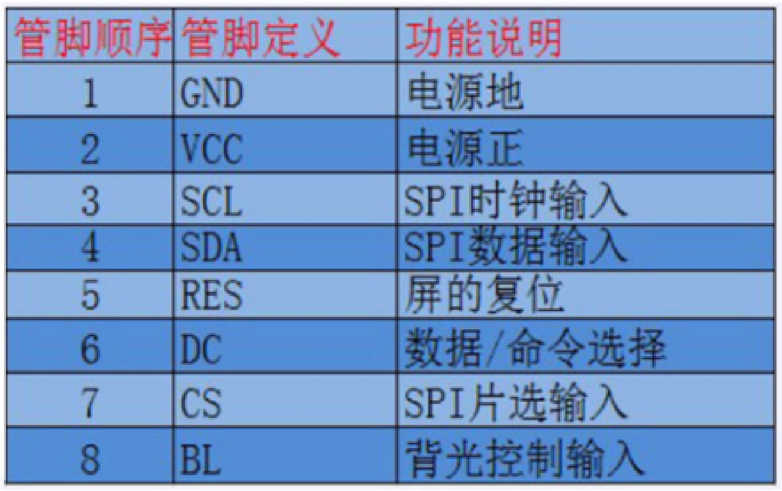
SPI RGB LCD display driver based on STEP FPGA

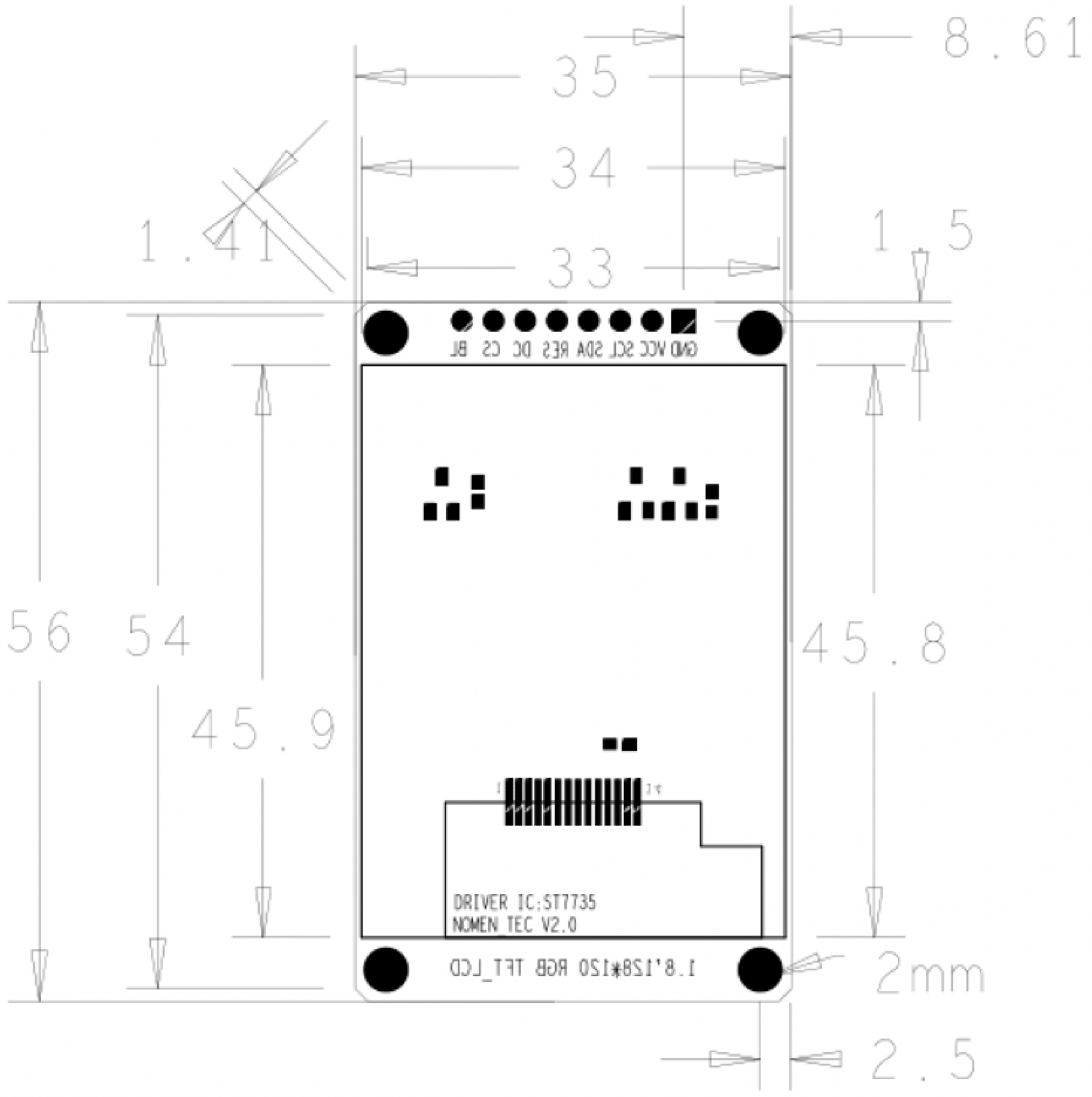
In this section, we will use the 1.8-inch RGB LCD screen on the FPGA driver backplane to realize the image display function.

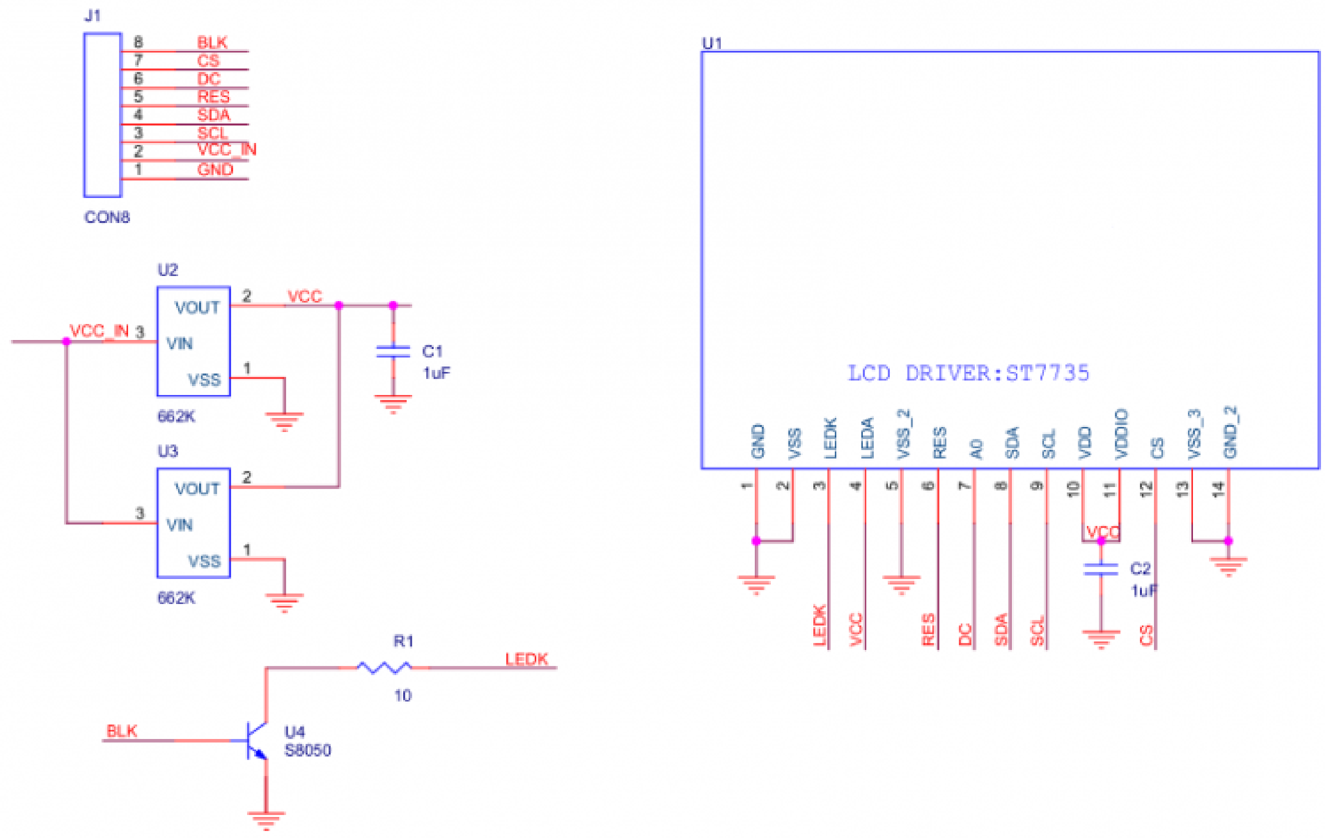
====Hardware description====

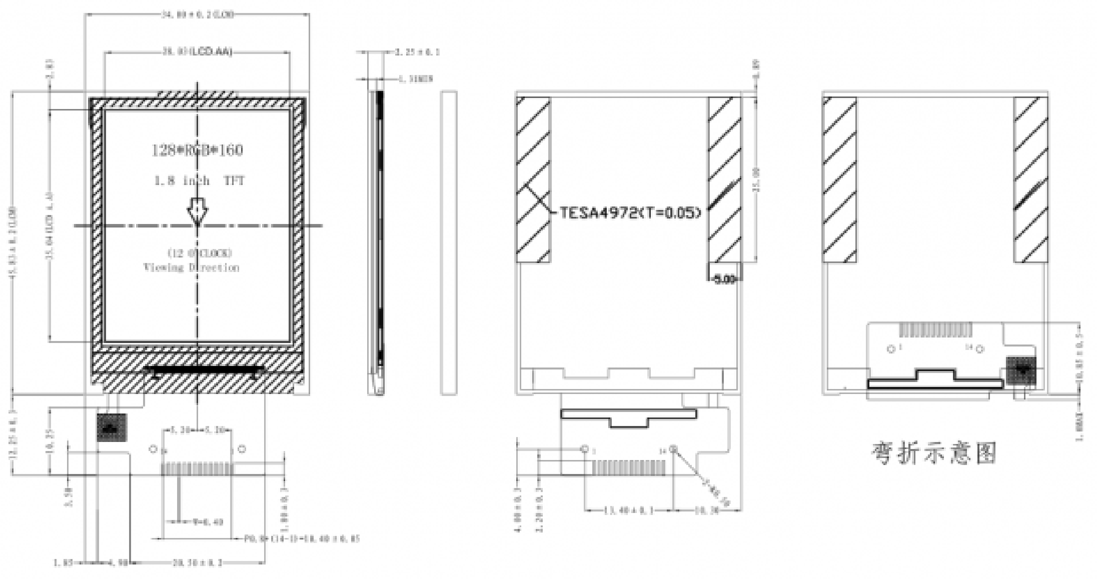
A 1.8-inch color LCD TFT *LCD module* is integrated on our STEP-BaseBoard backplane. *You can drive the LCD to display text, pictures or dynamic waveforms.*
First understand the LCD module, download the relevant information: https://pan.baidu.com/s/1bp6AYsR The (https://pan.baidu.com/s/1bp6AYsR) block diagram is as follows: the schematic diagram is as follows: the device U1 in the schematic diagram is an LCD screen, the LCD screen is 1.8 inches, 128 RGB 160 Pixels, serial bus (SPI), the LCD screen integrates the ST7735S driver, the processor communicates with the ST7735S to complete the display control of the LCD screen. ST7735S is a 132RGB x 162 pixel dot 262K controller/driver. The chip can be directly connected to an external processor. Support serial SPI communication and 8/9/16/18-bit parallel communication (the LCD screen integrates ST7735S without a parallel interface, so you can only use serial communication). For detailed parameters, please refer to the data sheet: st7735s_datasheet.pdf to be continued = ===Verilog code==== —— <code verilog>
—————————————————————— »»»»»»»»»»»»» > COPYRIGHT NOTICE «««««««««««<
————————————————————— Module: LCD

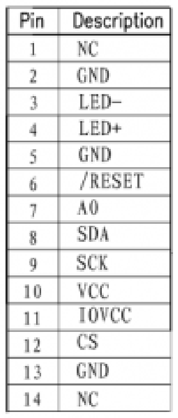


管脚顺序	管脚定义	功能说明
1	GND	电源地
2	VCC	电源正
3	SCL	SPI时钟输入
4	SDA	SPI数据输入
5	RES	屏的复位
6	DC	数据/命令选择
7	CS	SPI片选输入
8	BL	背光控制输入


8.61
35
34
33
1.41
1.5
BL CS DC RES SDA SCL VCC GND
56
54
45.9
45.8
DRIVER IC:ST7735
NOMEN_TEC V2.0
1.8'128*120 RGB TFT_LCD
2mm
2.5







Pin	Description
1	NC
2	GND
3	LED−
4	LED+
5	GND
6	/RESET
7	A0
8	SDA
9	SCK
10	VCC
11	IOVCC
12	CS
13	GND
14	NC

RGB Author: Step Description: Drive TFT *RGB*LCD *1.8 to display Web: www.stepfpga.com (http://www.stepfpga.com)* ———————————————————— *Code Revision History:* ——————————— ——————————— *Version: |Mod. Date: |Changes Made: V1.1 |2016/10/30 |Initial ver* —————————————————— —————— *module LCD RGB # (parameter LCD W = 8'd132, LCD pixel width parameter LCD* H = 8'd162 *LCD pixel height) (input clk in, 12MHz system clock input rst n in, system Reset, low effective output reg ram lcd clk en,* RAM clock enable

```
output reg [7:0] ram_lcd_addr, //RAM address signal
input [131:0] ram_lcd_data, //RAM data signal

output reg lcd_rst_n_out, //LCD screen reset
output reg lcd_bl_out, //LCD backlight control
output reg lcd_dc_out, //LCD data command control
output reg lcd_clk_out, //LCD clock signal
output reg lcd_data_out //LCD data signal
```

);

```
localparam INIT_DEPTH = 16'd73; //Number of LCD initialization commands and data

localparam RED = 16'hf800; //Red
localparam GREEN = 16'h07e0; //green
localparam BLUE = 16'h001f; //blue
localparam BLACK = 16'h0000; //black
localparam WHITE = 16'hffff; //white
localparam                      YELLOW  =       16'hffe0;       //黄色

localparam                      IDLE    =       3'd0;
localparam                      MAIN    =       3'd1;
localparam                      INIT    =       3'd2;
localparam                      SCAN    =       3'd3;
localparam                      WRITE   =       3'd4;
localparam                      DELAY   =       3'd5;

localparam                      LOW             =       1'b0;
localparam                      HIGH    =       1'b1;

//assign        lcd_bl_out = HIGH;                              // backlight active high level

wire [15:0] color_t = YELLOW; //The top color is yellow
wire [15:0] color_b = BLACK; //The background color is black

reg                     [7:0]   x_cnt;
reg [7: 0] y_cnt;
reg                     [131:0] ram_data_r;

reg                     [8:0]   data_reg;                               //
reg                     [8:0]   reg_setxy       [10:0];
reg                     [8:0]   reg_init        [72:0];
reg                     [2:0]   cnt_main;
reg                     [2:0]   cnt_init;
reg                     [2:0]   cnt_scan;
reg                     [5:0]   cnt_write;
reg                     [15:0]  cnt_delay;
reg                     [15:0]  num_delay;
reg                     [15:0]  cnt;
reg                             high_word;
reg                     [2:0]   state = IDLE;
reg                     [2:0]   state_back = IDLE;
always@(posedge clk_in or negedge rst_n_in) begin
        if(!rst_n_in) begin
                x_cnt <= 8'd0;
                y_cnt <= 8'd0;
                ram_lcd_clk_en <= 1'b0;
                ram_lcd_addr <= 8'd0;
                cnt_main <= 3'd0;
                cnt_init <= 3'd0;
                cnt_scan <= 3'd0;
                cnt_write <= 6'd0;
                cnt_delay <= 16'd0;
                num_delay <= 16'd50;
                cnt <= 16'd0;
                high_word <= 1'b1;
                lcd_bl_out <= LOW;
                state <= IDLE;
                state_back <= IDLE;
        end else begin
                case(state)
                        IDLE:begin
                                        x_cnt <= 8'd0;
                                        y_cnt <= 8'd0;
                                        ram_lcd_clk_en <= 1'b0;
                                        ram_lcd_addr <= 8'd0;
                                        cnt_main <= 3'd0;
                                        cnt_init <= 3'd0;
                                        cnt_scan <= 3'd0;
                                        cnt_write <= 6'd0;
                                        cnt_delay <= 16'd0;
                                        num_delay <= 16'd50;
                                        cnt <= 16'd0;
                                        high_word <= 1'b1;
                                        state <= MAIN;
                                        state_back <= MAIN;
                                end
                        MAIN:begin
                                        case(cnt_main) //MAIN state
                                                3'd0:   begin state <= INIT; cnt_main <= cnt_m
ain + 1'b1; end
                                                3'd1:   begin state <= SCAN; cnt_main <= cnt_m
ain + 1'b1; end
                                                3'd2:   begin cnt_main <= 1'b1; end
                                                default: state <= IDLE;
                                        endcase
                                end
                        INIT:begin //initialization state
                                        case(cnt_init)
                                                3'd0: begin lcd_rst_n_out <= 1'b0; cnt_init <=
cnt_init + 1'b1; end // reset is valid
                                                3'd1:   begin num_delay <= 16'd3000; state <=
 DELAY; state_back <= INIT; cnt_init <= cnt_init + 1'b1; end    //延时
                                                3'd2: begin lcd_rst_n_out <= 1'b1; cnt_init <=
cnt_init + 1'b1; end //Reset and restore
                                                3'd3:   begin num_delay <= 16'd3000; state <=
 DELAY; state_back <= INIT; cnt_init <= cnt_init + 1'b1; end    //延时
                                                3'd4: start
                                                                        if(cnt>=INIT_DEPTH) be
gin //When 73 commands and data are issued, the configuration is complete
                                                                                cnt <= 16'd0;
                                                                                cnt_init <= cn
t_init + 1'b1;
                                                                        end else begin
                                                                                data_reg <= re
g_init[cnt];
```

```
                                                                                if(cnt==16'd0)
num_delay <= 16'd50000; //The first instruction requires a longer delay
                                                                                else num_delay
<= 16'd50;
                                                                                cnt <= cnt + 1
6'd1;
                                                                                state <= WRIT
E;
                                                                                state_back <=
 INIT;
                                                                        end
                                                                end
                                                3'd5: begin cnt_init <= 1'b0; state <= MAIN; e
nd //initialization is complete, return to MAIN state
                                                default: state <= IDLE;
                                        endcase
                                end
                        SCAN:begin //Screen refresh status, read data from RAM to refresh the
 screen
                                        case(cnt_scan)
                                                3'd0: begin //Determine the coordinates of the
area for refreshing the screen, here is the full screen
                                                                        if(cnt >= 11) begin
//
                                                                                cnt <= 16'd0;
                                                                                cnt_scan <= cn
t_scan + 1'b1;
                                                                        end else begin
                                                                                data_reg <= re
g_setxy[cnt];
                                                                                cnt <= cnt + 1
6'd1;
                                                                                num_delay <= 1
6'd50;
                                                                                state <= WRIT
E;
                                                                                state_back <=
 SCAN;
                                                                        end
                                                                end
                                                3'd1: begin ram_lcd_clk_en <= HIGH; ram_lcd_ad
dr <= y_cnt; cnt_scan <= cnt_scan + 1'b1; end //RAM clock enable
                                                3'd2: begin cnt_scan <= cnt_scan + 1'b1; end
 // delay one clock
                                                3'd3: begin ram_lcd_clk_en <= LOW; ram_data_r
 <= ram_lcd_data; cnt_scan <= cnt_scan + 1'b1; end //Read RAM data and turn off RAM clock enab
le
                                                3'd4: begin //Each pixel needs 16 bits of dat
a, SPI transmits 8 bits each time, and transmits high 8 bits and low 8 bits twice respectively
                                                                        if(x_cnt>=LCD_W) begin
//When a data (one line of screen) is written,
                                                                                x_cnt <= 8'd0;
                                                                                if(y_cnt>=LCD_
H) begin y_cnt <= 8'd0; cnt_scan <= cnt_scan + 1'b1; end //If it is the last line, jump out of
the loop
                                                                                else begin y_c
nt <= y_cnt + 1'b1; cnt_scan <= 3'd1; end //Otherwise jump to RAM clock enable, and refresh th
e screen
                                                                        end else begin
                                                                                if(high_word)
 data_reg <= {1'b1,(ram_data_r[x_cnt]? color_t[15:8]:color_b[15:8])}; //According to the statu
s of the corresponding bit, the top color or background color is displayed The state of high_w
ord is judged to write high 8 bits or low 8 bits
                                                                                else begin dat
a_reg <= {1'b1,(ram_data_r[x_cnt]? color_t[7:0]:color_b[7:0])}; x_cnt <= x_cnt + 1'b1; end //
 Determined according to the status of the corresponding bit Display top color or background c
olor, write high 8 bits or low 8 bits according to the state of high_word, and point to the ne
xt bit at the same time
                                                                                high_word <= ~
high_word; //high_word status flip
                                                                                num_delay <= 1
6'd50; //Set the delay time
                                                                                state <= WRIT
E; //Jump to WRITE state
                                                                                state_back <=
 SCAN; //Return to SCAN state after executing WRITE and DELAY operations
                                                                        end
                                                                end
                                                3'd5:   begin cnt_scan <= 1'b0; lcd_bl_out <=
 HIGH; state <= MAIN; end
                                                default: state <= IDLE;
                                        endcase
                                end
                        WRITE:begin //WRITE state, send data to the screen according to SPI ti
ming
                                        if(cnt_write >= 6'd17) cnt_write <= 1'b0;
                                        else cnt_write <= cnt_write + 1'b1;
                                        case(cnt_write)
                                                6'd0: begin lcd_dc_out <= data_reg[8]; end //T
he highest bit of 9-bit data is the command data control bit
                                                6'd1: begin lcd_clk_out <= LOW; lcd_data_out <
= data_reg[7]; end //send high data first
                                                6'd2:   begin lcd_clk_out <= HIGH; end
                                                6'd3:   begin lcd_clk_out <= LOW; lcd_data_out
<= data_reg[6]; end
                                                6'd4:   begin lcd_clk_out <= HIGH; end
                                                6'd5:   begin lcd_clk_out <= LOW; lcd_data_out
<= data_reg[5]; end
                                                6'd6:   begin lcd_clk_out <= HIGH; end
                                                6'd7:   begin lcd_clk_out <= LOW; lcd_data_out
<= data_reg[4]; end
                                                6'd8:   begin lcd_clk_out <= HIGH; end
                                                6'd9:   begin lcd_clk_out <= LOW; lcd_data_out
<= data_reg[3]; end
                                                6'd10:  begin lcd_clk_out <= HIGH; end
                                                6'd11:  begin lcd_clk_out <= LOW; lcd_data_out
```

```
<= data_reg[2]; end
                                                6'd12:  begin lcd_clk_out <= HIGH; end
                                                6'd13:  begin lcd_clk_out <= LOW; lcd_data_out
<= data_reg[1]; end
                                                6'd14:  begin lcd_clk_out <= HIGH; end
                                                6'd15: begin lcd_clk_out <= LOW; lcd_data_out
 <= data_reg[0]; end //Send low data later
                                                6'd16:  begin lcd_clk_out <= HIGH; end
                                                6'd17:  begin lcd_clk_out <= LOW; state <= DEL
AY; end //
                                                default: state <= IDLE;
                                        endcase
                                end
                        DELAY:begin //Delay state
                                        if(cnt_delay >= num_delay) begin
                                                cnt_delay <= 16'd0;
                                                state <= state_back;
                                        end else cnt_delay <= cnt_delay + 1'b1;
                                end
                        default:state <= IDLE;
                endcase
        end
end

// data for setxy
initial //Set the display area command and data
        begin
                reg_setxy[0]    =       {1'b0,8'h2a};
                reg_setxy[1]    =       {1'b1,8'h00};
                reg_setxy[2]    =       {1'b1,8'h00};
                reg_setxy[3]    =       {1'b1,8'h00};
                reg_setxy[4]    =       {1'b1,LCD_W-1};
                reg_setxy[5]    =       {1'b0,8'h2b};
                reg_setxy[6]    =       {1'b1,8'h00};
                reg_setxy[7]    =       {1'b1,8'h00};
                reg_setxy[8]    =       {1'b1,8'h00};
                reg_setxy[9]    =       {1'b1,LCD_H-1};
                reg_setxy[10]   =       {1'b0,8'h2c};
        end

// data for init
initial //LCD initialization command and data
        begin
                reg_init[0]             =       {1'b0,8'h11};
                reg_init[1]             =       {1'b0,8'hb1};
                reg_init[2]             =       {1'b1,8'h05};
                reg_init[3]             =       {1'b1,8'h3c};
                reg_init[4]             =       {1'b1,8'h3c};
                reg_init[5]             =       {1'b0,8'hb2};
                reg_init[6]             =       {1'b1,8'h05};
                reg_init[7]             =       {1'b1,8'h3c};
                reg_init[8]             =       {1'b1,8'h3c};
                reg_init[9]             =       {1'b0,8'hb3};
                reg_init[10]    =       {1'b1,8'h05};
                reg_init[11]    =       {1'b1,8'h3c};
                reg_init[12]    =       {1'b1,8'h3c};
                reg_init[13]    =       {1'b1,8'h05};
                reg_init[14]    =       {1'b1,8'h3c};
                reg_init[15]    =       {1'b1,8'h3c};
                reg_init[16]    =       {1'b0,8'hb4};
                reg_init[17]    =       {1'b1,8'h03};
                reg_init[18]    =       {1'b0,8'hc0};
                reg_init[19]    =       {1'b1,8'h28};
                reg_init[20]    =       {1'b1,8'h08};
                reg_init[21]    =       {1'b1,8'h04};
                reg_init[22]    =       {1'b0,8'hc1};
                reg_init[23]    =       {1'b1,8'hc0};
                reg_init[24]    =       {1'b0,8'hc2};
                reg_init[25]    =       {1'b1,8'h0d};
                reg_init[26]    =       {1'b1,8'h00};
                reg_init[27]    =       {1'b0,8'hc3};
                reg_init[28]    =       {1'b1,8'h8d};
                reg_init[29]    =       {1'b1,8'h2a};
                reg_init[30]    =       {1'b0,8'hc4};
                reg_init[31]    =       {1'b1,8'h8d};
                reg_init[32]    =       {1'b1,8'hee};
                reg_init[32]    =       {1'b0,8'hc5};
                reg_init[33]    =       {1'b1,8'h1a};
                reg_init[34]    =       {1'b0,8'h36};
                reg_init[35]    =       {1'b1,8'hc0};
                reg_init[36]    =       {1'b0,8'he0};
                reg_init[37]    =       {1'b1,8'h04};
                reg_init[38]    =       {1'b1,8'h22};
                reg_init[39]    =       {1'b1,8'h07};
                reg_init[40]    =       {1'b1,8'h0a};
                reg_init[41]    =       {1'b1,8'h2e};
                reg_init[42]    =       {1'b1,8'h30};
                reg_init[43]    =       {1'b1,8'h25};
                reg_init[44]    =       {1'b1,8'h2a};
                reg_init[45]    =       {1'b1,8'h28};
                reg_init[46]    =       {1'b1,8'h26};
                reg_init[47]    =       {1'b1,8'h2e};
                reg_init[48]    =       {1'b1,8'h3a};
                reg_init[49]    =       {1'b1,8'h00};
                reg_init[50]    =       {1'b1,8'h01};
                reg_init[51]    =       {1'b1,8'h03};
                reg_init[52]    =       {1'b1,8'h13};
                reg_init[53]    =       {1'b0,8'he1};
                reg_init[54]    =       {1'b1,8'h04};
                reg_init[55]    =       {1'b1,8'h16};
                reg_init[56]    =       {1'b1,8'h06};
                reg_init[57]    =       {1'b1,8'h0d};
                reg_init[58]    =       {1'b1,8'h2d};
                reg_init[59]    =       {1'b1,8'h26};
                reg_init[60]    =       {1'b1,8'h23};
                reg_init[61]    =       {1'b1,8'h27};
                reg_init[62]    =       {1'b1,8'h27};
                reg_init[63]    =       {1'b1,8'h25};
                reg_init[64]    =       {1'b1,8'h2d};
                reg_init[65]    =       {1'b1,8'h3b};
                reg_init[66]    =       {1'b1,8'h00};
                reg_init[67]    =       {1'b1,8'h01};
                reg_init[68]    =       {1'b1,8'h04};
                reg_init[69]    =       {1'b1,8'h13};
                reg_init[70]    =       {1'b0,8'h3a};
                reg_init[71]    =       {1'b1,8'h05};
                reg_init[72]    =       {1'b0,8'h29};

        end
```

endmodule </code>

====Summary====

This section mainly explains the picture display framework of the 1.8-inch RGB LCD screen. You need to master it while creating your own project, and generate the FPGA configuration file loading test through the entire design process.
If you are not familiar with the use of Diamond software, please refer to here: Use of Diamond .

====Related Information====

Use STEP-MXO2 second generation of RGB 1.8 inch LCD display driver: download link will be updated, the subsequent use of STEP-MAX10 1.8 inch LCD display driver RGB: subsequent download link will be updated