

1.3 寸 oled 屏显示实验说明

1、实验简介

实验使用“小眼睛科技”公司的 FPGA 开发板以及 1.3 寸 oled 屏 PMOD 模块，使用 SPI 通信协议点亮 1.3 寸 OLED 屏。

2、实验目的

在 1.3 寸 OLED 屏幕上显示“深圳市小眼睛科技”8 个汉字。

3、实验使用模块介绍

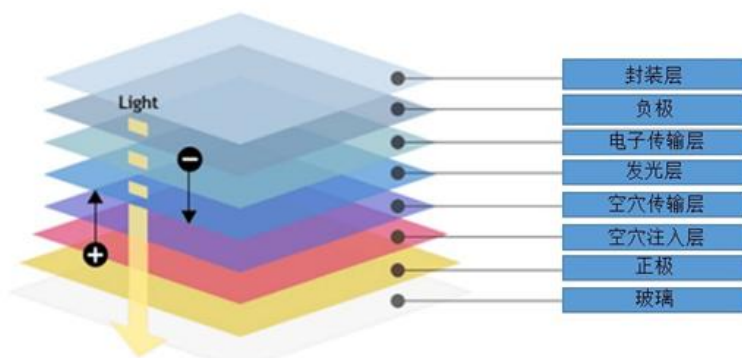
实验使用“小眼睛科技”的 PMOD-OLED 模块，OLED 尺寸为 1.3 寸，显示颜色为单色白色，使用 SPI 通信协议进行寄存器配置。

4、实验原理

(1) OLED 发光原理

OLED 全称 OrganicLight-Emitting Diode，即有机发光二极管，发光强度与注入的电流成正比，且不需要背光电源。

OLED 结构图如下图所示，自上到下由封装层、负极、电子传输层、发光层、空穴传输层、空穴注入层、正极、玻璃组成：



OLED 的发光位置在发光层，在电流的驱动下，电子通过电子传输层进入发光层，空穴通过空穴注入层进入发光层，二者在发光层发成复合反应，从而使发光层的发光材料发光。

(2) OLED 显示原理

实验中使用的“小眼睛科技”公司的 PMOD-OLED 模块，OLED 显示原理如下所述。

- 1) 在 OLED 模块使用前需要先对模块进行指令配置，相关指令请参考数据手册 *0.96 UN-2864KSWEG01.pdf*。

需注意的时 DEMO 中配置的屏幕刷新方向：左-->右；

- 2) 1.3 寸 PMOD-OLED 模块的像素点共 64×128 个像素点，其中 64 行像素点分为 8 页，每 8 行为 1 页。
- 3) 向 1.3 寸 PMOD-OLED 模块像素点寄存器写入像素点数据时，需要按页进行写操作。



- 4)

在进行页写操作时，需要通过指令传达页地址、起始列地址等信息。

页地址指令： $8'hb0 + PAGE$ ；（PAGE 为页地址）

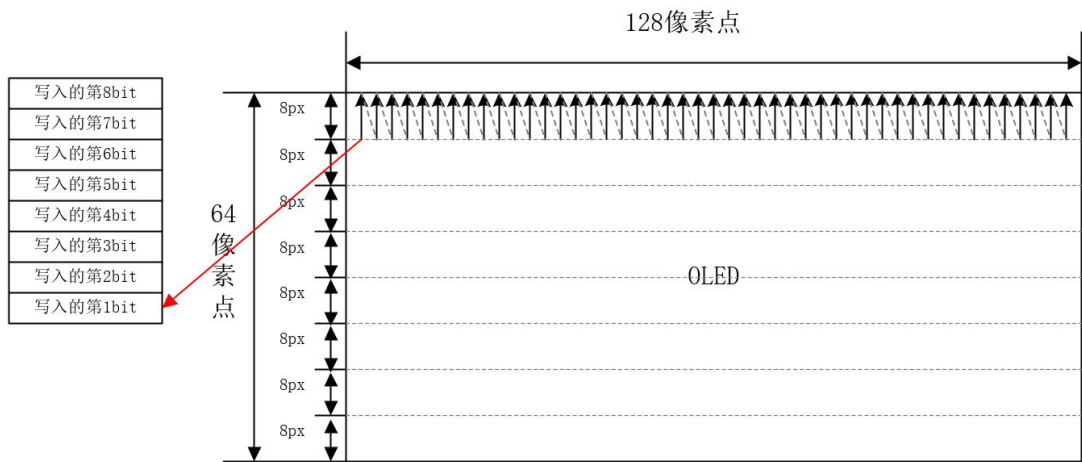
列地址指令： $8'h00 + column[3:0] + 8'd2$ ；（加 2 为列地址补偿）

$8'h10 + column[7:4]$ ；（column 为起始列地址）

(一般设置起始列地址为 00)

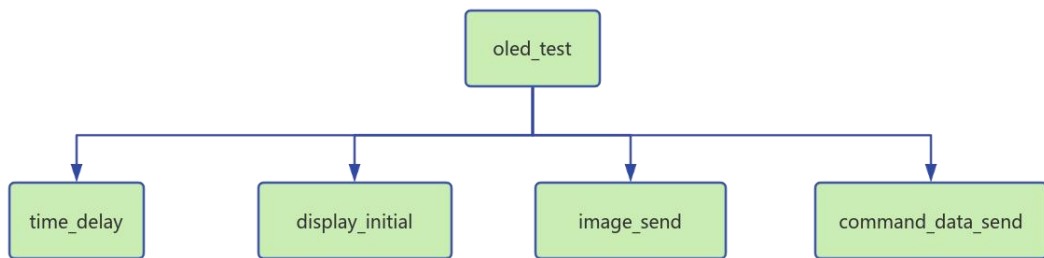
- 5) 向像素点寄存器写数据时，写入 1，则点亮此像素点，写入 0 则不点亮此像素点，由上述可知每页为 8 行 128 列像素点，配置屏幕刷新方向为从左到右，则写入像素点的数据与像素位置对应关系如下所述：

一页对应屏幕的一个长条形区域，以 8bit 数据为一组，则一组数据对应一列，8bit 数据中最先被写入的 1bit 数据刷新在长条形区域的下方，依次由下到上进行刷新，写完一列后再写下一列，列的刷新方向根据配置由左到右刷新。整体呈现 N 字形刷新走向。



5、实验代码设计

实验代码 module 框架如下图所示：



模块注释如下表所示：

module 名	注释
oled_test	顶层
time_delay	延时模块

display_initial	OLED 初始化指令配置模块
image_send	图像数据发送模块
command_data_send	SPI 驱动控制发送模块

顶层代码如下所示：

在点亮 OLED 屏前需要先对 OLED 屏进行指令配置，在此期间需要保持一段时间复位状态，在复位结束后仍需等待一段时间后才能进行指令配置；在配置完成后，按照像素刷新顺序对像素点寄存器进行刷新，从而实现点亮屏幕并在屏幕上显示期望的图像。注意使用 SPI 发送数据时，需要遵循手册的时序标准，具体内容请参考：*GM013001 (1.3 模块) SERIES 数据手册*。

```
module oled_test(
    input        sys_clk    /* synthesis PAP_MARK_DEBUG="true" */, //50MHz
    input        rst_n      ,
    output       rs         , //地址数据切换信号
    output       reset      ,
    output       scl        ,
    output       sda        /* synthesis PAP_MARK_DEBUG="true" */
);
wire cs ;

parameter time_delay1 = 32'd499_999 ;
parameter time_delay2 = 32'd499_999 ;

/////////////////////////////////////////////////////////////////
wire clk_5 ;                //5MHz
wire clk ;
wire rst_n ;
reg key_reg ;
reg rst_reg = 1'b0;
reg [7:0]cnt_clk ;

wire delay_end_0 /* synthesis PAP_MARK_DEBUG="true" */;
wire delay_end_1 /* synthesis PAP_MARK_DEBUG="true" */;
wire delay_start_0 /* synthesis PAP_MARK_DEBUG="true" */;
wire delay_start_1 /* synthesis PAP_MARK_DEBUG="true" */;
wire sda_start_init /* synthesis PAP_MARK_DEBUG="true" */;
wire [8:0]data_out_init /* synthesis PAP_MARK_DEBUG="true" */ ;
wire sda_end_init /* synthesis PAP_MARK_DEBUG="true" */;
wire initial_end /* synthesis PAP_MARK_DEBUG="true" */;
```

```

wire sda_end      /* synthesis PAP_MARK_DEBUG="true" */;
wire sda_start    /* synthesis PAP_MARK_DEBUG="true" */;
wire [8:0]sda_data /* synthesis PAP_MARK_DEBUG="true" */;
wire start        /* synthesis PAP_MARK_DEBUG="true" */;
wire send_end     /* synthesis PAP_MARK_DEBUG="true" */;
wire [8:0]data_send /* synthesis PAP_MARK_DEBUG="true" */;

////////////////////////////////CLK_GEN////////////////////////////////////////
PLL u_pll (
    .clk_in1(sys_clk),      // input
    .lock(),               // output
    .clkout0(clk)          // output
);

////////////////////////////////delay////////////////////////////////////////
time_delay time_delay_first(
    .clk          (clk          ) ,      //5MHz

    .rst_n        (rst_n        ) ,

    .delay_start  (delay_start_0 ) ,

    .delay_parameter (time_delay1 ) ,

    .delay_end    (delay_end_0   )
);

time_delay time_delay_second(
    .clk          (clk          ) ,      //5MHz

    .rst_n        (rst_n        ) ,

    .delay_start  (delay_start_1 ) ,

    .delay_parameter (time_delay2 ) ,

    .delay_end    (delay_end_1   )
);

////////////////////////////////////////////////////////////////////////

command_data_send command_data_send(
    .clk          (clk          ) ,

    .rst_n        (rst_n        ) ,

    .start        (start        ) ,//input

```

```

.rs_ctrl      (data_send[8] ) ,//input
.data_in      (data_send[7:0]) ,//input
.send_end     (send_end ) ,//input
.cs           (cs ) ,//output
.rs           (rs ) ,//output
.scl          (scl ) ,//output
.sda          (sda ) //output
);
////////////////////////////////////初始化////////////////////////////////////

display_initial display_initial(
    .clk      (clk ) ,
    .rst_n    (rst_n ) ,
    .delay_end_0 (delay_end_0 ) ,
    .delay_end_1 (delay_end_1 ) ,
    .sda_end    (sda_end_init ) ,
    .delay_start_0 (delay_start_0) ,
    .delay_start_1 (delay_start_1) ,
    .sda_start   (sda_start_init) ,
    .data_out    (data_out_init ) ,//[8:0]
    .reset      (reset ) ,
    .initial_end (initial_end )
);

////////////////////////////////////图像信息////////////////////////////////////

image_send image_send(
    .clk      (clk ) ,
    .rst_n    (rst_n ) ,
    .initial_end (initial_end) ,
    .sda_end    (sda_end ) ,
    .sda_start  (sda_start ) ,
    .sda_data   (sda_data ) // [8:0]
);

assign start      = (initial_end)? sda_start : sda_start_init ;
assign sda_end    = (initial_end)? send_end : 1'b0 ;
assign sda_end_init = (initial_end)? 1'b0 : send_end ;
assign data_send  = (initial_end)? sda_data : data_out_init ;

endmodule

```

延时模块

```
module time_delay (
    input      clk ,//5MHz
    input      rst_n ,
    input      delay_start ,
    input [31:0]delay_parameter ,
    output reg  delay_end
);
reg [31:0]cnt_delay ;
reg delay_flag ;

always @(posedge clk ) begin
    if (~rst_n)
        cnt_delay <= 32'd0 ;
    else if (cnt_delay >= delay_parameter)
        cnt_delay <= 32'd0 ;
    else if (delay_flag)
        cnt_delay <= cnt_delay + 32'b1 ;
    else
        cnt_delay <= 32'd0 ;
end

always @(posedge clk ) begin
    if (~rst_n)
        delay_flag <= 1'b0 ;
    else if (cnt_delay >= delay_parameter)
        delay_flag <= 1'b0 ;
    else if (delay_start == 1'b1)
        delay_flag <= 1'b1 ;
    else
        delay_flag <= delay_flag ;
end

always @(posedge clk ) begin
    if (~rst_n)
        delay_end <= 1'b0 ;
    else if (cnt_delay >= delay_parameter)
        delay_end <= 1'b1 ;
    else
        delay_end <= 1'b0 ;
end
endmodule
```

OLED 初始化指令发送模块:

```
module display_initial (
    input          clk          ,
    input          rst_n        ,
    input          delay_end_0   ,
    input          delay_end_1   ,
    input          sda_end       ,
    output reg      delay_start_0 ,
    output reg      delay_start_1 ,
    output reg      sda_start     ,
    output reg [8:0] data_out     ,
    output reg      reset        ,
    output reg      initial_end
);

reg [6:0] cnt_index /* synthesis PAP_MARK_DEBUG="true" */;
reg flag /* synthesis PAP_MARK_DEBUG="true" */;
reg [11:0] index /* synthesis PAP_MARK_DEBUG="true" */;

////////////////////////////////////

reg [8:0] state ;

parameter IDLE = 9'b000_000_001 ;
parameter state_reset = 9'b000_000_010 ;//send
delay_start_0
parameter state_reset_delay = 9'b000_000_100 ;//wait delay 255ms
parameter state_command = 9'b000_001_000 ;//send
delay_start_1
parameter state_command_delay = 9'b000_010_000 ;//wait delay 120ms
parameter state_index_begin = 9'b000_100_000 ;//send sda_start
and data_out
parameter state_index_wait = 9'b001_000_000 ;//wait for the data
to finish sending
parameter state_index_arbi = 9'b010_000_000 ;//check whather
the index has been send
parameter state_end = 9'b100_000_000 ;//send initial end

////////////////////////////////////

always @(posedge clk ) begin
    if (~rst_n) begin
        state <= IDLE ;
        cnt_index <= 7'b0 ;
    end
end
```



```

flag <= 1'b0 ;
end
else begin
    case (state)
        IDLE :                begin
                                state <= state_reset      ;
                            end
        state_reset :          begin
                                state <= state_reset_delay ;
                            end
        state_reset_delay :    begin
                                if ((delay_end_0) && (~flag))begin
                                    state <= state_reset ;
                                    flag <= ~flag ;
                                end
                                else if ((delay_end_0) && (flag))begin
                                    state <= state_command ;
                                    flag <= ~flag ;
                                end
                                else begin
                                    state <= state_reset_delay ;
                                    flag <= flag ;
                                end
                            end
        state_command :        begin
                                state <= state_index_begin ;
                            end
        state_command_delay :   begin
                                if ((delay_end_1) && (~flag))begin
                                    state <= state_command ;
                                    flag <= ~flag ;
                                end
                                else if ((delay_end_1) && (flag))begin
                                    state <= state_index_begin ;
                                    flag <= ~flag ;
                                end
                                else begin
                                    state <= state_command_delay ;
                                    flag <= flag ;
                                end
                            end
        state_index_begin :     begin
                                state <= state_index_wait ;
                            end
    end
end

```

```

        state_index_wait :      begin
            if (sda_end)
                state <= state_index_arbi ;
            else
                state <= state_index_wait ;
            end
        state_index_arbi :      begin
            if (cnt_index >= 7'd24) begin
                state <= state_end ;
                cnt_index <= 7'd0 ;
            end
            else if (cnt_index <= 7'd1) begin
                state <= state_command_delay ;
                cnt_index <= cnt_index + 7'd1 ;
            end
            else begin
                state <= state_index_begin ;
                cnt_index <= cnt_index + 7'd1 ;
            end
        end
        state_end :              begin
            state <= state_end ;
        end
        default: begin state <= IDLE ;
                       cnt_index <= 7'b0 ;
                       flag <= flag ;
        end
    endcase
end

////////////////////////////////////

always @(posedge clk ) begin
    if (~rst_n)
        delay_start_0 <= 1'b0 ;
    else if (state == state_reset)
        delay_start_0 <= 1'b1 ;
    else
        delay_start_0 <= 1'b0 ;
end

always @(posedge clk ) begin
    if (~rst_n)

```

```

        delay_start_1 <= 1'b0 ;
    else if ((state == state_index_arbi)&&(cnt_index <= 7'd1))
        delay_start_1 <= 1'b1 ;
    else
        delay_start_1 <= 1'b0 ;
end

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

always @(posedge clk ) begin
    if (~rst_n)
        reset <= 1'b0 ;
    else if (state == state_reset)
        reset <= flag ;
    else
        reset <= reset ;
end

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

always @(posedge clk ) begin
    if (~rst_n)
        sda_start <= 1'b0 ;
    else if (state == state_index_begin)
        sda_start <= 1'b1 ;
    else
        sda_start <= 1'b0 ;
end

always @(posedge clk ) begin
    if (~rst_n)
        data_out <= 9'd0 ;
    else if (state == state_index_begin)
        data_out <= index[8:0];
    else
        data_out <= data_out ;
end

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

always @(posedge clk ) begin
    if (~rst_n)
        initial_end <= 1'b0 ;
    else if (state == state_end)
        initial_end <= 1'b1 ;

```

```

    else
        initial_end <= 1'b0 ;
end

////////////////////////////////////////////////////////////////

always @(*) begin
    case (cnt_index)
        0    :index <= 12'h0_AE ;
        1    :index <= 12'h0_D5 ;
        2    :index <= 12'h0_80 ;
        3    :index <= 12'h0_A8 ;
        4    :index <= 12'h0_3F ;
        5    :index <= 12'h0_D3 ;
        6    :index <= 12'h0_00 ;
        7    :index <= 12'h0_40 ;
        8    :index <= 12'h0_8D ;
        9    :index <= 12'h0_14 ;
        10   :index <= 12'h0_20 ;
        11   :index <= 12'h0_02 ;
        12   :index <= 12'h0_A1 ;
        13   :index <= 12'h0_C8 ;
        14   :index <= 12'h0_DA ;
        15   :index <= 12'h0_12 ;
        16   :index <= 12'h0_81 ;
        17   :index <= 12'h0_66 ;
        18   :index <= 12'h0_D9 ;
        19   :index <= 12'h0_F1 ;
        20   :index <= 12'h0_DB ;
        21   :index <= 12'h0_30 ;
        22   :index <= 12'h0_A4 ;
        23   :index <= 12'h0_A6 ;
        24   :index <= 12'h0_AF ;
        default: index <= 12'h0_af ;
    endcase
end

endmodule

```

SPI 数据发送模块

```
module command_data_send (
    input        clk        ,
    input        rst_n      ,
    input        start      ,
    input        rs_ctrl    ,
    input  [7:0]  data_in    ,
    output       send_end   ,
    output       cs         ,
    output       rs         ,
    output       scl        ,
    output       sda
);

reg [7:0] sda_data /* synthesis PAP_MARK_DEBUG="true" */;
reg rs_reg /* synthesis PAP_MARK_DEBUG="true" */;
reg [3:0] cnt /* synthesis PAP_MARK_DEBUG="true" */;
reg [4:0] state /* synthesis PAP_MARK_DEBUG="true" */;
wire sda_data_in /* synthesis PAP_MARK_DEBUG="true" */;
wire sda_data_out /* synthesis PAP_MARK_DEBUG="true" */;

parameter IDLE      = 5'b00001 ;
parameter CS_LOW    = 5'b00010 ;
parameter SDA_SEND  = 5'b00100 ;
parameter CS_HIGH   = 5'b01000 ;
parameter END_STATE = 5'b10000 ;

/////////////////////////////////进行锁存/////////////////////////////////

always @(posedge clk ) begin
    if (~rst_n)
        sda_data <= 8'd0 ;
    else if (start)
        sda_data <= data_in[7:0] ;
    else
        sda_data <= sda_data ;
end

always @(posedge clk ) begin
    if (~rst_n)
        rs_reg <= 1'b1 ;
    else if (start)
        rs_reg <= rs_ctrl ;
    else
```

```

        rs_reg <= rs_reg ;
end

////////////////////////////////////状态机////////////////////////////////////

always @(posedge clk ) begin
    if (~rst_n) begin
        state <= IDLE ;
        cnt <= 4'd0 ;
    end
    else begin
        case (state)
            IDLE      :begin
                if (start)
                    state <= CS_LOW      ;
                else
                    state <= IDLE      ;
            end
            CS_LOW    :begin
                state <= SDA_SEND      ;
            end
            SDA_SEND:begin
                if (cnt >= 4'd7) begin
                    state <= CS_HIGH      ;
                    cnt <= 4'd0 ;
                end
                else begin
                    state <= SDA_SEND      ;
                    cnt <= cnt + 1'b1      ;
                end
            end
            CS_HIGH   :begin
                state <= END_STATE      ;
            end
            END_STATE:begin
                state <= IDLE      ;
            end
            default:begin
                state <= IDLE      ;
                cnt <= 4'd0      ;
            end
        endcase
    end
end
end

```

```
////////////////////////////////////输出赋值////////////////////////////////////  
assign rs = rs_reg ;  
assign cs = ((state == CS_LOW) || (state == SDA_SEND) || (state == CS_HIGH))?  
1'b0 : 1'b1 ;  
  
assign scl = (state == SDA_SEND)? ~clk : 1'b1 ;  
assign sda_data_out = (state == SDA_SEND)? sda_data[7-cnt] : 1'b1 ;  
  
assign sda = (state == SDA_SEND)? sda_data_out : 1'b0 ;  
assign sda_data_in = sda ;  
  
assign send_end = (state == END_STATE)? 1'b1 : 1'b0 ;  
endmodule
```

图像数据发送模块

```
module image_send (
    input          clk          ,
    input          rst_n        ,
    input          initial_end  ,
    input          sda_end      ,
    output         sda_start    ,
    output [8:0]    sda_data
);

reg [9:0]addr /* synthesis PAP_MARK_DEBUG="true" */;
reg [1:0]cnt /* synthesis PAP_MARK_DEBUG="true" */; //
0:column 1:page 2:data
reg [7:0]cnt_c /* synthesis PAP_MARK_DEBUG="true" */;
reg [2:0]cnt_p /* synthesis PAP_MARK_DEBUG="true" */;
reg [8:0]data_sig /* synthesis PAP_MARK_DEBUG="true" */;
reg start_reg /* synthesis PAP_MARK_DEBUG="true" */;
reg data_start /* synthesis PAP_MARK_DEBUG="true" */;
wire [7:0]rd_data /* synthesis PAP_MARK_DEBUG="true" */;
wire [7:0]order[0:2]/* synthesis PAP_MARK_DEBUG="true" */;
//assign order[0] = 8'h22 ;
//assign order[1] = 8'hb0 + cnt_p;
//assign order[2] = 8'h07 ;
assign order[0] = 8'hb0 + cnt_p;
assign order[1] = 8'h10 + ((cnt_c + 8'd2)>>4) ;
assign order[2] = 8'h00 + ((cnt_c + 8'd2)&8'h0f);
reg [6:0]state /* synthesis PAP_MARK_DEBUG="true" */;

parameter idle          = 7'b0_000_001;
parameter state_command = 7'b0_000_010; //send instruction
parameter state_command_wait = 7'b0_000_100; //wait for sending to
complete
parameter state_send    = 7'b0_001_000; //send addr
parameter state_send_wait = 7'b0_010_000;
parameter state_wait    = 7'b0_100_000; //send data
parameter state_end      = 7'b1_000_000;

////////////////////////////////////

always @(posedge clk ) begin
    if (~rst_n)
        state <= idle ;
    else begin
```



```

    case (state)
        idle :begin
            if (initial_end)
                state <= state_command ;
            else
                state <= idle ;
            end
        state_command :begin
            state <= state_command_wait ;
        end
        state_command_wait :begin
            if ((cnt >= 2'd3)&&(sda_end))
                state <= state_send ;
            else if (sda_end)
                state <= state_command ;
            else
                state <= state_command_wait ;
            end
        state_send :begin
            state <= state_send_wait ;
        end
        state_send_wait :begin
            if ((cnt_c >= 8'd128)&&(sda_end))
                state <= state_wait ;
            else if (sda_end)
                state <= state_send ;
            else
                state <= state_send_wait ;
            end
        state_wait :begin
            if (cnt_p >= 3'd7)
                state <= state_end ;
            else
                state <= state_command ;
            end
        state_end : begin
            state <= state_end ;
        end
        default:state <= idle ;
    endcase
end
end

```

////////////////////////////////////

```

always @(posedge clk ) begin
    if (~rst_n) begin
        cnt_p <= 3'd0 ;
        cnt_c <= 8'd0 ;
    end
    else if (state == state_send)begin
        cnt_p <= cnt_p ;
        cnt_c <= cnt_c + 8'd1 ;
    end
    else if (state == state_wait)begin
        cnt_p <= cnt_p + 3'd1 ;
        cnt_c <= 8'd0 ;
    end
    else if (state == state_end)begin
        cnt_p <= 3'd0 ;
        cnt_c <= 8'd0 ;
    end
end

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
always @(posedge clk ) begin
    if (~rst_n)
        cnt <= 2'd0 ;
    else if (state ==state_command)
        cnt <= cnt + 2'd1 ;
    else if (state == state_send)
        cnt <= 2'd0 ;
end

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
always @(posedge clk ) begin
    if (~rst_n)
        data_sig <= 9'd0 ;
    else if (state == state_command)
        data_sig <= {1'b0 ,order[cnt] };
    else if (state == state_send)
        data_sig <= {1'b1 ,rd_data};
    else
        data_sig <= data_sig ;
end

always @(posedge clk ) begin
    if (~rst_n)
        start_reg <= 1'b0 ;
    else if ((state == state_command)|| (state == state_send))
        start_reg <= 1'b1 ;

```

```

        else
            start_reg <= 1'b0 ;
        end

assign sda_data = data_sig ;
assign sda_start = start_reg ;

always @(posedge clk ) begin
    if (~rst_n)
        addr <= 10'd0 ;
    else if (state == state_send)
        addr <= addr + 10'd1 ;
    else if (state == state_end)
        addr <= 10'd0 ;
    end

rom_image rom_image_display (
    .addr(addr),           // input [9:0]
    .clk(clk),            // input
    .rst(~rst_n),         // input
    .rd_data(rd_data)     // output [7:0]
);

endmodule

```

6、实验现象

MES_OLED 模块连接至 PGL50H 开发板的 PMOD 接口，OLED 屏幕上显示“深圳市小眼睛科技”的字样。