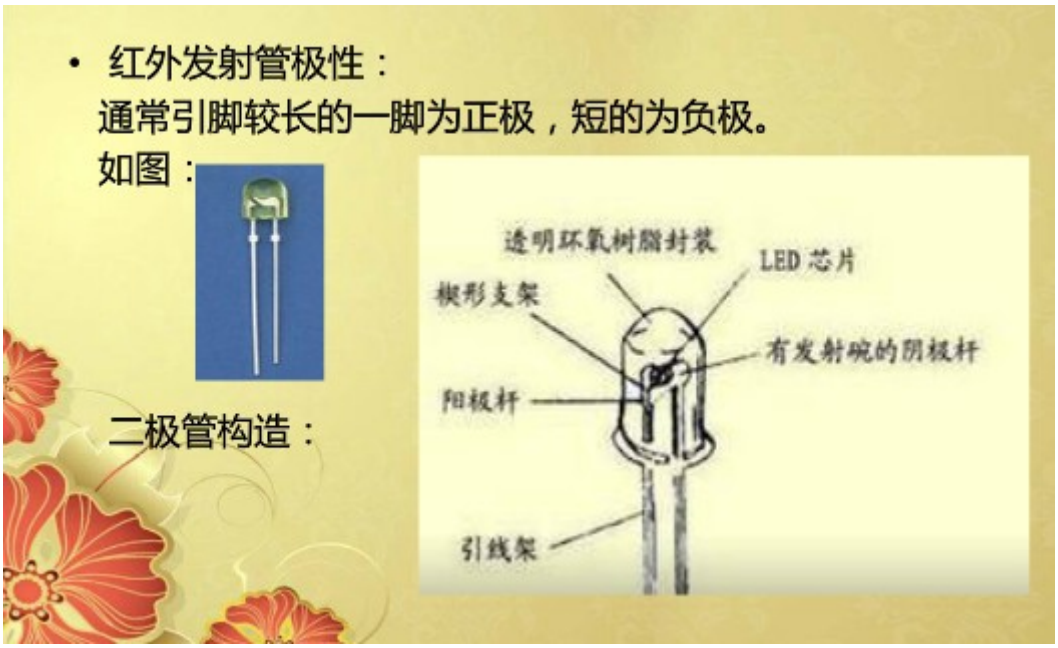
# 红外解码编码学习----verilog

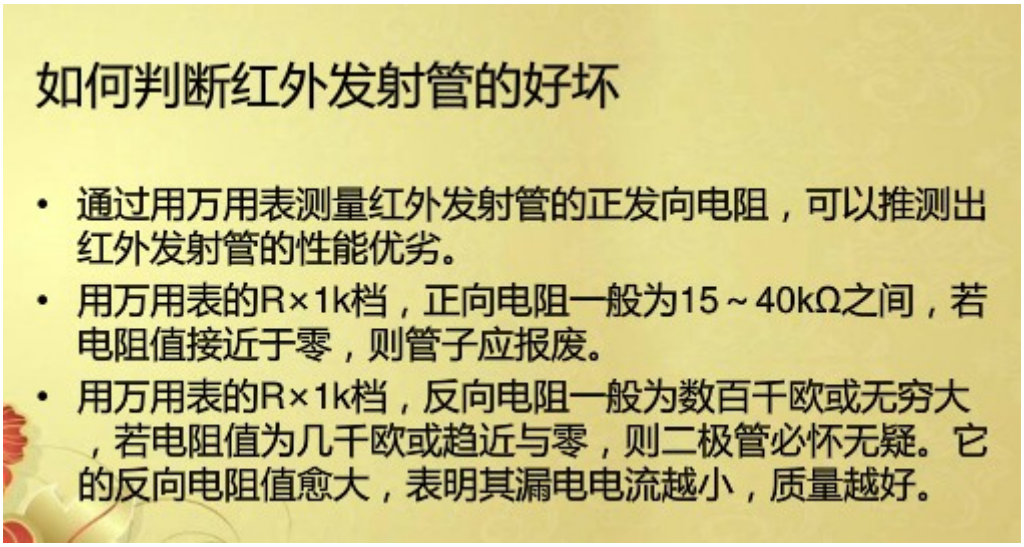
在设计中运用红外遥控器可以很好的解决按键缺少的问题，还可以方便的控制产品。

红外发射部分：

红外发射管：

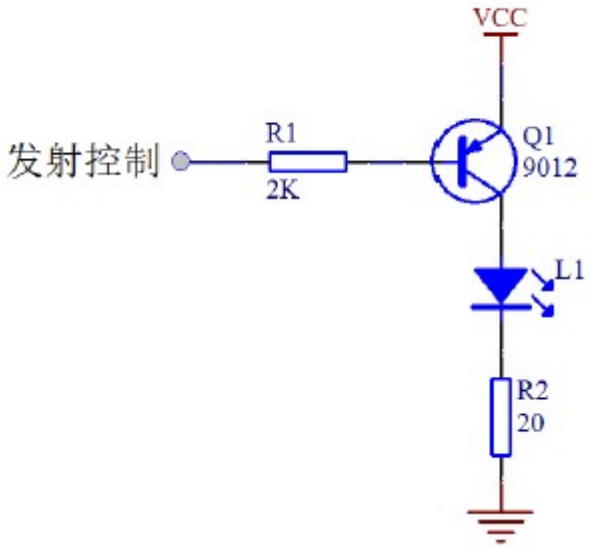



判断红外发射管的好坏



:

电路原理图：




发射部分：当发射控制输出高电平时，三极管 Q1 不导通，红外发射管 L1 不会发射红外信号；当发射控制输出低电平的时候，通过三极管 Q1 导通让 L1 发出红外光。

我们平时用到的红外遥控器里的红外通信，通常是使用 38K 左右的载波进行调制的，下面我把原理大概给大家介绍一下，先看发送部分原理。

调制：就是用待传送信号去控制某个高频信号的幅度、相位、频率等参量变化的过程，即用一个信号去装载另一个信号。比如我们的红外遥控信号要发送的时候，先经过 38K 调制，如图 16-4 所示。

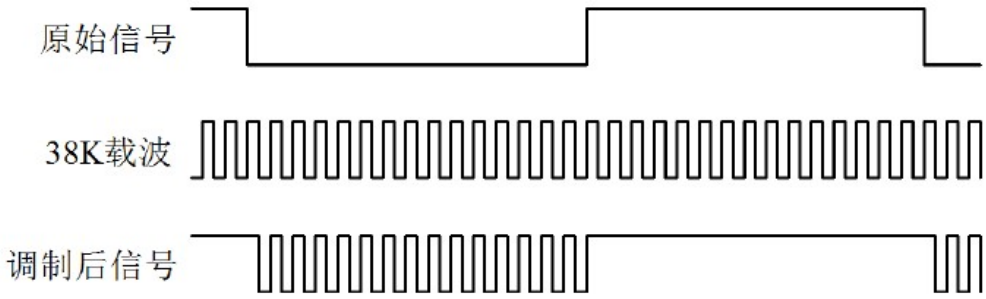



图　　红外信号调制

原始信号就是我们要发送的一个数据“0”位或者一位数据“1”位，而所谓 38K 载波就是频率为 38K 的方波信号，调制后信号就是最终我们发射出去的波形。我们使用原始信号来控制 38K 载波，当信号是数据“0”的时候，38K 载波毫无保留的全部发送出去，当信号是数据“1”的时候，不发送任何载波信号。

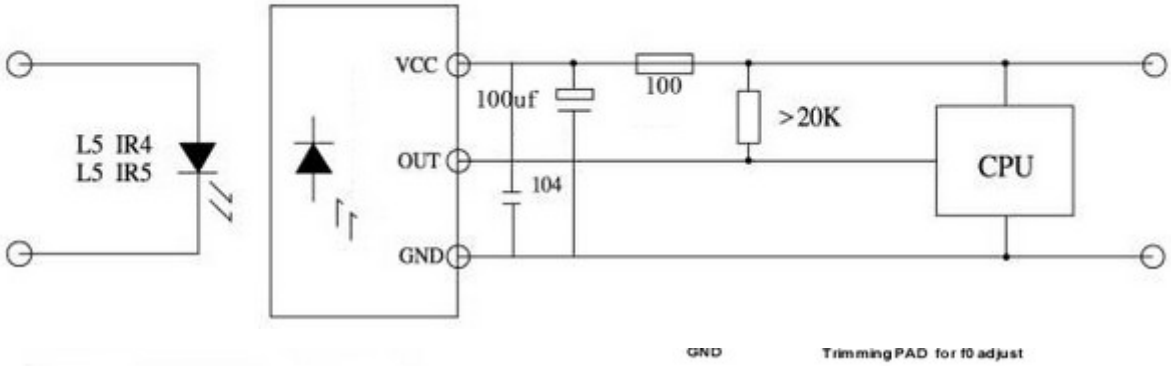
接收部分：

**型号:1838T（深圳兰丰科技产红外线接收头、发射管、发光二极管等光电系列产品）**

1. 特性：
   - ●小型设计；
   - ●内置专用 IC；
   - ●宽角度及长距离接收；
   - ●抗干挠能力强；
   - ●能抵挡环境干挠光线；
   - ●低电压工作；
2. 应用：
   - ■视听器材（音箱，电视，录影机，碟机）
   - ■家庭电器（冷气机，电风扇，电灯）
   - ■其它红外线遥控产品；

**型号：1838T**

4. 应用电路图：

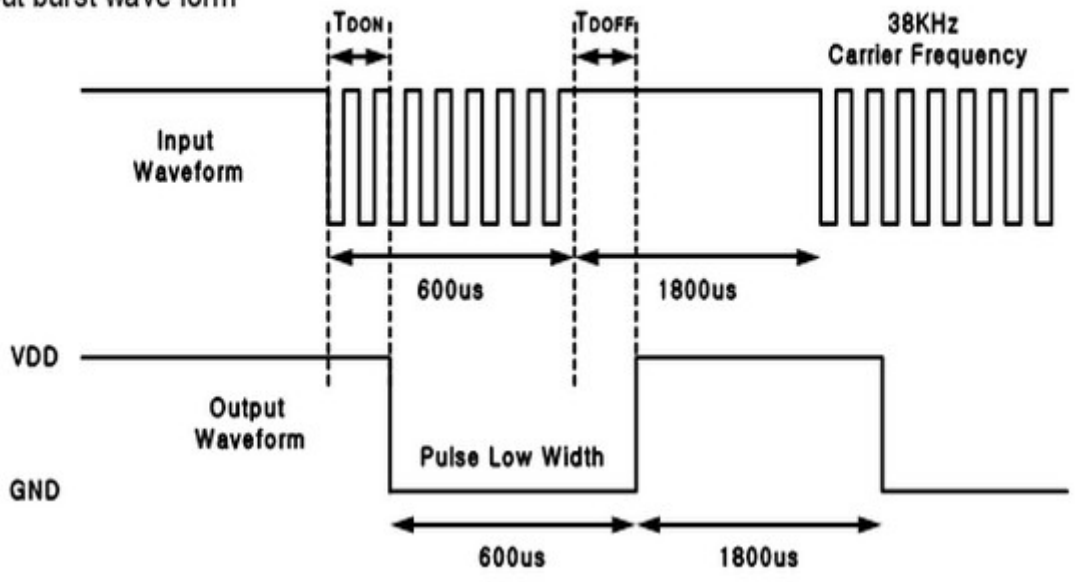



6. 光电参数(T=25℃ Vcc=5v $f_0$=38KHZ)：

| 参 数 | 符号 | 测试条件 | Min | Type | Max | 单 位 |
|---|---|---|---|---|---|---|
| 工作电压 | $V_{cc}$ | | 2.7 | | 5.5 | V |
| 工作电流 | Icc | | 0.6 | 0.8 | - | mA |
| 静态电流 | Ice | 无信号输入时 | 0.1 | | 0.5 | mA |
| 接收距离 | L | ※ | 22 | 25 | | M |
| 接收角度 | θ1/2 | | | +/-35 | | Deg |
| 载波频率 | $f_0$ | | | 38 | | KHZ |
| BMP 宽度 | $f_{BW}$ | -3Db Bandwidth | - | 8 | - | kHz |
| 低电平输出 | $V_{OL}$ | Vin=0V Vcc=5V | | | 0.4 | V |
| 高电平输出 | $V_{OH}$ | Vcc=5V | Vcc-0.3 | | Vcc | V |
| 输出脉冲宽度 | $T_{PWL}$ | Vin=50mVp-p | 500 | 600 | 700 | μS |
| | $T_{PWH}$ | Vin=50mVp-p | 540 | 640 | 740 | μS |

※ 光轴上测试,以宽度 600/900μs 为发射脉冲,在 5CM 之接收范围内，取 50 次接收脉冲之平均值。

Note 1:
Input burst wave form




传输的NEC协议：

NEC 协议的数据格式包括了引导码、用户码、用户码（或者用户码反码）、按键键码和键码反码，最后一个停止位。停止位主要起隔离作用，一般不进行判断，编程时我们也不予理会。其中数据编码总共是 4 个字节 32 位，如图 16-7 所示。第一个字节是用户码，第二个字节可能也是用户码，或者是用户码的反码，具体由生产商决定，第三个字节就是当前按键的键数据码，而第四个字节是键数据码的反码，可用于对数据的纠错。

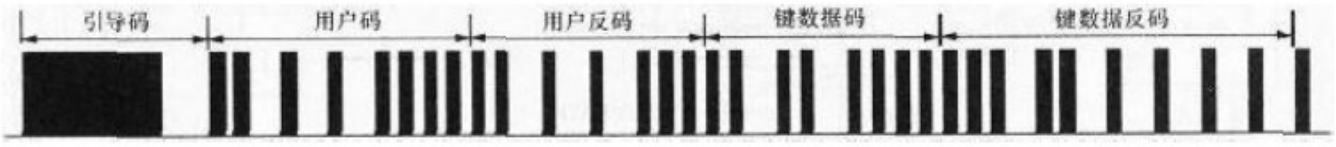



图 16-7　NEC 协议数据格式

这个 NEC 协议，表示数据的方式不像我们之前学过的比如 UART 那样直观，而是每一位数据本身也需要进行编码，编码后再进行载波调制。

引导码：9ms 的载波+4.5ms 的空闲。

比特值“0”：560us 的载波+560us 的空闲。

比特值“1”：560us 的载波+1.68ms 的空闲。

结合图 16-7 我们就能看明白了，最前面黑乎乎的一段，是引导码的 9ms 载波，紧接着是引导码的 4.5ms 的空闲，而后边的数据码，是众多载波和空闲交叉，它们的长短就由其要传递的具体数据来决定。HS0038B 这个红外一体化接收头，当收到有载波的信号的时候，会输出一个低电平，空闲的时候会输出高电平，我们用逻辑分析仪抓出来一个红外按键通过 HS0038B 解码后的图形来了解一下，如图 16-8 所示。

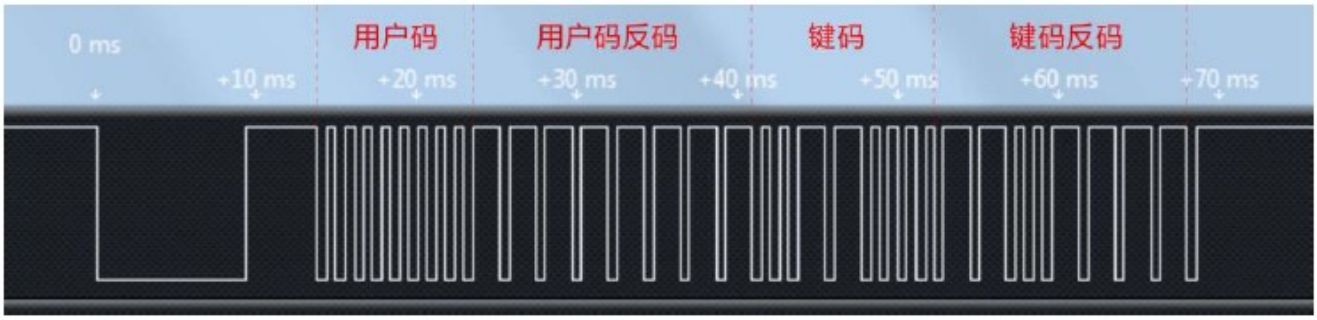
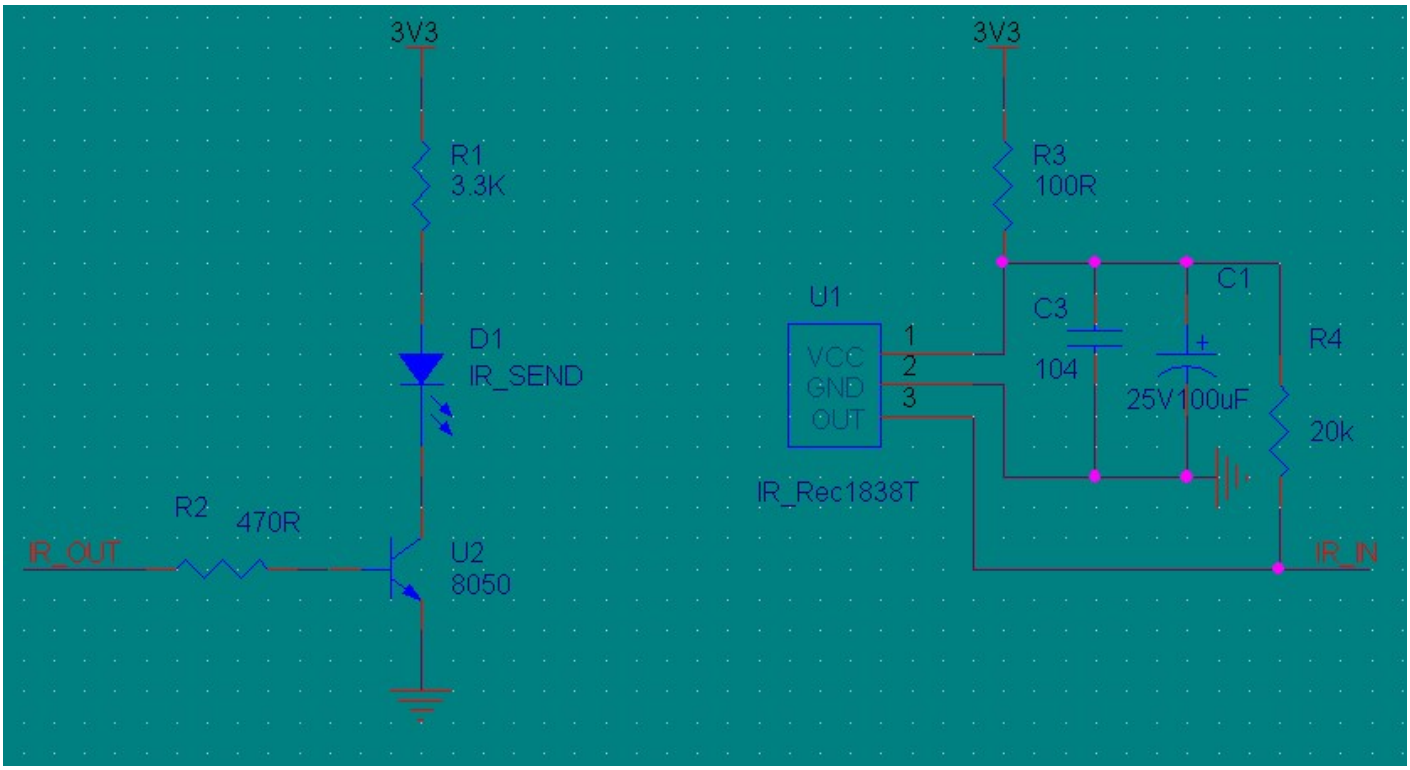



图 16-8　红外遥控器按键编码

从图上可以看出，先是 9ms 载波加 4.5ms 空闲的起始码，数据码是低位在前，高位在后，数据码第一个字节是 8 组 560us 的载波加 560us 的空闲，也就是 0x00，第二个字节是 8 组 560us 的载波加 1.68ms 的空闲，可以看出来是 0xFF，这两个字节就是用户码和用户码的反码。按

本实验电路：




verilog 程序：

发送程序：

```
/*********************************Copyright**************************************
**----------------------------File information-------------------------
** File name  :IR_send.v
** CreateDate :2015.06
** Funtions   :红外信号的发送程序,发送格式：引导码+8bit用户码+8bit用户反码（或者用户反码）+8bit数据码+8b
** Operate on :M5C06N3L114C7
** Copyright  :All rights reserved.
** Version    :V1.0
**----------------------------Modify the file information----------------
** Modified by   :
** Modified data :
** Modify Content:
*********************************************************************************/

module  IR_send  (
           clk,
           rst_n,

                    key_1,
                    key_2,

                    IR_out,
                    led_1,
                    led_2,
                    led_3,
                    led_4
```

```
//                          testdata
                );
    input           clk;    //24M/20m
    input           rst_n;

    input           key_1;
    input           key_2;

    output          IR_out;

    output  reg     led_1;
    output  reg     led_2;
   output       led_3;
    output       led_4;

//     output  [7;0]  testdata;
    //--------------------//

    `define     CLK_20M
//    `define     CLK_24M
//    `define     CLK_50M


 `ifdef      CLK_20M
             parameter t_38k    = 10'd526;
             parameter t_38k_half = 10'd263;
             parameter t_9ms    = 18'd179999;
             parameter t_4_5ms  = 17'd89999;
             parameter t_13_5ms = 19'd269999;
             parameter t_560us  = 14'd11199;
             parameter t_1_12ms = 15'd22399;
             parameter t_1_68ms = 16'd33599;
             parameter t_2_24ms = 16'd44799;
     `endif

    `ifdef      CLK_24M
             parameter t_38k    = 10'd630;
             parameter t_38k_half = 10'd315;
             parameter t_9ms    = 18'd215999;
             parameter t_4_5ms  = 17'd107999;
             parameter t_13_5ms = 19'd323999;
             parameter t_560us  = 14'd13439;
             parameter t_1_12ms = 15'd26879;
             parameter t_1_68ms = 16'd40319;
             parameter t_2_24ms = 16'd53759;
     `endif

    `ifdef      CLK_50M
             parameter t_38k    = 11'd1315;
             parameter t_38k_half = 10'd657;
             parameter t_9ms    = 19'd449999;
             parameter t_4_5ms  = 18'd224999;
             parameter t_13_5ms = 20'd674999;
             parameter t_560us  = 15'd27999;
             parameter t_1_12ms = 16'd55999;
```

```
            parameter t_1_68ms = 17'd83999;
            parameter t_2_24ms = 17'd111999;
     `endif

    parameter DATA_USER = 8'h00;

    //---------------------------------//
     //分频38Khz时钟
     reg  [10:0]   cnt1;
       wire         clk_38k;
       always @(posedge clk or negedge rst_n)
        begin
         if(!rst_n)
          begin
              cnt1 <= 0;
           end
         else if(cnt1 == t_38k)
           begin
              cnt1 <= 0;
           end
           else cnt1 <= cnt1 + 1;
        end
  assign  clk_38k = (cnt1<t_38k_half)?1:0;
 //-----------------------------------------//

//    wire         key_1_flg;
//        wire         key_2_flg;
//      key_shake   U1(
//            .clk_100M(clk),
//            .rst_n(rst_n),
//
//            .key_in(key_1),
//            .key_out(key_1_flg)
//             );
//
//        key_shake  U2(
//            .clk_100M(clk),
//            .rst_n(rst_n),
//
//            .key_in(key_2),
//            .key_out(key_2_flg)
//             );
   reg   [2:0]      key_1_flag;
     wire             key_1_neg;
     wire             key_1_pos;
     always @(posedge clk or negedge rst_n)
     begin
      if(!rst_n)
       begin
           key_1_flag <= 3'b000;
        end
      else
        begin
           key_1_flag <= {key_1_flag[1:0],key_1};
        end
      end
```

```
    assign key_1_pos = (key_1_flag[2:1]== 2'b01);


     reg   [2:0]      key_2_flag;
     wire             key_2_neg;
     wire             key_2_pos;
     always @(posedge clk or negedge rst_n)
     begin
      if(!rst_n)
       begin
           key_2_flag <= 3'b000;
        end
       else
        begin
           key_2_flag <= {key_2_flag[1:0],key_2};
        end
       end
    assign key_2_pos = (key_2_flag[2:1] == 2'b01);

//-----------------------------------------//
    parameter  IDEL       = 3'D0;         //初始化状态，等待发送命令
        parameter  START      = 3'D1;         //开始发送引导码
        parameter  SEND_USER  = 3'D2;         //发送用户码
        parameter  SEND_UNUSER= 3'D3;         //发送用户反码
        parameter  SEND_DATA  = 3'D4;         //发送数据
        parameter  SEND_UNDATA= 3'D5;         //发送数据反码
        parameter  FINISH     = 3'D6;         //发送结束码
        parameter  FINISH_1   = 3'D7;         //发送结束
    reg   [2:0]     state;
    reg             start_en;
        wire            start_over;
        reg             zero_en;
        wire            zero_over;
        reg             one_en;
        wire            one_over;
         reg             finish_en;
        wire            finish_over;
        reg             sendover;
        reg   [7:0]     shiftdata;
        reg   [3:0]     i;
        reg   [7:0]     DATA;
        always @(posedge clk or negedge rst_n)
         begin
          if(!rst_n)
           begin
              state <= IDEL;
                    start_en <= 0;
                    zero_en <= 0;
                    one_en <= 0;
                    finish_en <= 0;
                    sendover <= 0;
                    shiftdata <= 0;
                    i <= 0;
                    DATA <= 8'D0;
```

```
                led_1 <= 1;
                led_2 <= 1;
          end
        else
          begin
             case(state)
                  IDEL:
                     begin
                       start_en <= 0;
                             zero_en <= 0;
                             one_en <= 0;
                             finish_en <= 0;
                             sendover <= 0;
                             shiftdata <= 0;
                             i <= 0;
                             DATA <= 8'd0;

                             if(key_1_pos)
                               begin
                                  state <= START;
                                      DATA <= 8'd1;
                                  end
                              else if(key_2_pos)
                                  begin
                                          state <= START;
                                          DATA <= 8'd2;
                                     end
                                else state <= IDEL;
                        end
                    START:                //发送引导码
                         begin

                                if(start_over)
                                  begin
                                        start_en <= 0;
                                        state <= SEND_USER;
                                        shiftdata <= DATA_USER;
                                     end
                                else
                                     begin
                                        start_en <= 1;
                                        state <= START;
                                     end
                           end
                      SEND_USER:
                        begin
//                                    led_3 <= 1;
                                   if((i==7)&&(one_over||zero_over))   //结束位
                                      begin
                                            i <=0;
                                     shiftdata <= ~DATA_USER;
                                            state <= SEND_UNUSER;
                                              one_en <= 0;
                                            zero_en <= 0;
                                       end
```

```
                    else
                      begin
                            if(zero_over||one_over)    //1bit发送结束
                              begin
                                i <= i + 1;
                                    one_en <= 0;
                                    zero_en <= 0;
                               end
                            else if(shiftdata[i])
                                begin
                                    one_en <= 1;
                                end
                            else if(!shiftdata[i]) zero_en <= 1;
                          else
                                begin
                                 i <= i ;
                                    one_en <= one_en;
                                    zero_en <= zero_en;
                                end
                        end
                end
            SEND_UNUSER:
               begin
                        led_1 <= ~led_1;
                     if((i==7)&&(one_over||zero_over))  //结束位
                           begin
                                i <=0;
                                state <= SEND_DATA;
                                  shiftdata <= DATA;
                                  one_en <= 0;
                                  zero_en <= 0;

                             end
                    else
                      begin
                            if(zero_over||one_over)    //1bit发送结束
                              begin
                                 i <= i + 1;
                                    one_en <= 0;
                                    zero_en <= 0;
                                end
                            else if(shiftdata[i])
                                begin
                                    one_en <= 1;
                                end
                            else if(!shiftdata[i]) zero_en <= 1;
                          else
                                begin
                                 i <= i ;
                                    one_en <= one_en;
                                    zero_en <= zero_en;
                                end
                        end

                end
```

```
                SEND_DATA:
                   begin
                             led_2 <= ~led_2    ;
                            if((i==7)&&(one_over||zero_over))  //结束位
                                begin
                                      i <=0;
                                      state <= SEND_UNDATA;
                                        shiftdata <= ~DATA;
                                        one_en <= 0;
                                        zero_en <= 0;
                                 end
                            else
                              begin
                                    if(zero_over||one_over)   //1bit发送结束
                                      begin
                                        i <= i + 1;
                                             one_en <= 0;
                                             zero_en <= 0;
                                       end
                                    else if(shiftdata[i])
                                        begin
                                             one_en <= 1;
                                       end
                                    else if(!shiftdata[i]) zero_en <= 1;
                                   else
                                        begin
                                         i <= i ;
                                             one_en <= one_en;
                                             zero_en <= zero_en;
                                       end
                                end

                        end
            SEND_UNDATA:
                begin

                        if((i==7)&&(one_over||zero_over))  //结束位
                                begin
                                      i <=0;
                                      shiftdata <= 0;
                                      state <= FINISH;
                                        one_en <= 0;
                                        zero_en <= 0;
                                 end
                            else
                              begin
                                    if(zero_over||one_over)   //1bit发送结束
                                      begin
                                         i <= i + 1;
                                             one_en <= 0;
                                             zero_en <= 0;
                                       end
                                    else if(shiftdata[i])
                                        begin
                                             one_en <= 1;
```

```
                                        end
                                    else if(!shiftdata[i]) zero_en <= 1;
                                else
                                        begin
                                        i <= i ;
                                            one_en <= one_en;
                                            zero_en <= zero_en;
                                        end
                                end

                        end
                    FINISH:
                       begin
                                if(finish_over)
                                begin
                                     finish_en <= 0;
                                     state <= FINISH_1;
                                  end
                               else
                                  begin
                                     finish_en <= 1;
                                     state <= FINISH;
                                  end

                                end
                    FINISH_1:
                          begin

                                  sendover <= 1;
                                    state <= IDEL;

                          end

                  default: state <= IDEL;
                endcase
          end
        end



//---------------------------------------------//
   //引导码, 9ms载波加4.5ms空闲
   reg    [19:0]     cnt2;
   wire              start_flag;
   always @(posedge clk or negedge rst_n)
   begin
    if(!rst_n)
     begin
         cnt2 <= 0;
      end
    else if(start_en)
      begin
        if(cnt2 >= t_13_5ms)  cnt2 <= t_13_5ms+1;
          else cnt2 <= cnt2 + 1;
          end
```

```
        else cnt2  <= 0;
    end
   assign start_over = (cnt2 == t_13_5ms)?1:0;
   assign start_flag = (start_en&&(cnt2 <= t_9ms))?1:0;

//-----------------------------------------------//
    //比特0,  560us载波 + 560us空闲
    reg     [15:0]      cnt3;
    wire                zero_flag;
    always @(posedge clk or negedge rst_n)
    begin
     if(!rst_n)
      begin
          cnt3 <= 0;
      end
     else if(zero_en)
       begin
         if(cnt3 >= t_1_12ms)  cnt3 <= t_1_12ms+1;
           else cnt3 <= cnt3 + 1;
           end
       else cnt3  <= 0;
    end
   assign zero_over = (cnt3 == t_1_12ms)?1:0;
   assign zero_flag = (zero_en&&(cnt3 <= t_560us))?1:0;

 //-----------------------------------------------//
    //比特1,  560us载波 + 1.68ms空闲
    reg     [16:0]      cnt4;
    wire                one_flag;
    always @(posedge clk or negedge rst_n)
    begin
     if(!rst_n)
      begin
          cnt4 <= 0;
      end
     else if(one_en)
       begin
         if(cnt4 >= t_2_24ms)  cnt4 <= t_2_24ms+1;
           else cnt4 <= cnt4 + 1;
           end
       else cnt4  <= 0;
    end
   assign one_over = (cnt4 == t_2_24ms)?1:0;
   assign one_flag = (one_en&&(cnt4 <= t_560us))?1:0;

  //-----------------------------------------------//
    //结束码,  560us载波
    reg     [14:0]      cnt5;
    wire                finish_flag;
    always @(posedge clk or negedge rst_n)
    begin
     if(!rst_n)
      begin
          cnt5 <= 0;
      end
```

```
     else if(finish_en)
       begin
         if(cnt5 >= t_560us)  cnt5 <= t_560us+1;
           else cnt5 <= cnt5 + 1;
           end
       else cnt5  <= 0;
    end
   assign finish_over = (cnt5 == t_560us)?1:0;
   assign finish_flag = (finish_en&&(cnt5 <= t_560us))?1:0;

//-----------------------------------//
wire   ir_out;
assign ir_out = start_flag||zero_flag||one_flag||finish_flag;
assign IR_out = ir_out&&clk_38k;

assign led_3 = i[1];
assign led_4 = i[0];

   endmodule
```

接收程序：

```
   /*******************************Copyright*************************************
   **-----------------------------File information--------------------------
   ** File name  :ir_resive.v
   ** CreateDate :2015.06
   ** Funtions   : 中断接收程序。结束数据为：引导码+用户码+用户反码+数据码+数据反码。
   ** Operate on :M5C06N3L114C7
   ** Copyright  :All rights reserved.
   ** Version    :V1.0
   **-----------------------------Modify the file information----------------
   ** Modified by   :
   ** Modified data :
   ** Modify Content:
   ***************************************************************************/

    module  IR_resive  (
              clk,
              rst_n,

                       ir_in,

                       led_error,

                       led_5,
                       led_6,
                       led_7,
```

```
                        test_data
                );
    input           clk;
    input           rst_n;

    input           ir_in;

    output          led_error;

    output         led_5;
    output         led_6;
    output          led_7;


    output  [7:0]  test_data;
    //-----------------------------------------//
      reg    [2:0]    ir_in_reg;
        wire            ir_in_pos;
        wire            ir_in_neg;
        wire            ir_in_change;
        always @(posedge clk or negedge rst_n)
        begin
         if(!rst_n)
          begin
              ir_in_reg <= 3'b111;
           end
         else
           begin
             ir_in_reg <= {ir_in_reg[1:0],ir_in};
           end
         end
       assign     ir_in_pos = (ir_in_reg[2:1] == 2'b01)?1:0;
       assign     ir_in_neg = (ir_in_reg[2:1] == 2'b10)?1:0;
       assign  ir_in_change = ir_in_pos|ir_in_neg;

    //-------------------------------------//
    //设计分频和计数部分：从1838T的 技术手册中，可以得出最小的脉冲持续时间为500us,在采样时可以对最小的电平采样1
   //则每次的采样间隔时间是  500us/16=31.25us     时钟频率为FCLK = X MHZ,  则最小采样计数为：N =31.25*X,
    //然后再用一个计数器计数同一电平的采样计数时间。
    //最后判断是leader的9ms 还是4.5ms，或是数据的 0 还是 1。
    //---------------------------------------------------------//
        `define     FCLK_20M
//          `define     FCLK_24M `

        `ifdef    FCLK_20M
         parameter   t_31_25us = 10'd625;
               parameter   t_100k = 200;
           `endif

           `ifdef    FCLK_24M
         parameter   t_31_25us = 10'd750; //
               parameter   t_100k = 240;
           `endif

           `ifdef    FCLK_50M
         parameter   t_31_25us = 11'd1562; //
```

```
            parameter   t_100k = 500;
        `endif

          parameter   t_low_H = 26;
            parameter   t_low_L = 13;       //16
            parameter   t_high_H=67;        //54  1.7MS左右
            parameter   t_high_L=35 ;
            parameter   t_9ms_H   =9'd398;   //288
            parameter   t_9ms_L   =9'd278;
            parameter   t_4_5ms_H =9'd154;   //144
            parameter   t_4_5ms_L =9'd134;

            parameter   t_watch  = 9'd500;   //定时，计数达到500，则已经跑飞，

            parameter   t_1s     = 16'd31999;



    //---------------------------------------------//
     reg              idel_flag;
     reg     [10:0]    cnt;
     always @(posedge clk or negedge rst_n)
     begin
      if(!rst_n)
       begin
          cnt <= 0;
        end
        else if(idel_flag) cnt <= 0;         //空闲状态，不再变化
      else if(ir_in_change)  cnt <= 0;
      else if(cnt == t_31_25us) cnt <= 0;
        else cnt<= cnt + 1;
     end

reg    [8:0]        cnt1;
always @(posedge clk or negedge rst_n)
begin
 if(!rst_n)
  begin
     cnt1 <= 0;
   end
   else if(idel_flag) cnt1 <= 0;
 else if(ir_in_change)  cnt1 <= 0;
 else if(cnt == t_31_25us) cnt1 <= cnt1 + 1;
   else cnt1 <= cnt1;
end


      wire          t_9ms_flag;
       wire          t_4_5_ms_flag;
       wire          short_flag;     //短电平，可是高电平也可以是低电平
       wire          long_flag;      //长电平，肯定是高电平
       assign  t_9ms_flag = ((cnt1 > t_9ms_L)&&(cnt1 < t_9ms_H));
       assign  t_4_5_ms_flag = ((cnt1 > t_4_5ms_L)&&(cnt1 < t_4_5ms_H));
       assign  long_flag = ((cnt1 > t_high_L)&&(cnt1 < t_high_H));
       assign  short_flag = ((cnt1 > t_low_L)&&(cnt1 < t_low_H));
```

```
            wire          watchdog;
            assign  watchdog =  (cnt1 > t_watch)?1:0;
    //-----------------------------------------//
    parameter   IDEL    = 4'd0;
    parameter   L_9MS   = 4'd1;
    parameter   L_4_5MS = 4'd2;
    parameter   DATA_R  = 4'd4;
    parameter   FINISH_R= 4'd8;

   reg      [3:0]      state;
     reg      [31:0]     shiftdata;
     reg      [5:0]      n;
     reg                 error_flag;
     reg                 r_over;
     reg      [31:0]     rdata;
 always @(posedge clk or negedge rst_n)
 begin
  if(!rst_n)
   begin
      state <= IDEL;
            shiftdata <= 0;
            n <= 0;
            error_flag <= 0;
            rdata <= 0;
            r_over<= 0;
            idel_flag <= 0;

//            led_5 <= 1;
//            led_6 <= 1;
    end
//  else  if(watchdog) state <= IDEL;
    else
    begin
      case(state)
             IDEL:
               begin

                     idel_flag <= 1;    //空闲状态
                       shiftdata <= 0;
                n <= 0;
                error_flag <= 0;
                      r_over<= 0;
                      if(ir_in_reg[1] ==0)
                       begin
                            state <= L_9MS;  //检测到拉低数据线
                            idel_flag <= 0;
                        end
                      else  state <= IDEL;
                  end
            L_9MS:         //9ms为低电平，数据线拉高时结束
              begin
//                    led_5 <= 0;
                  if(watchdog) state <= IDEL;
                  else if(ir_in_pos)
```

```
                    begin
                        if(t_9ms_flag)  state <= L_4_5MS;
                        else
                            begin
                                state <= IDEL;
                                error_flag <= 1;
                               end
                     end
                   else   state <= L_9MS;
               end
           L_4_5MS:
             begin
                if(watchdog) state <= IDEL;
                   else if(ir_in_neg)
                    begin
                        if(t_4_5_ms_flag)  state <= DATA_R;
                        else
                            begin
                                state <= IDEL;
                                error_flag <= 1;
                               end
                     end
                   else   state <= L_4_5MS;
            end
           DATA_R:
             begin
//                    led_6 <= 0;
                if(watchdog) state <= IDEL;
                 else if((n == 32)&&(ir_in_reg[2:1] == 2'b11))  state <= FINISH_R;
                   else if(ir_in_pos)
                       begin
                           if(short_flag)  state <= DATA_R;
                           else
                               begin
                                   state <= IDEL;
                                   error_flag <= 1;
                                  end
                        end
                   else if(ir_in_neg)
                       begin
                              n<= n + 1;
                              if(short_flag) shiftdata[n] <= 0;          //从低位到高位依次接收，这样数据
                              else if(long_flag)  shiftdata[n] <= 1;   //从原来的{用户码，用户反码，数据
                              else                                       //所以要调整数据位置
                                  begin
                                   state <= IDEL;
                                   error_flag <= 1;
                                  end
                       end
                   else   state <= DATA_R;
             end
           FINISH_R:
             begin
                   r_over <= 1;
                   rdata <= {shiftdata[7:0],shiftdata[15:8],shiftdata[23:16],shiftdata[31:24]};
```

```
                    state <= IDEL;
                end
            default:state <= IDEL;
          endcase
    end
  end

     //-------------------------------------------//
     assign led_5 = (rdata[15:8] == 8'h01)?0:1;
     assign led_6 = (rdata[15:8] == 8'h02)?0:1;
   assign led_7 = (rdata[31:24] == 8'h00)?0:1;
     assign test_data = rdata[15:8];
//  assign test_data = {error_flag,t_4_5_ms_flag,ir_in_change,ir_in_reg[1],state[3:0]};
    endmodule
```

按键消抖（与以前有修改）：

```
/*********************************Copyright**************************************
**----------------------------File information--------------------------
** File name  :key_shake.v
** CreateDate :2015.03
** Funtions   : 按键的消抖操作：在复位之后的100us内，不响应按键的操作，在之后有按键按下后，有20ms的延迟，之后输
** Operate on :M5C06N3L114C7
** Copyright  :All rights reserved[F].
** Version    :V1.0
**---------------------------Modify the file information----------------
** Modified by   :
** Modified data :             2015.04                              2015.06
** Modify Content:V1.1:clk-->clk_100M, 常数声明放到一起，便于修改。      增加实用20Mclk,并将输出修改为连续
*********************************************************************************/

 module  key_shake (
           clk_100M,
           rst_n,

            key_in,
            key_out
             );
 input          clk_100M;            //100Mhz
 input          rst_n;

 input          key_in;
 output         key_out;

 //-------------------------------------

 //在复位之后的100us内，不响应按键的操作
```

```
//  `define   CLK_100M
    `define   CLK_20M

 `ifdef  CLK_100M
           parameter    t_100us  = 14'd9999;
           parameter    t1ms = 17'd99999;       //定时1ms
           parameter    t_20ms = 5'd20;
  `endif

  `ifdef  CLK_20M
           parameter    t_100us  = 14'd1999;
           parameter    t1ms = 17'd19999;       //定时1ms
           parameter    t_20ms = 5'd20;
  `endif

  reg    [13:0]   cnt;
    reg             key_en;         //复位之后允许按键输入标志
 always @(posedge clk_100M or negedge rst_n)
 begin
  if(!rst_n)
   begin
      cnt <= 0;
            key_en <=0;
    end
  else
    begin
      if(cnt == t_100us)
             begin
                  key_en <= 1;
               end
       else
             begin
                   key_en <= 0;
                  cnt <= cnt + 1;
             end
    end
  end

 //-------------------------------------------------
 wire         HtoL_flag;         //下降沿标志
 wire         LtoH_flag;         //上升沿标志
 reg   [2:0]   key_reg;
 always @(posedge clk_100M or negedge rst_n)
 begin
  if(!rst_n)
   begin
      key_reg <= 3'b111;            //默认没按下状态为高，按下之后为低.反之则为3'b000;
    end
  else
    begin
      key_reg <= {key_reg[1:0],key_in};
    end
  end
```

```
 assign HtoL_flag = key_en?(key_reg[2:1] == 2'b10):0;              //下降沿检测，一个时钟的高电平
 assign LtoH_flag = key_en?(key_reg[2:1] == 2'b01):0;                 //上升沿检测，一个时钟的高电平
//-----------------------------------------
 reg          cnt_en;                  //计数使能标志

 reg   [16:0]  cnt2;
 always @(posedge clk_100M or negedge rst_n)
 begin
  if(!rst_n)
   begin
      cnt2 <= 17'd0;
    end
  else if((cnt_en)&&(cnt2 == t1ms))
    begin
      cnt2 <= 17'd0;
    end
    else if(cnt_en)
    begin
      cnt2 <= cnt2 + 17'd1;
    end
    else
          cnt2 <= 17'd0;
  end

 reg   [4:0]   cnt3;
 always @(posedge clk_100M or negedge rst_n)
 begin
  if(!rst_n)
   begin
       cnt3 <= 5'd0;
    end
  else if((cnt_en)&&(cnt2 == t1ms))
    begin
            if(cnt3 == t_20ms )
               cnt3 <= t_20ms;
            else
        cnt3 <= cnt3 + 1;
    end
    else if(!cnt_en)
       cnt3 <= 5'd0;
  end

//----------------------------------
//按键状态机
    reg  [2:0]  i;
    reg      key_down;         //按键按下标志
    reg      key_up;           //按键释放标志
    always @(posedge clk_100M or negedge rst_n)
     begin
      if(!rst_n)
       begin
                key_down <= 0;
                key_up <= 0;
                i <= 0;
                cnt_en <= 0;
```

```
        end
      else
        begin
          case(i)
                'd0:
                  begin
                           key_down <= 0;
                     key_up <= 0;
                    if(HtoL_flag) i <= 'd1;          //检测到按下
                          else if(LtoH_flag) i <= 'd2;     //检测到释放按键
                          else  i  <= 'd0;
                   end
                  'd1:
                    begin
                          if(cnt3 == t_20ms )
                            begin
                               if(!key_in)                    //检测到按键依然被按下
                                 begin
                                     key_down <= 1;              //按键按下成功
                                     i <= 'd3;
                                     cnt_en <= 0;
                                   end
                                else
                                   begin
                                    key_down <= 0;
                                     i <= 'd0;
                                     cnt_en <= 0;
                                     end
                              end
                           else
                              cnt_en <= 1;
                          end
                  'd2:
                    begin
                          if(cnt3 == t_20ms )
                            begin
                                  if(key_in)                    //检测到按键被释放
                                 begin
                                     key_up <= 1;              //按键释放成功
                                     i <= 'd4;
                                     cnt_en <= 0;
                                   end
                                else
                                    begin
                                       key_up <= 0;
                                     i <= 'd0;
                                     cnt_en <= 0;
                                     end
                              end
                           else
                              cnt_en <= 1;
                          end
                  'd3:
                    begin
                                              if(key_in)
```

```
                                        begin
                            key_down <= 0;
                            i <= 'd0;
                                            end
                                        else
                                            i <= 'd3;
                        end
                                'd4:
                                    begin
                                        if(!key_in)
                                        begin
                            key_up <= 0;
                            i <= 'd0;
                                            end
                                        else
                                            i <= 'd4;
                                    end
                    default:i <= 'd0;
                endcase
        end
      end

 assign      key_out = key_down;          //当按键被按下有效时
// assign    key_out = key_up;          //当按键被释放后才有效时
endmodule
```

将两个程序合在一起的顶层文件：

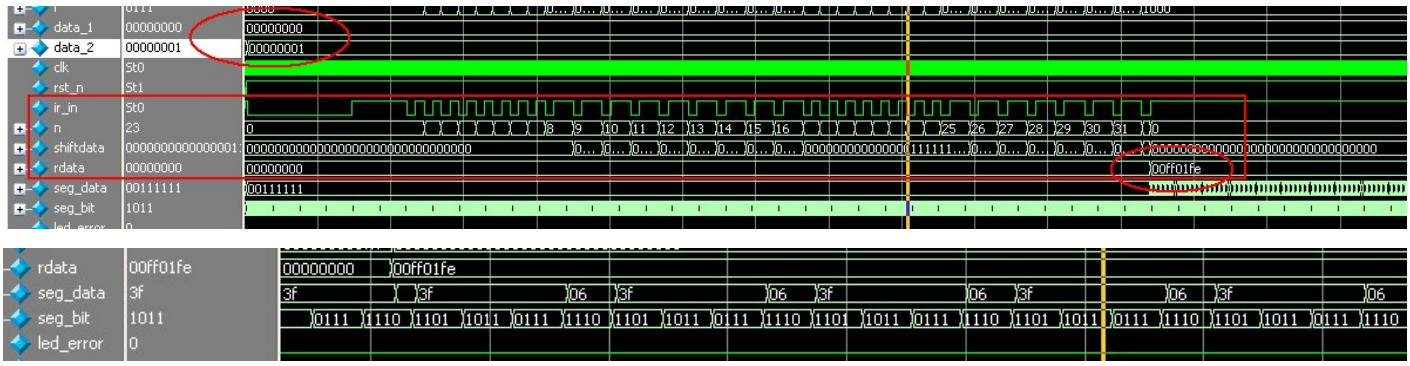
```
/*******************************Copyright***************************************
**----------------------------File information--------------------------
** File name  :IR_TOP.v
** CreateDate :2015.06
** Funtions   : 中断的顶层文件
** Operate on :M5C06N3F256C7
** Copyright  :All rights reserved.
** Version    :V1.0
**----------------------------Modify the file information----------------
** Modified by   :
** Modified data :
** Modify Content:
*******************************************************************************/

module  IR_TOP  (
            clk,
            rst_n,

                    ir_in,
                    ir_out,
```

```
                    key_1,
                    key_2,

                    led_d1,
                    led_d2,
                    led_d3,
                    led_d4,
                    led_d5,
                    led_d6,
                    led_d7,
                    led_d8,

                    test_data
            );
input           clk;
input           rst_n;

input           key_1;
input           key_2;

input           ir_in;
output          ir_out;

output          led_d1;
output          led_d2;
output          led_d3;
output          led_d4;
output          led_d5;
output          led_d6;
output          led_d7;
output          led_d8;

output  [7:0]   test_data;

//-----------------------------------------------//
 wire          key_1_flag;
   wire          key_2_flag;
 key_shake   U1(
       .clk_100M(clk),
       .rst_n(rst_n),

       .key_in(key_1),
       .key_out(key_1_flag)
        );

   key_shake  U2(
       .clk_100M(clk),
       .rst_n(rst_n),

       .key_in(key_2),
       .key_out(key_2_flag)
        );

 wire  [7:0]   data_t;

 IR_send  u1(
```

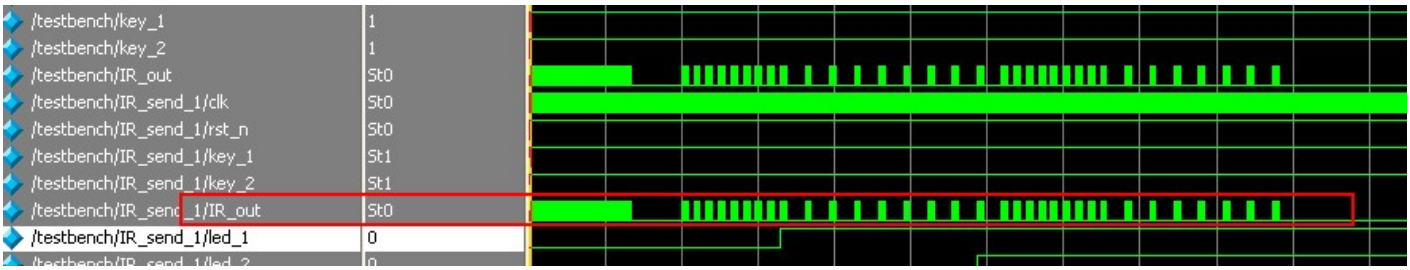
```
                .clk(clk),
                .rst_n(rst_n),

                        .key_1(key_1_flag),
                        .key_2(key_2_flag),

                        .IR_out(ir_out),
                        .led_1(led_d1),
                        .led_2(led_d2),
                        .led_3(led_d3),
                        .led_4(led_d4)
                  );

      IR_resive  u2(
                .clk(clk),
                .rst_n(rst_n),

                        .ir_in(ir_in),

                        .led_error(led_d8),

                        .led_5(led_d5),
                        .led_6(led_d6),
                        .led_7(led_d7),
                        .test_data(data_t)
                  );
     //---------------------------------------------//
     assign  test_data = data_t;
    endmodule
```

仿真图：

接收仿真图：




发送仿真图：


/testbench/key_1
/testbench/key_2
/testbench/IR_out
/testbench/IR_send_1/clk
/testbench/IR_send_1/rst_n
/testbench/IR_send_1/key_1
/testbench/IR_send_1/key_2
/testbench/IR_send_1/IR_out
/testbench/IR_send_1/led_1
1
1
St0
St0
St0
St1
St1
St0
0