key_rst_pre <= key_rst;               //非阻塞赋值。相当于经
过两个时钟触发，key_rst存储的是当前时刻key的值，key_rst_pre存储的是前一个时
钟的key的值
             end
           end

        assign  key_edge = key_rst_pre & (~key_rst);//脉冲边沿检测。当
key检测到下降沿时，key_edge产生一个时钟周期的高电平

        reg [17:0]   cnt;                           //产生延时所用的计数器，系统
时钟12MHz，要延时20ms左右时间，至少需要18位计数器
```

```
        //产生20ms延时，当检测到key_edge有效是计数器清零开始计数
       always @(posedge clk or negedge rst)
         begin
           if(!rst)
              cnt <= 18'h0;
           else if(key_edge)
              cnt <= 18'h0;
           else
              cnt <= cnt + 1'h1;
           end

       reg     [N-1:0]   key_sec_pre;              //延时后检测电平寄存
器变量
       reg     [N-1:0]   key_sec;


       //延时后检测key，如果按键状态变低产生一个时钟的高脉冲。如果按键状态是高
的话说明按键无效
       always @(posedge clk  or  negedge rst)
         begin
           if (!rst)
               key_sec <= {N{1'b1}};
           else if (cnt==18'h3ffff)
               key_sec <= key;
         end
      always @(posedge clk  or  negedge rst)
         begin
           if (!rst)
               key_sec_pre <= {N{1'b1}};
           else
               key_sec_pre <= key_sec;
        end
      assign  key_pulse = key_sec_pre & (~key_sec);

endmodule
```

以上就是一个 N 位按键的消抖程序，如果有按键按下会输出一个时钟周期的高脉冲。下面我们可以试试用这个按键消抖的输出来触发 LED 的显示，既按键一次 LED 翻转。你也可以不加按键消抖试试用按键来控制 LED（按一次变亮，再按一次灭掉）。

下面的程序是例化调用 debounce 模块来控制 LED：

```
module top (clk,rst,key,led);

       input            clk;
       input            rst;
       input            key;
    output   reg      led;

       wire             key_pulse;

       //当按键按下时产生一个高脉冲，翻转一次led
       always @(posedge clk  or  negedge rst)
         begin
           if (!rst)
        led <= 1'b1;
         else if (key_pulse)
        led <= ~led;
         else
              led <= led;
       end
        //例化消抖module，这里没有传递参数N，采用了默认的N=1
        debounce  u1 (
                    .clk (clk),
                    .rst (rst),
                    .key (key),
                    .key_pulse (key_pulse)
                    );
 endmodule
```

3.引脚分配

设置好复位键可消抖的按键，编译完成后下载，通过按键就可以翻转 LED。你也可以定义多个按键控制多个 LED，还可以比较不加按键消抖情况下实际的效果对比如何。

信号	引脚
clk	J5
rst	J9
key	K14
led	N15

4. 小结

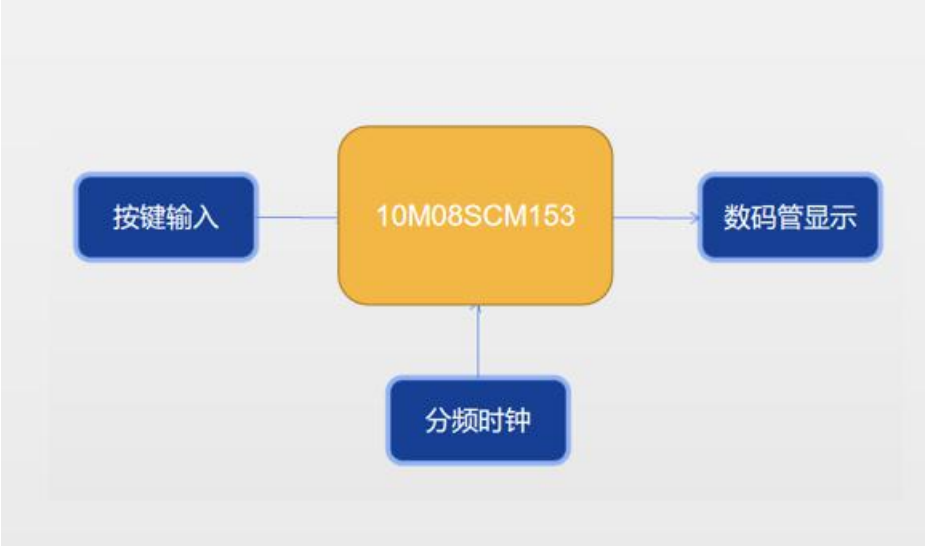
在本实验学习了如何进行按键的消抖。在很多应用情况下我们必须采取消抖才能更好地控制逻辑。在下一个实验计时控制中我们将学习计时的显示和控制，在这里我们要用到按键的消抖以及数码管，我们甚至可以用小脚丫做一个计时器甚至电子表。

实验八　计时控制

在之前的实验中我们掌握了如何进行时钟分频、如何进行数码管显示与按键消抖的处理，那么在本节实验之中，我们将会实现一个篮球赛场上常见的 24 秒计时器。

1. 硬件说明

在之前的实验中我们为读者详细介绍过小脚丫 MAX10 板卡上的按键、数码管、LED 等硬件外设，在此不再赘述。本节将实现由数码管作为显示模块，按键作为控制信号的输入(包含复位信号和暂停信号)，Altera MAX10 作为控制核心的篮球读秒系统，实现框图如下：



2. Verilog 代码

```
module counter
(
    clk             ,    //时钟
    rst             ,    //复位
    hold            ,    //启动暂停按键
    seg_led_1       ,    //数码管 1
    seg_led_2       ,    //数码管 2
    led                  //led
);

    input   clk,rst;
    input   hold;
```

```
    output [8:0]  seg_led_1,seg_led_2;
    output reg [7:0]  led;

    wire       clk1h;       //1Hz 时钟
    wire       hold_pulse;   //按键消抖后信号
    reg        hold_flag;    //按键标志位
    reg        back_to_zero_flag ; //计时完成信号
    reg        [6:0]   seg       [9:0];
    reg        [3:0]  cnt_ge;      //个位
    reg        [3:0]  cnt_shi;     //十位

    initial
    begin
        seg[0] = 7'h3f;      //  0
        seg[1] = 7'h06;      //  1
        seg[2] = 7'h5b;      //  2
        seg[3] = 7'h4f;      //  3
        seg[4] = 7'h66;      //  4
        seg[5] = 7'h6d;      //  5
        seg[6] = 7'h7d;      //  6
        seg[7] = 7'h07;      //  7
        seg[8] = 7'h7f;      //  8
        seg[9] = 7'h6f;      //  9
/*若需要显示 A-F,解除此段注释即可
        seg[10]= 7'hf7;      //  A
        seg[11]= 7'h7c;      //  b
        seg[12]= 7'h39;    //  C
        seg[13]= 7'h5e;    //  d
        seg[14]= 7'h79;    //  E
        seg[15]= 7'h71;    //  F*/
    end



    // 启动/暂停按键进行消抖
    debounce  U2 (
```

```
            .clk(clk),
            .rst(rst),
            .key(hold),
            .key_pulse(hold_pulse)
            );
// 用于分出一个 1Hz 的频率
divide #(.WIDTH(32),.N(12000000)) U1 (
        .clk(clk),
        .rst_n(rst),
        .clkout(clk1h)
        );
//按键动作标志信号产生
always @ (posedge hold_pulse)
    if(!rst==1)
        hold_flag <= 0;
    else
        hold_flag <= ~hold_flag;
//计时完成标志信号产生
always @ (*)
    if(!rst == 1)
        back_to_zero_flag <= 0;
    else if(cnt_shi==0 && cnt_ge==0)
        back_to_zero_flag <= 1;
    else
        back_to_zero_flag <= 0;
//24 秒倒计时控制
always @ (posedge clk1h or negedge rst) begin
    if (!rst == 1) begin
        cnt_ge <= 4'd4;
        cnt_shi <= 4'd2;
        end
    else if(hold_flag == 1)begin
        cnt_ge <= cnt_ge;
        cnt_shi <= cnt_shi;
        end
    else if(cnt_shi==0 && cnt_ge==0) begin
        cnt_shi <= cnt_shi;
```

```
            cnt_ge <= cnt_ge;
            end
        else if(cnt_ge==0)begin
            cnt_ge <= 4'd9;
            cnt_shi <= cnt_shi-1;end
        else
            cnt_ge <= cnt_ge -1;
        end
    //计时完成点亮 led
    always @ ( back_to_zero_flag)begin
        if (back_to_zero_flag==1)
            led = 8'b0;
        else
            led = 8'b11111111;
        end

    assign seg_led_1[8:0] = {2'b00,seg[cnt_ge]};

    assign seg_led_2[8:0] = {2'b00,seg[cnt_shi]};


endmodule
```

3. 引脚分配

信号	引脚	信号	引脚
clk	J5	**seg_led_1[3]**	P1
rst	J9	**seg_led_1[4]**	N1
hold	K14	**seg_led_1[5]**	C1
seg_led_2[0]	E1	**seg_led_1[6]**	C2
seg_led_2[1]	D2	**seg_led_1[7]**	R2
seg_led_2[2]	K2	**seg_led_1[8]**	B1
seg_led_2[3]	J2	**led[0]**	**N15**
seg_led_2[4]	G2	**led[1]**	**N14**

seg_led_2[5]	F5	**led[2]**	**M14**
seg_led_2[6]	G5	**led[3]**	**M12**
seg_led_2[7]	L1	**led[4]**	**L15**
seg_led_2[8]	E2	**led[5]**	**K12**
seg_led_1[0]	A3	**led[6]**	**L11**
seg_led_1[1]	A2	**led[7]**	**K11**
seg_led_1[2]	P2		

4. 小结

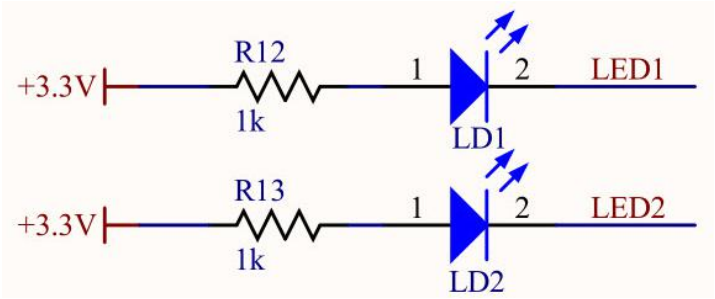
本实验主要介绍了计时器的实现方式，并且包含了复位与暂停功能，读者可自行修改程序内部的时钟参数来调节计时时间。下一节将介绍 PWM 调制技术的应用呼吸灯。

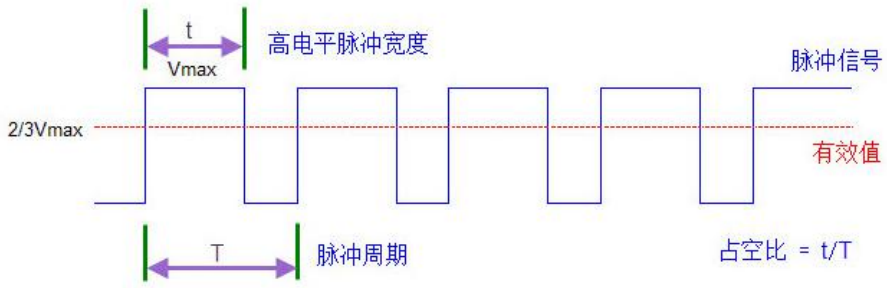
实验九　脉宽调制

本节将向您介绍 Verilog 语法之中的精髓内容——状态机，并且将利用状态机实现十字路口的交通灯。

1. 硬件说明

呼吸灯的设计较为简单，我们使用 12MHz 的系统时钟作为高频信号做分频处理，调整占空比实现 PWM，通过 LED 灯 LD1 指示输出状态。



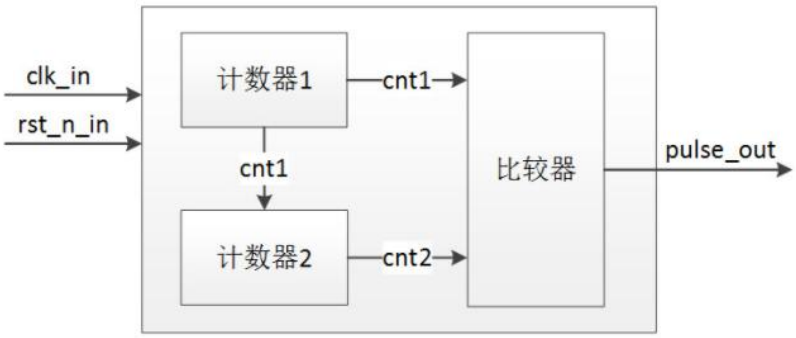


实现原理如上图所示，脉冲信号的周期为 T，高电平脉冲宽度为 t，占空比为 t/T。为了实现 PWM 脉宽调制，我们需要保持周期 T 不变，调整高电平脉宽 t 的时间，从而改变占空比。

当 t = 0 时，占空比为 0%，因为我们的 LED 硬件为低电平点亮，所以为最亮的状态。

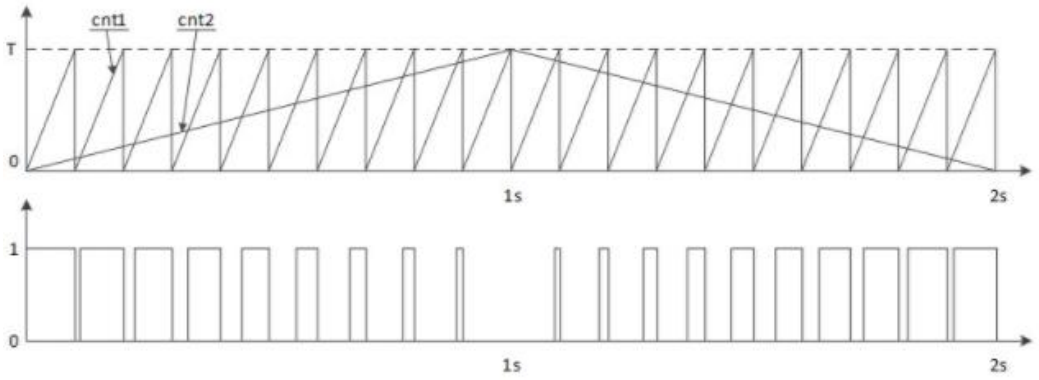
当 t = T 时，占空比为 100%，LED 灯为最暗（熄灭）的状态。

结合呼吸灯的原理，整个呼吸的周期为最亮→最暗→最亮的时间，即 t 的值的变化：0→T→0 逐渐变化，这个时间应该为 2s



呼吸灯设计要求呼吸的周期为 2s，也就是说 LED 灯从最亮的状态开始，第一秒时间内逐渐变暗，第二秒的时间内再逐渐变亮，依次进行。

本设计中需要两个计数器 cnt1 和 cnt2，cnt1 随系统时钟同步计数（系统时钟上升沿时 cnt1 自加 1）范围为 0~T，cnt2 随 cnt1 的周期同步计数（cnt1 等于 T 时，cnt2 自加 1）范围也是 0~T，这样每次 cnt1 在 0~T 的计数时，cnt2 为一个固定值，相邻 cnt1 计数周期对应的 cnt2 的值逐渐增大，我们将 cnt1 计数 0~T 的时间作为脉冲周期，cnt2 的值作为脉冲宽度，则占空比 = cnt2/T,占空比从 0%到 100%的时间 = cnt2*cnt1 = T^2 = 1s = 12M 个系统时钟，T = 2400，我们定义 CNT_NUM = 2400 作为两个计数器的计数最大值。



2. Verilog 代码

```
module breath_led(clk,rst,led);

    input clk;              //系统时钟输入
    input rst;              //复位输出
    output led;             //led输出

    reg [24:0] cnt1;        //计数器1
    reg [24:0] cnt2;        //计数器2
    reg flag;               //呼吸灯变亮和变暗的标志位

    parameter   CNT_NUM = 2400;  //计数器的最大值 period = (2400^2)*2 =
24000000 = 2s
    //产生计数器cnt1
    always@(posedge clk or negedge rst) begin
        if(!rst) begin
            cnt1<=13'd0;
            end
         else begin
             if(cnt1>=CNT_NUM-1)
                      cnt1<=1'b0;
             else
                      cnt1<=cnt1+1'b1;
            end
        end

    //产生计数器cnt2
    always@(posedge clk or negedge rst) begin
        if(!rst) begin
            cnt2<=13'd0;
            flag<=1'b0;
            end
         else if(cnt1==CNT_NUM-1) begin             //当计数器1计满时计数
器2开始计数加一或减一
            if(!flag) begin                    //当标志位为0时计数器2递增计
数，表示呼吸灯效果由暗变亮
                if(cnt2>=CNT_NUM-1)         //计数器2计满时，表示亮度已最
大，标志位变高，之后计数器2开始递减
                      flag<=1'b1;
                else
                      cnt2<=cnt2+1'b1;
                end
            else begin                     //当标志位为高时计数器2递减计数
```

```
                if(cnt2<=0)                    //计数器2级到0，表示亮度已最小，
标志位变低，之后计数器2开始递增
                        flag<=1'b0;
                else
                        cnt2<=cnt2-1'b1;
                end
            end
          else
            cnt2<=cnt2;                //计数器1在计数过程中计数器2保持不变
    end

//比较计数器1和计数器2的值产生自动调整占空比输出的信号，输出到led产生呼吸灯效果
    assign led = (cnt1<cnt2)?1'b0:1'b1;

endmodule
```

3. 引脚分配

引脚分配如下：

管脚名称	clk	rst	led
FPGA 管脚	J5	J9	N15

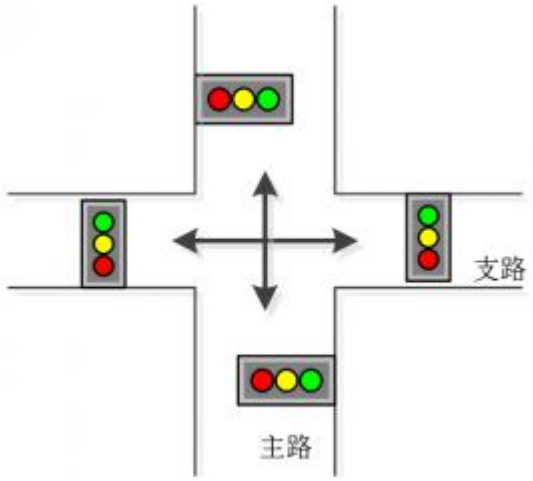
4. 小结

脉宽调制是一种值得广大工程师在许多应用设计中使用的有效技术，你也可以根据本节介绍的流水灯程序，实现 RGB 三色灯的呼吸。在下一小节我们会学习状态机的使用方法：交通灯的设计。

实验十　简易交通灯

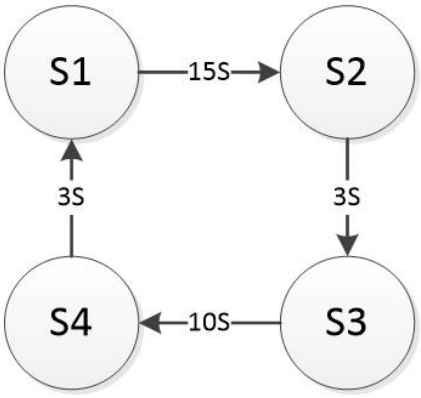
本节将向您介绍 Verilog 语法之中的精髓内容——状态机，并且将利用状态机实现十字路口的交通灯。

1. 硬件说明与实现项目框图



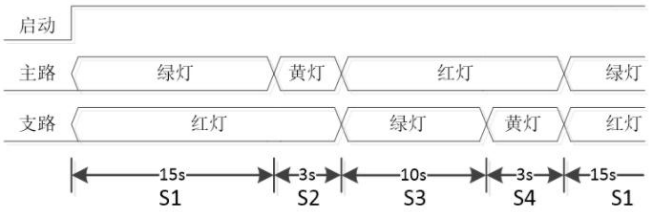
上图为十字路口交通示意图分之路与主路，要求如下：

- 交通灯主路上绿灯持续 15s 的时间，黄灯 3s 的时间，红灯 10s 的时间；
- 交通灯支路上绿灯持续 7s 的时间，　黄灯持续 3 秒的时间，红灯 18 秒的时间；



根据上述要求，状态机设计框架分析如下：

- S1:主路绿灯点亮，支路红灯点亮，持续 15s 的时间；
- S2:主路黄灯点亮，支路红灯点亮，持续 3s 的时间；
- S3:主路红灯点亮，支路绿灯点亮，持续 10s 的时间；
- S4:主路红灯点亮，支路黄灯点亮，持续 3s 的时间；



2. Verilog 代码

工程中会用到之前实验的分频模块，以 STEP-MAX10 上的 2 个红绿蓝三色 LED 代表十字路口的红绿黄交通灯。

接下来就是利用三段式状态机实现的交通灯部分：

```
module traffic
(
    clk         ,    //时钟
    rst_n       ,    //复位
    out              //三色 led 代表交通灯
);

    input  clk,rst_n;
    output reg[5:0]   out;

    parameter          S1 = 4'b00,    //状态机状态编码
                  S2 = 4'b01,
                  S3 = 4'b10,
                  S4 = 4'b11;

    parameter      time_s1 = 4'd15, //计时参数
                  time_s2 = 4'd3,
                  time_s3 = 4'd10,
                  time_s4 = 4'd3;
    //交通灯的控制
    parameter      led_s1 = 6'b101011, // LED2 绿色 LED1 红色
                  led_s2 = 6'b110011, // LED2 蓝色 LED1 红色
                  led_s3 = 6'b011101, // LED2 红色 LED1 绿色
                  led_s4 = 6'b011110; // LED2 红色 LED1 蓝色
```

```
reg     [3:0]   timecont;
reg     [1:0]   cur_state,next_state;  //现态、次态

wire            clk1h;  //1Hz 时钟

//产生 1 秒的时钟周期
divide #(.WIDTH(32),.N(12000000)) CLK1H (
                            .clk(clk),
                            .rst_n(rst_n),
                            .clkout(clk1h));

//第一段 同步逻辑 描述次态到现态的转移
always @ (posedge clk1h or negedge rst_n)
begin
    if(!rst_n)
        cur_state <= S1;
    else
        cur_state <= next_state;
end
//第二段 组合逻辑描述状态转移的判断
always @ (cur_state or rst_n or timecont)
begin
    if(!rst_n) begin
        next_state = S1;
        end
    else begin
        case(cur_state)
            S1:begin
                if(timecont==1)
                    next_state = S2;
                else
                    next_state = S1;
            end

           S2:begin
                if(timecont==1)
                    next_state = S3;
```

```
                else
                    next_state = S2;
            end

            S3:begin
                if(timecont==1)
                    next_state = S4;
                else
                    next_state = S3;
            end

            S4:begin
                if(timecont==1)
                    next_state = S1;
                else
                    next_state = S4;
            end

            default: next_state = S1;
        endcase
    end
end
//第三段  同步逻辑 描述次态的输出动作
always @ (posedge clk1h or negedge rst_n)
begin
    if(!rst_n==1) begin
        out <= led_s1;
        timecont <= time_s1;
        end
    else begin
        case(next_state)
            S1:begin
                out <= led_s1;
                if(timecont == 1)
                    timecont <= time_s1;
                else
                    timecont <= timecont - 1;
```

```
                end

                S2:begin
                    out <= led_s2;
                    if(timecont == 1)
                        timecont <= time_s2;
                    else
                        timecont <= timecont - 1;
                end

                S3:begin
                    out <= led_s3;
                    if(timecont == 1)
                        timecont <= time_s3;
                    else
                        timecont <= timecont - 1;
                end

                S4:begin
                    out <= led_s4;
                    if(timecont == 1)
                        timecont <= time_s4;
                    else
                        timecont <= timecont - 1;
                end

                default:begin
                    out <= led_s1;
                    end
            endcase
        end
    end
endmodule
```

3. 引脚分配

信号	引脚	信号	引脚
clk	J5	**rst**	J9
out[0]	E14	**out[1]**	E15
out[2]	G15	**out[3]**	D12
out[4]	C14	**out[5]**	C15

配置好以后编译下载程序。您也可以试试修改程序，观察修改代码对于 FPGA 内部电路所造成的影响。

4. 小结

状态机是一类很重要的时序逻辑电路，是许多数字系统的核心部件，掌握状态机的使用是利用 FPGA 与 CPLD 进行开发的一项必会技能，本小节的交通灯程序即是利用三段式状态机描述方法实现的，希望读者能够快速掌握这项技能。