

2. 键控流水灯实验例程

2.1 MES22GP 开发板简介

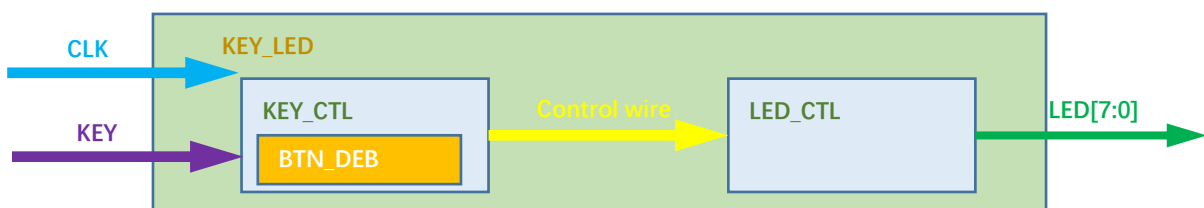
MES22GP 扩展底板提供了 8 个用户按键（USER_KEY1~8）和 1 个重加载复位按键，重加载按键通过一个延时复位芯片连接到 PGL22G 的 RSTN 管脚；8 个用户按键都连接到 PGL22G 的普通 IO 上，按键低电平有效，但按键按下时，IO 上的输入电压为低；当没有按下按键时，IO 上的输入电压为高电平（详情请查看“MES22GP 开发板硬件使用手册”）。

2.2 实验目的

由 USER_KEY1 按键输入，切换 USER_LED1~ USER_LED8 的输出效果。

2.3 实验原理

实现框架如下：



- (1) 顶层实现按键切换 LED 的流水灯状态；
- (2) 需要设计一个输入控制模块及一个输出控制模块；

这个实验带大家将多个模块整合成为一个工程，涉及到的知识点有子模块设计、模块例化；子模块的设计主要是依据功能定位，确定输入输出，再做具体的设计；

模块例化方式如下：

```

1  module_name # (
2      .PARAM      ( PARAM_SET )      // PARAM为例化模块的常量接口；PARAM_SET为常量赋值内容
3  ) uint_name(      // module_name 为例化module名；uint_name为例化后单元名称
4      .port        ( signal          )      // port为例化模块中的管脚；signal为当前模块的信号
5  );

```

2.3.1 按键控制模块功能

接收按键输入信号。统计按键按下次数，由于流水灯模式是 3 种，计数统计范围是 0~2

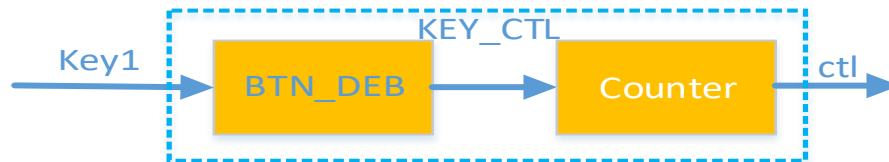
循环，将计数结果传递给 LED 控制模块；

根据需求输入信号有：时钟，按键；输出信号有：流水灯控制信号；

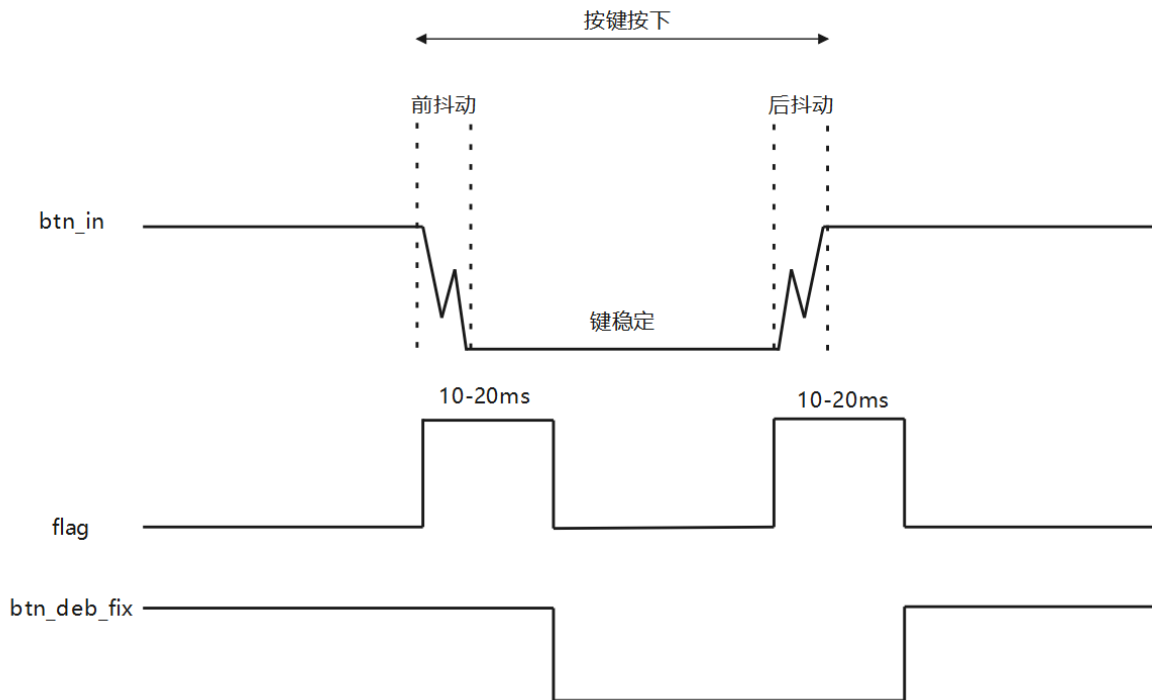
内部功能处理：

<1>内部需要对按键信号做消抖处理；

<2>按键触发计数器（计数值输出）改变继而调整流水灯的状态；



2.3.2 按键消抖模块



前后抖动时间约为 5~10ms，取按键抖动区间开始标识，持续 10-20ms 后标识归零，在抖动区间内输出保持，非消抖区间，按键状态输出。

2.3.3 LED 控制模块功能

3 种流水灯模式有按键传递过来的计数控制切换，每一个 LED 的显示状态完整后进入下一模式初始化。根据需求可得到如下信息：

输入信号：时钟，流水灯模式控制信号； 出信号：8bit 位宽的 LED 控制信号；

功能处理注意事项: 流水灯状态切换点, 不同状态的切换时如何初始化;

2.4 实验源码设计

2.4.1 顶层文件源码

```
1  `timescale 1ns / 1ps
2
3  `define UD #1
4  module key_led_top(
5      input      clk,
6      input      key,
7      output [7:0] led
8  );
9
10     wire [1:0] ctrl;
11
12     key_ctl key_ctl(
13         .clk      ( clk ),//input      clk,
14         .key       ( key ),//input      key,
15         .ctrl      ( ctrl )//output [1:0]ctrl
16     );
17
18     led u_led(
19         .clk      ( clk ),//input      clk,
20         .ctrl     ( ctrl ),//input [1:0] ctrl,
21         .led      ( led ) //output[7:0] led
22     );
23
24
25 endmodule
```

2.4.2 按键控制模块

```

1  `timescale 1ns / 1ps
2  `define UD #1
3  module key_ctl(
4      input          clk,
5      input          key,
6
7      output [1:0] ctrl
8  );
9
10     wire btn_deb;
11     // 按键消抖
12     btn_deb#(
13         .BTN_WIDTH ( 4'd1 ) //parameter          BTN_WIDTH = 4'd8
14     ) U_btn_deb
15     (
16         .clk        ( clk      ),//input          clk,
17         .btn_in     ( key      ),//input          [BTN_WIDTH-1:0] btn_in,
18
19         .btn_deb    ( btn_deb ) //output reg [BTN_WIDTH-1:0] btn_deb
20     );
21
22     reg btn_deb_1d;
23     always @(posedge clk)
24     begin
25         btn_deb_1d <= `UD btn_deb; //get the btn_deb delay one clock cycle
26     End
27
28     //下降沿获取方式: 前一个时钟周期为高电平, 当前时钟周期为低电平;
29     // 故而将按键消抖后的信号打一拍 (保持上一时钟周期的状态)
30     //
31     // sig          _____|_____
32     //
33     // sig_reg      _____|_____
34     //
35     // falling      _____|_|_____
36
37     reg [1:0] key_push_cnt=2'd0;
38     always @(posedge clk)
39     begin
40         if(~btn_deb & btn_deb_1d) //get he falling edge of btn_deb
41         begin
42             key_push_cnt <= `UD key_push_cnt + 2'd1;
43         end
44     end
45
46     assign ctrl = key_push_cnt;
47
48 endmodule
49

```

2.4.3 按键消抖模块

```

1  `timescale 1ns / 1ps
2  `define UD #1
3  module btn_deb_fix#(
4      parameter          BTN_WIDTH = 4'd8,
5      parameter          BTN_DELAY = 20'h7_ffff
6  )
7  (
8      input               clk,    //
9      input               [BTN_WIDTH-1:0] btn_in,
10
11     output reg [BTN_WIDTH-1:0] btn_deb_fix
12 );
13     //16'h3ad43;
14     reg [17:0]           cnt[BTN_WIDTH-1:0];
15     reg [BTN_WIDTH-1:0] flag;
16
17     reg [BTN_WIDTH-1:0] btn_in_reg;
18
19     always @(posedge clk)
20     begin
21         btn_in_reg <= `UD btn_in;
22     end
23
24     genvar i;
25     generate
26     begin
27         for(i=0;i<BTN_WIDTH;i=i+1)
28         begin
29             always @(posedge clk)
30             begin
31                 if (btn_in_reg[i] ^ btn_in[i]) //取按键边沿开始抖动区间标识
32                     flag[i] <= `UD 1'b1;
33                 else if (cnt[i]== BTN_DELAY) //持续 20ms 后归零
34                     flag[i] <= `UD 1'b0;
35                 else
36                     flag[i] <= `UD flag[i];
37             end
38
39             always @(posedge clk)
40             begin
41                 if(cnt[i]== BTN_DELAY) //计数 20ms 时归零
42                     cnt[i] <= `UD 18'd0;
43                 else if(flag[i]) //抖动区间有效时计数
44                     cnt[i] <= `UD cnt[i] + 1'b1;
45                 else //非抖动区间保持 0
46                     cnt[i] <= `UD 18'd0;
47             end
48
49             always @(posedge clk)
50             begin
51                 if(flag[i]) //抖动区间，消抖输出保持
52                     btn_deb_fix[i] <= `UD btn_deb_fix[i];
53                 else //非抖动区间，按键状态传递到消抖输出
54                     btn_deb_fix[i] <= `UD btn_in[i];
55             end
56         endgenerate

```

Verilog 中的 generate 语句常用于编写可配置的、可综合的 RTL 的设计结构。它可用于创建模块的多个实例化，或者有条件的实例化代码块。

我们常用 generate 语句做三件事情。一个是用来构造循环结构，用来多次实例化某个模块。一个是构造条件 generate 结构，用来在多个块之间最多选择一个代码块，条件 generate 结构包含 if--generate 结构和 case--generate 形式。还有一个是用来断言。

在 Verilog 中，generate 在建模 (elaboration) 阶段实施，出现预处理之后，正式模拟仿真之前。因此，generate 结构中的所有表达式都必须是常量表达式，并在建模 (elaboration) 时确定。例如，generate 结构可能受参数值的影响，但不受动态变量的影响。

Verilog 中的 generate 块创建了新的作用域和新的层次结构，就像实例化模块一样。因此在尝试对 generate 块中的信号进行引用时，很容易因此混乱，因此请记住这一点。

(参考来源: 知乎作者数字 IC 小站, [Verilog 中 generate 的使用 - 知乎 \(zhihu.com\)](https://zhuanlan.zhihu.com/p/100000000))

2. 4. 4LED 控制模块

```

1  `timescale 1ns / 1ps
2  `define UD #1
3  module led(
4      input      clk,
5      input  [1:0] ctrl,
6      output [7:0] led
7  );
8
9      reg [24:0] led_light_cnt = 25'd0;
10     reg [ 7:0] led_status = 8'b1000_0000;
11
12     // time counter
13     always @(posedge clk)
14     begin
15         if(led_light_cnt == 25'd19_999_999)
16             led_light_cnt <= `UD 25'd0;
17         else
18             led_light_cnt <= `UD led_light_cnt + 25'd1;
19     end
20
21     reg [1:0] ctrl_1d; //保存上一个 led 状态周期的 ctrl 值
22     always @(posedge clk)
23     begin
24         if(led_light_cnt == 25'd19_999_999)
25             ctrl_1d <= ctrl; //此处设计能保证状态切换时, 从 0 时刻开始下一次流水状态
26     end
27
28     // led status change
29     always @(posedge clk)
30     begin
31         if(led_light_cnt == 25'd19_999_999) //0.5s 周期
32         begin
33             case(ctrl)
34                 2'd0 : //从高位到低位的 led 流水灯
35                 begin
36                     if(ctrl_1d != ctrl)
37                         led_status <= `UD 8'b1000_0000;
38                     else
39                         led_status <= `UD {led_status[0], led_status[7:1]};
40                 end
31                 2'd1 : //隔一亮一交替
42                 begin
43                     if(ctrl_1d != ctrl)
44                         led_status <= `UD 8'b1010_1010;
45                     else
46                         led_status <= `UD ~led_status;
47                 end
48             end

```

```
49         2'd2 : //从高位到低位暗灯流水
50         begin
51             if(ctrl1_d != ctrl )
52                 led_status <= `UD 8'b0111_1111;
53             else
54                 led_status <= `UD {led_status[0],led_status[7:1]};
55             end
56         endcase
57     end
58 end
59
60 assign led = led_status;
61
62 endmodule
63
```

2.5 实验现象

每按下一次 KEY1，LED 灯状态切换一次，总共三种 LED 模式供循环切换；

LED 模式一：从高位到低位的 LED 流水灯；

LED 模式二：隔一亮一交替点亮；

LED 模式三：从高位到低位暗灯流水；