同济大学计算机系

数字逻辑课程综合实验报告

学　　号 _____

姓　　名 _____

专　　业 _____

授课老师 _____

一、实验内容

1.1 项目简述

本项目是基于 FPGA 开发板、OV2640 摄像头、VGA 以及 HC-06 蓝牙模块，使用 verilog 语言开发的多个印刷体数字识别实验。

1.2 项目说明

通过摄像头读入画面，在 FPGA 开发板中进行 RGB 转灰度、二值化等操作，对拍摄的数字（最多支持 4 个）进行定位与框线标注；根据蓝牙信号选择不同的显示模式（如图 1），并将判别的数字显示在 7 段数码管上（如图 2）。

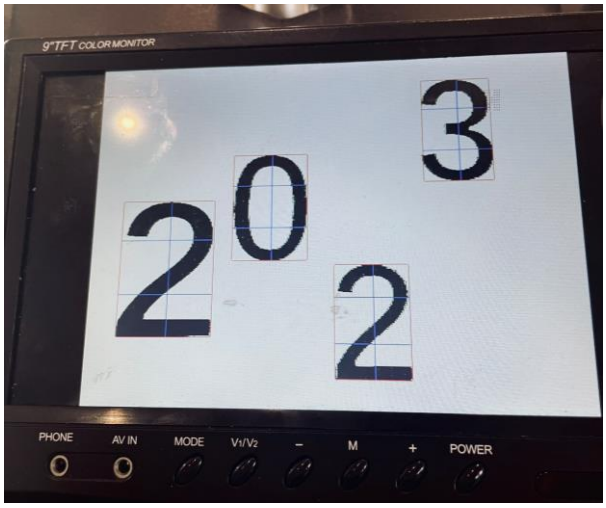

图 1

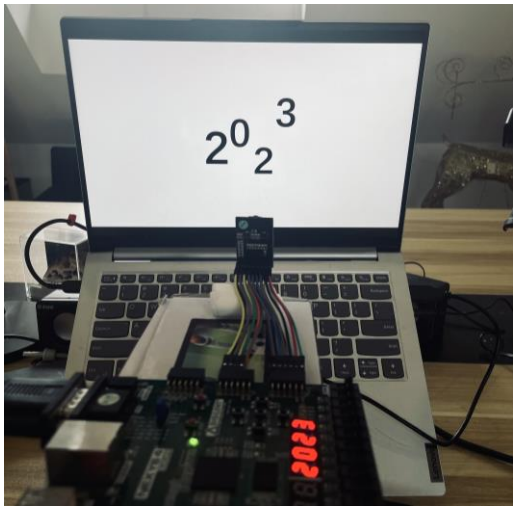

图 2

1.3 项目硬件与外围模块

开发板：Nexys4 DDR Artix-7 FPGA Board

VGA：TFT LCD Color Monitor

摄像头：OV2640 Camera Module

蓝牙：HC-06 Bluetooth Module

二、印刷体数字识别系统

2.1 系统组成总框图

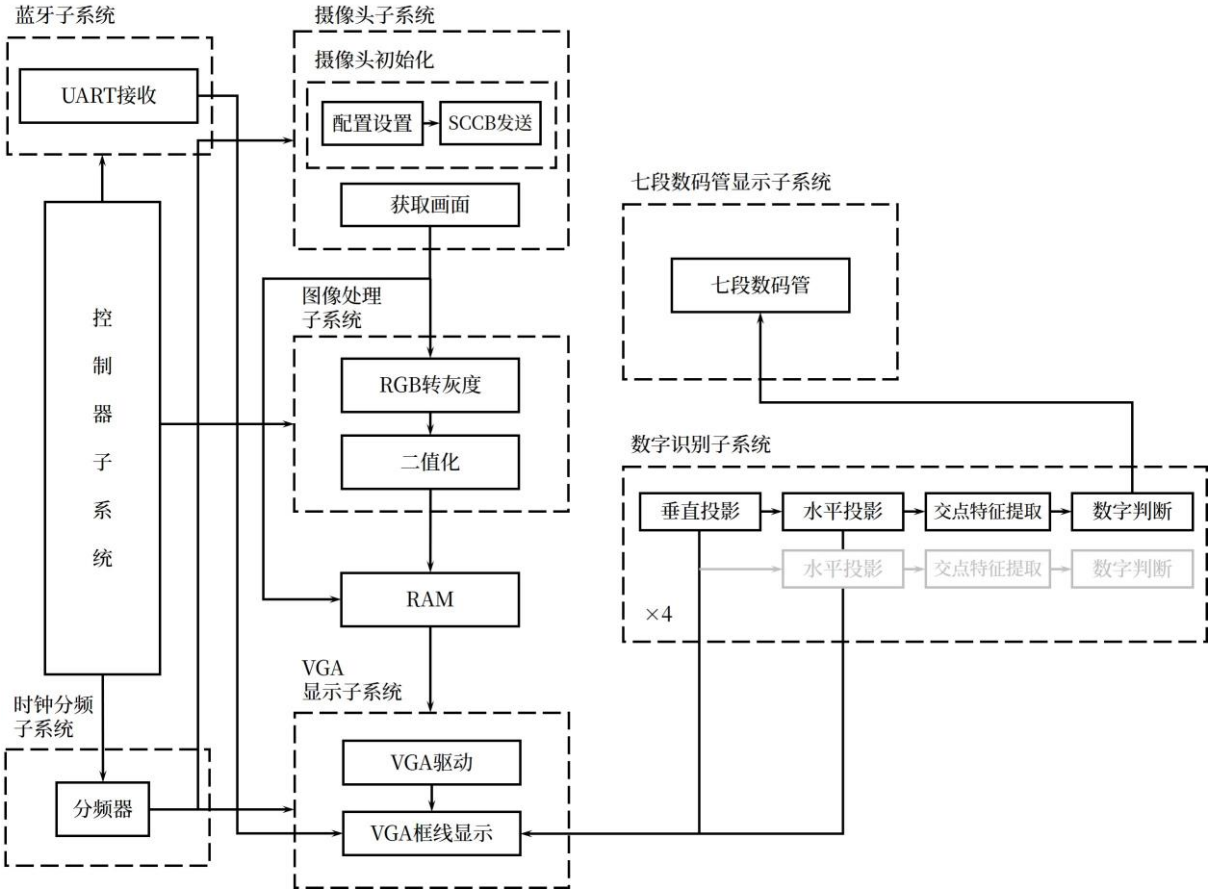

图 3　系统组成总框图

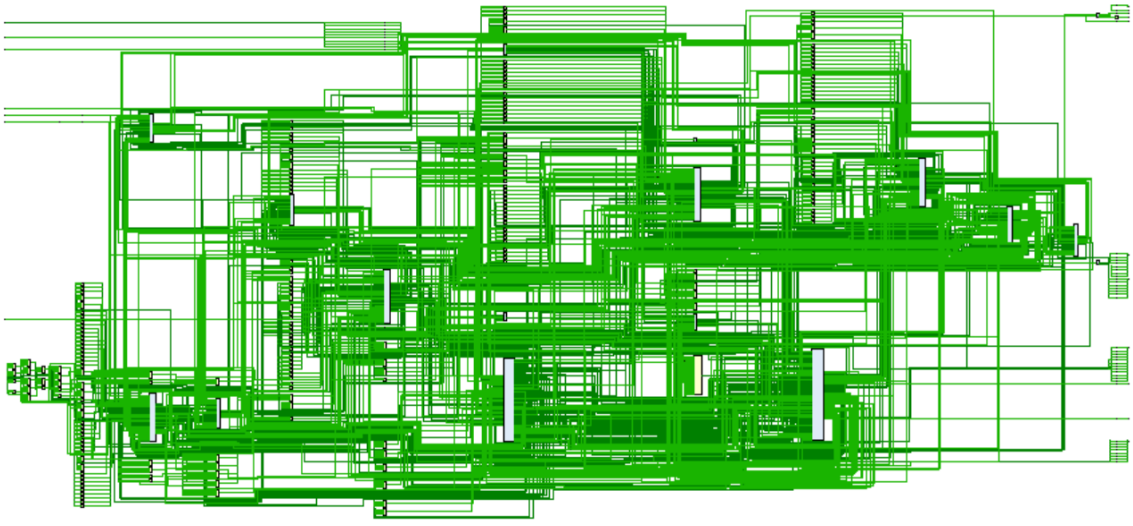

图 4　RTL 图

2.2 子系统功能简介

2.2.1 控制器子系统

控制器子系统负责控制整个印刷体数字识别系统中各个子系统的输入输出与数据交互，协调各子系统有条不紊的工作。

2.2.2 时钟分频子系统

时钟分频子系统将 100MHz 的系统时钟分频为 25.175MHz 与 25MHz 以供其余模块后续使用。

2.2.3 蓝牙子系统

蓝牙子系统负责通过 HC-06 蓝牙模块接收蓝牙信号，并根据收到的蓝牙信号决定不同的 VGA 显示模式。

2.2.4 摄像头子系统

摄像头子系统分为摄像头初始化与画面传输两部分。前者负责向摄像头寄存器中以 sccb 协议写入摄像头配置信息文件；后者负责从 OV2640 摄像头中读入像素数据，传入图像处理模块进行处理或直接写入 RAM 中用于 VGA 显示。

2.2.5 图像处理子系统

图像处理子系统从摄像头子系统中获取实时的像素等信息，并通过流水线处理进行 RGB 转灰度、二值化操作，在写入 RAM 中的同时传入数值识别子系统，以便于 VGA 图像显示以及后续数字判断。

2.2.6 VGA 显示子系统

VGA 显示子系统从 RAM 中读取 RGB444 的像素信息，在控制器子系统的控制下进行图像显示，并对数字区域进行框线标注。

2.2.7 数字识别子系统

数字识别子系统从图像处理子系统中获取像素信息。首先进行垂直投影，通过像素上升沿与下降沿确定各个数字的水平位置；然后对各个数字分别进行

水平投影以确定各个数字分别的竖直位置；之后对每个数字进行交点特征的提取与最终的数字判断，交由七段数码管显示子系统进行显示。

2.2.8 七段数码管显示子系统

七段数码管显示子系统接收来自数字识别系统的最多 4 位数字，以十进制形式在七段数码管中显示呈现；未识别到的数字位数则不显示。

三、系统控制器设计

3.1 ASM 流程图

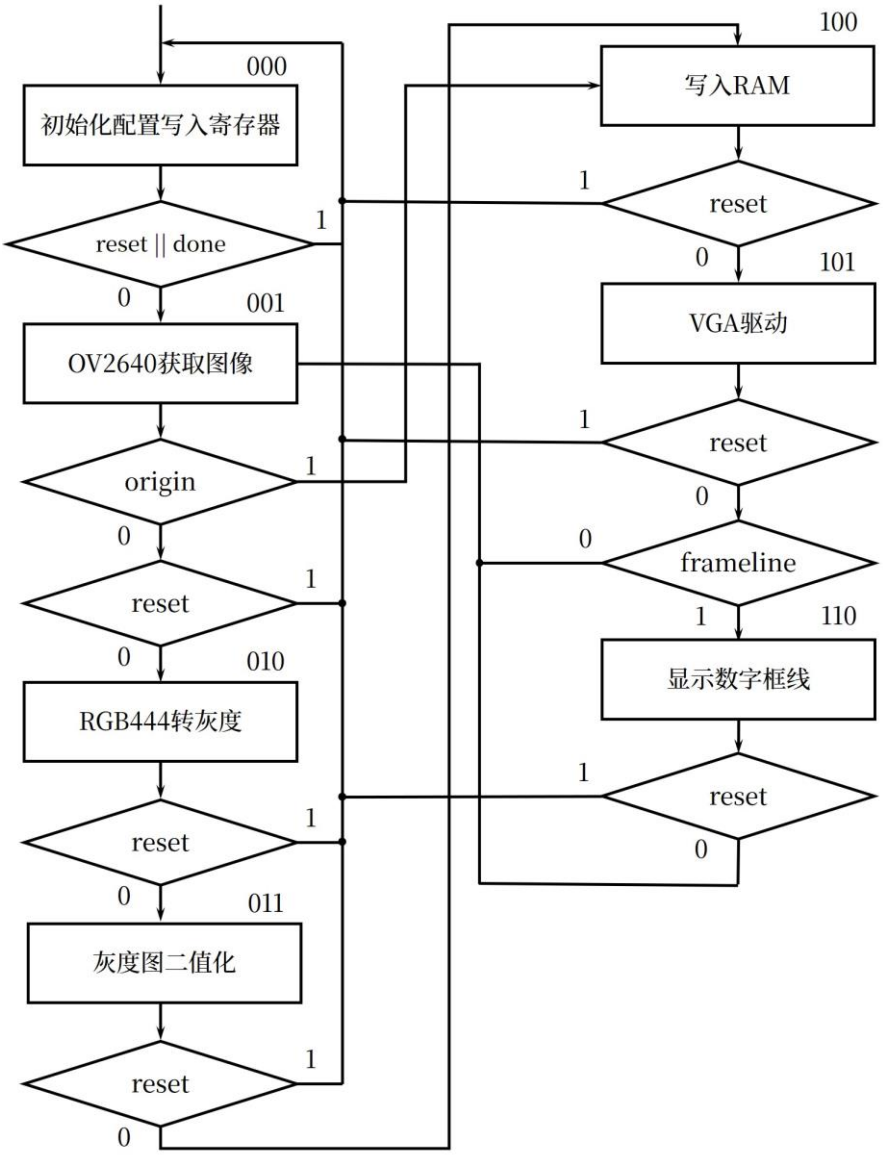

图 5

3.2 状态转移真值表

PS（现态）			NS（次态）			转换条件
C	B	A	C	B	A	
0	0	0	0	0	0	reset || !done
			0	0	1	!reset && done
0	0	1	0	0	0	reset
			0	1	0	!reset && !origin
			1	0	0	!reset && origin
0	1	0	0	0	0	reset
			0	1	1	!reset
0	1	1	0	0	0	reset
			1	0	0	!reset
1	0	0	0	0	0	reset
			1	0	1	!reset
1	0	1	0	0	0	reset
			0	0	1	!reset && !frameline
			1	1	0	!reset && frameline
1	1	0	0	0	0	reset
			0	0	1	!reset

3.3 次态激励函数表达式

$$A = (\overline{A}B + \overline{A}C + \overline{A}\,\overline{B}\,\overline{C} \cdot done + A\overline{B}C \cdot \overline{frameline}) \cdot \overline{reset}$$

$$B = (\overline{A}B\overline{C} + A\overline{B}\,\overline{C} \cdot \overline{origin} + A\overline{B}C \cdot frameline) \cdot \overline{reset}$$

$$C = (\overline{A}B + \overline{A}C + A\overline{B}\,\overline{C} \cdot origin + A\overline{B}C \cdot frameline) \cdot \overline{reset}$$

3.4 logisim 控制器逻辑方案图

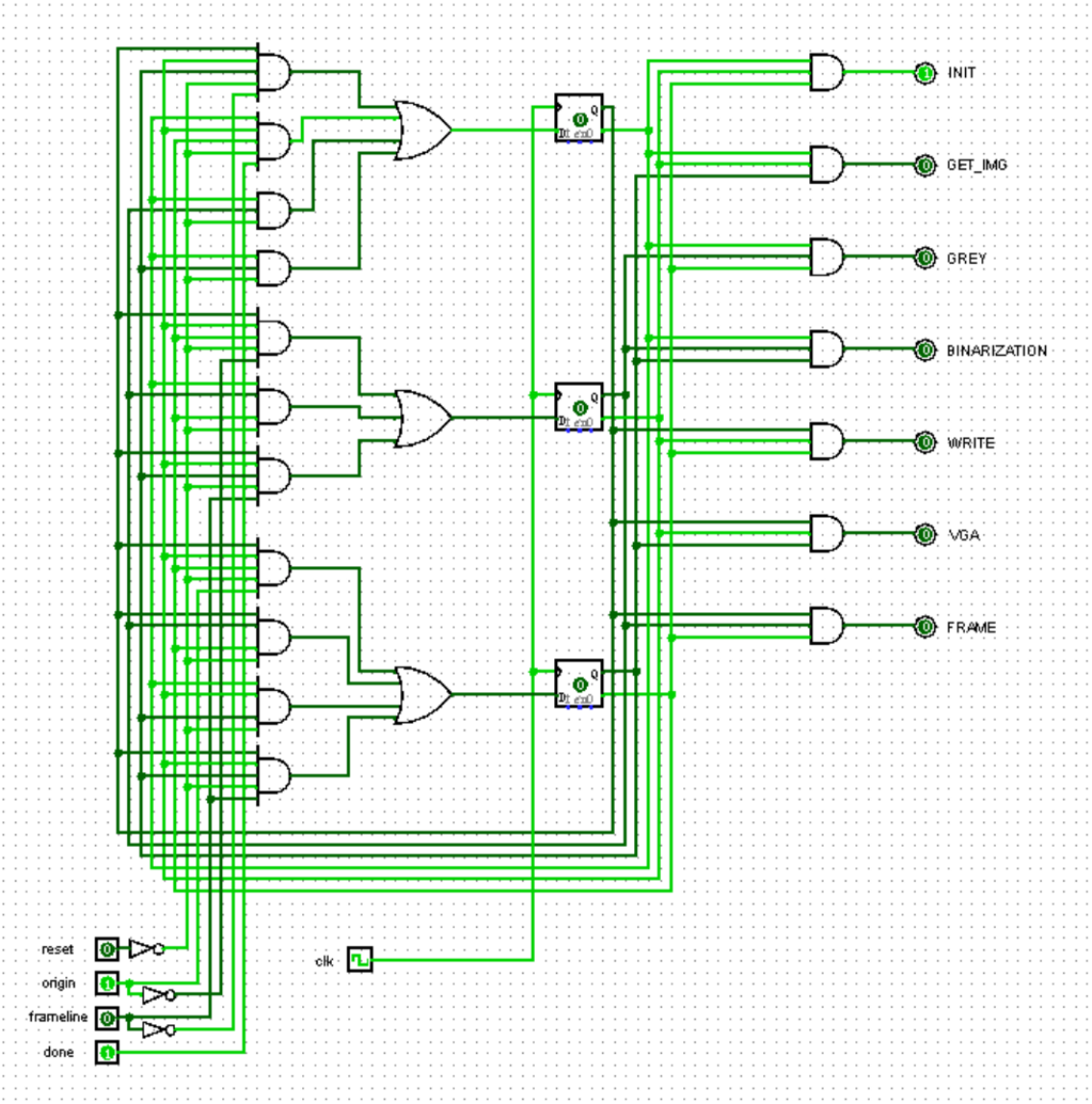

图 6

四、子系统模块建模

4.1 顶层模块

模块简介：顶层模块连接各个子模块，同时与外围模块接口相连接。

```
module top(
    input               clk,        // 系统时钟
    input               reset,      // 系统复位信号
```

```
    // 摄像头
    output              camera_sio_c,
    inout               camera_sio_d,
    output              camera_ret,
    output              camera_pwdn,
    output              camera_xclk,
    input               camera_pclk,
    input               camera_href,
    input               camera_vsync,
    input [7:0]         camera_data,

    // VGA
    output [11:0]       vga_rgb,
    output              vga_hsync,
    output              vga_vsync,

    // 蓝牙
    input               bluetooth_rxd,

    // 7 段数码管
    output [7:0]        display7_enable,
    output [6:0]        display7_segment,

    // LED 灯
    output [7:0]        led
    );


    /*********************
        线网与寄存器定义
    *********************/

    // 时钟
    wire clk_vga;
    wire clk_sccb;

    // VGA 显示
    wire        vga_display_on;
    wire [10:0] vga_pix_x;
    wire [10:0] vga_pix_y;

    // 显示模式选择
    wire        mode_binarization;
    wire        mode_frameline;


    // 摄像头读取
```

```
    wire        camera_pix_ena;
    wire [11:0] camera_data_out;
    wire [18:0] camera_data_addr;


    // 图像处理
    wire        post_href;
    wire        post_vsync;
    wire        post_clken;
    wire [11:0] post_data_out;
    wire [18:0] post_data_addr;
    wire        bin;


    // RAM 读写
    wire        ram_write_ena;
    wire [11:0] ram_write_value;
    wire [18:0] ram_write_addr;
    wire        ram_read_ena;
    wire [11:0] ram_read_value;
    wire [18:0] ram_read_addr;


    // 蓝牙
    wire [7:0]  bluetooth_data;
    wire        bluetooth_flag;


    // 数字框线
    wire [10:0] line_left;
    wire [10:0] line_left2;
    wire [10:0] line_left3;
    wire [10:0] line_left4;
    wire [10:0] line_right;
    wire [10:0] line_right2;
    wire [10:0] line_right3;
    wire [10:0] line_right4;
    wire [10:0] line_top;
    wire [10:0] line_top2;
    wire [10:0] line_top3;
    wire [10:0] line_top4;
    wire [10:0] line_bottom;
    wire [10:0] line_bottom2;
    wire [10:0] line_bottom3;
    wire [10:0] line_bottom4;

    // 交点特征
    wire [3:0]  intersection_v;
```

```
    wire [3:0]  intersection_h1;
    wire [3:0]  intersection_h2;
    wire        intersection_h1_pst;
    wire        intersection_h2_pst;
    wire [3:0]  intersection_v_2;
    wire [3:0]  intersection_h1_2;
    wire [3:0]  intersection_h2_2;
    wire        intersection_h1_pst_2;
    wire        intersection_h2_pst_2;
    wire [3:0]  intersection_v_3;
    wire [3:0]  intersection_h1_3;
    wire [3:0]  intersection_h2_3;
    wire        intersection_h1_pst_3;
    wire        intersection_h2_pst_3;
    wire [3:0]  intersection_v_4;
    wire [3:0]  intersection_h1_4;
    wire [3:0]  intersection_h2_4;
    wire        intersection_h1_pst_4;
    wire        intersection_h2_pst_4;

    // 7 段数码管
    wire [3:0]  display7_num1;
    wire [3:0]  display7_num2;
    wire [3:0]  display7_num3;
    wire [3:0]  display7_num4;

    /*********************
           线网连接
    *********************/

    assign led = bluetooth_data;

    // 显示模式选择
    assign mode_binarization = (bluetooth_data[7:4] == 4'hF);   // 高四
位为 1111 表示显示二值化图像
    assign mode_frameline    = (bluetooth_data[3:0] == 4'hF);   // 低四
位为 1111 表示显示辅助框线

    // RAM
    assign ram_write_value = mode_binarization ? post_data_out :
camera_data_out;
    assign ram_write_addr  = mode_binarization ? post_data_addr :
camera_data_addr;
    assign ram_write_ena   = mode_binarization ? post_clken :
camera_pix_ena;

    assign ram_read_addr = vga_display_on ? (vga_pix_y * 640 +
vga_pix_x) : 12'b0; //地址计算
```

```
 /*********************
      模块实例化
*********************/

// 实例化系统时钟分频
clk_divider clk_div(
    .clk_in1(clk),
    .clk_vga(clk_vga),
    .clk_sccb(clk_sccb)
);

// 实例化蓝牙接收模块
bluetooth_uart_receive bluetooth(
    .clk(clk),
    .reset(reset),
    .rxd(bluetooth_rxd),
    .data_out(bluetooth_data),
    .data_flag(bluetooth_flag)
);

// 实例化摄像头初始化模块
camera_init_top camera_init(
    .clk(clk_sccb),
    .reset(reset),
    .sio_c(camera_sio_c),
    .sio_d(camera_sio_d),
    .pwdn(camera_pwdn),
    .ret(camera_ret),
    .xclk(camera_xclk)
);

// 实例化摄像头传输画面模块
camera_get_img get_img(
    .pclk(camera_pclk),
    .reset(reset),
    .href(camera_href),
    .vsync(camera_vsync),
    .data_in(camera_data),
    .data_out(camera_data_out),
    .pix_ena(camera_pix_ena),
    .ram_out_addr(camera_data_addr)
    );

// 实例化图像处理模块
image_process img_process(
    .clk(camera_pclk),
```

```
        .reset(reset),
        .href(camera_href),
        .vsync(camera_vsync),
        .clken(camera_pix_ena),
        .rgb(camera_data_out),
        .bin(bin),
        .post_href(post_href),
        .post_vsync(post_vsync),
        .post_clken(post_clken),
        .data_out(post_data_out),
        .data_out_addr(post_data_addr)
    );

    // 实例化垂直投影模块
    vertical_projection ver_projection(
        .clk(camera_pclk),
        .reset(reset),
        .vsync(post_vsync),
        .href(post_href),
        .clken(post_clken),
        .bin(~bin),
        .line_left1(line_left),
        .line_right1(line_right),
        .line_left2(line_left2),
        .line_right2(line_right2),
        .line_left3(line_left3),
        .line_right3(line_right3),
        .line_left4(line_left4),
        .line_right4(line_right4)
    );

    // 实例化水平投影模块 1
    horizontal_projection hor_projection(
        .clk(camera_pclk),
        .reset(reset),
        .vsync(post_vsync),
        .href(post_href),
        .clken(post_clken),
        .bin(~bin),
        .line_left(line_left),
        .line_right(line_right),
        .line_top(line_top),
        .line_bottom(line_bottom)
    );

    // 实例化水平投影模块 2
    horizontal_projection hor_projection2(
        .clk(camera_pclk),
```

```
        .reset(reset),
        .vsync(post_vsync),
        .href(post_href),
        .clken(post_clken),
        .bin(~bin),
        .line_left(line_left2),
        .line_right(line_right2),
        .line_top(line_top2),
        .line_bottom(line_bottom2)
    );

    // 实例化水平投影模块 3
    horizontal_projection hor_projection3(
        .clk(camera_pclk),
        .reset(reset),
        .vsync(post_vsync),
        .href(post_href),
        .clken(post_clken),
        .bin(~bin),
        .line_left(line_left3),
        .line_right(line_right3),
        .line_top(line_top3),
        .line_bottom(line_bottom3)
    );

    // 实例化水平投影模块 4
    horizontal_projection hor_projection4(
        .clk(camera_pclk),
        .reset(reset),
        .vsync(post_vsync),
        .href(post_href),
        .clken(post_clken),
        .bin(~bin),
        .line_left(line_left4),
        .line_right(line_right4),
        .line_top(line_top4),
        .line_bottom(line_bottom4)
    );

    // 实例化交点特征统计模块 1
    intersection_count count1(
        .clk(camera_pclk),
        .reset(reset),
        .vsync(post_vsync),
        .href(post_href),
        .clken(post_clken),
        .bin(~bin),
        .line_top(line_top),
```

```
        .line_bottom(line_bottom),
        .line_left(line_left),
        .line_right(line_right),
        .v_cnt(intersection_v),
        .h_cnt1(intersection_h1),
        .h_cnt2(intersection_h2),
        .h1(intersection_h1_pst),
        .h2(intersection_h2_pst)
    );

    // 实例化交点特征统计模块 2
    intersection_count count2(
        .clk(camera_pclk),
        .reset(reset),
        .vsync(post_vsync),
        .href(post_href),
        .clken(post_clken),
        .bin(~bin),
        .line_top(line_top2),
        .line_bottom(line_bottom2),
        .line_left(line_left2),
        .line_right(line_right2),
        .v_cnt(intersection_v_2),
        .h_cnt1(intersection_h1_2),
        .h_cnt2(intersection_h2_2),
        .h1(intersection_h1_pst_2),
        .h2(intersection_h2_pst_2)
    );

    // 实例化交点特征统计模块 3
    intersection_count count3(
        .clk(camera_pclk),
        .reset(reset),
        .vsync(post_vsync),
        .href(post_href),
        .clken(post_clken),
        .bin(~bin),
        .line_top(line_top3),
        .line_bottom(line_bottom3),
        .line_left(line_left3),
        .line_right(line_right3),
        .v_cnt(intersection_v_3),
        .h_cnt1(intersection_h1_3),
        .h_cnt2(intersection_h2_3),
        .h1(intersection_h1_pst_3),
        .h2(intersection_h2_pst_3)
    );
```

```
    // 实例化交点特征统计模块 4
    intersection_count count4(
        .clk(camera_pclk),
        .reset(reset),
        .vsync(post_vsync),
        .href(post_href),
        .clken(post_clken),
        .bin(~bin),
        .line_top(line_top4),
        .line_bottom(line_bottom4),
        .line_left(line_left4),
        .line_right(line_right4),
        .v_cnt(intersection_v_4),
        .h_cnt1(intersection_h1_4),
        .h_cnt2(intersection_h2_4),
        .h1(intersection_h1_pst_4),
        .h2(intersection_h2_pst_4)
    );

    // 实例化数字判断模块
    figure_recognition fig_recogn(
        .v_cnt(intersection_v),
        .h_cnt1(intersection_h1),
        .h_cnt2(intersection_h2),
        .h1(intersection_h1_pst),
        .h2(intersection_h2_pst),
        .figure(display7_num1)
    );

    // 实例化数字判断模块 2
    figure_recognition fig_recogn2(
        .v_cnt(intersection_v_2),
        .h_cnt1(intersection_h1_2),
        .h_cnt2(intersection_h2_2),
        .h1(intersection_h1_pst_2),
        .h2(intersection_h2_pst_2),
        .figure(display7_num2)
    );


    // 实例化数字判断模块 3
    figure_recognition fig_recogn3(
        .v_cnt(intersection_v_3),
        .h_cnt1(intersection_h1_3),
        .h_cnt2(intersection_h2_3),
        .h1(intersection_h1_pst_3),
        .h2(intersection_h2_pst_3),
        .figure(display7_num3)
```

```
    );

    // 实例化数字判断模块 4
    figure_recognition fig_recogn4(
        .v_cnt(intersection_v_4),
        .h_cnt1(intersection_h1_4),
        .h_cnt2(intersection_h2_4),
        .h1(intersection_h1_pst_4),
        .h2(intersection_h2_pst_4),
        .figure(display7_num4)
    );

    // 实例化双端口 ram
    ram ram_inst (
        .clka(clk),
        .wea(ram_write_ena),
        .addra(ram_write_addr),
        .dina(ram_write_value),
        .clkb(clk),
        .enb(1'b1),
        .addrb(ram_read_addr),
        .doutb(ram_read_value)
    );

    // 实例化 VGA 显示
    vga_sync vga_display(
        .vga_clk(clk_vga),
        .reset(reset),
        .hsync(vga_hsync),
        .vsync(vga_vsync),
        .display_on(vga_display_on),
        .pixel_x(vga_pix_x),
        .pixel_y(vga_pix_y)
    );

    // 实例化 vga 显示控制模块
    vga_control vga_ctrl(
        .org_rgb(ram_read_value),
        .display_on(vga_display_on),
        .pixel_x(vga_pix_x),
        .pixel_y(vga_pix_y),
        .ena(mode_frameline),
        .line_left(line_left),
        .line_left2(line_left2),
        .line_left3(line_left3),
        .line_left4(line_left4),
        .line_right(line_right),
        .line_right2(line_right2),
```

```
        .line_right3(line_right3),
        .line_right4(line_right4),
        .line_top(line_top),
        .line_top2(line_top2),
        .line_top3(line_top3),
        .line_top4(line_top4),
        .line_bottom(line_bottom),
        .line_bottom2(line_bottom2),
        .line_bottom3(line_bottom3),
        .line_bottom4(line_bottom4),
        .out_rgb(vga_rgb)
    );

    // 实例化 7 段数码管显示模块
    display7 display7_seg(
        .clk(clk),
        .num1(display7_num1),
        .num2(display7_num2),
        .num3(display7_num3),
        .num4(display7_num4),
        .enable(display7_enable),
        .segment(display7_segment)
    );

endmodule
```

4.2 时钟分频模块

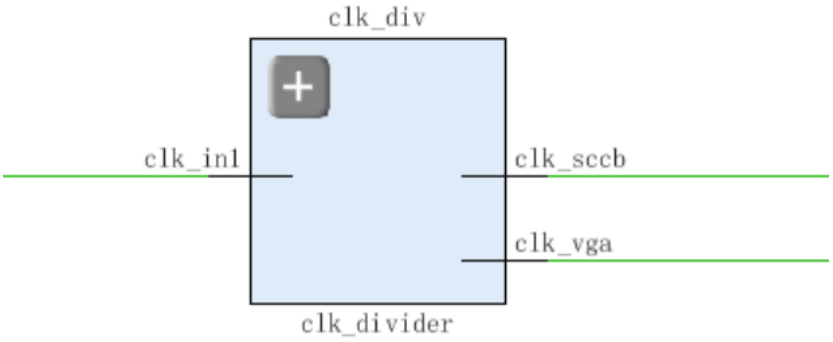

图 7

模块简介：使用 Vivado 中的 IP 核 Clocking Wizard 对输入的系统时钟（100MHz）进行分频，输出 25.175MHz 与 25MHz 频率的时钟用于 VGA 显示模块与 sccb 信息传输模块使用。

4.3 蓝牙接收模块

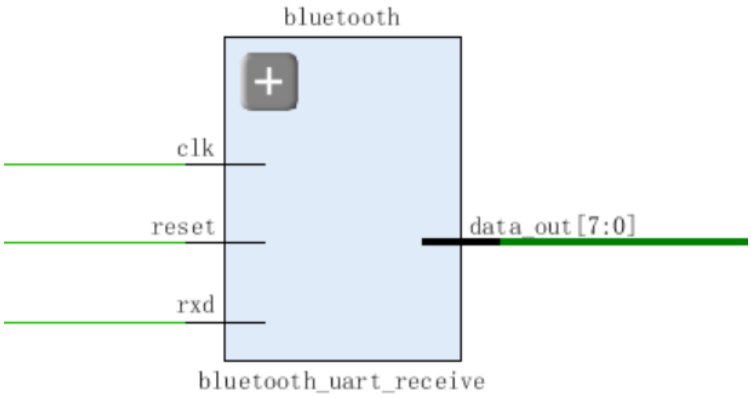

图 8

模块简介：将通过 rxd 串行输入的收到的蓝牙信号每 8 位进行一次并行输出，并在输出时将输出有效信号 data_flag 拉高。

```
module bluetooth_uart_receive(
    input clk,                          // 系统时钟
    input reset,                        // 复位信号
    input rxd,                          // 串行输入数据
    output reg [7:0] data_out,          // 并行输出数据
    output reg data_flag                // 并行数据输出信号
    );

    parameter CLK_FREQ = 100000000;              // 系统时钟频率
    parameter UART_BPS = 9600;                   // 串口波特率
    localparam BPS_CNT = CLK_FREQ / UART_BPS;

    // 寄存器与线网定义
    reg rxd_reg1;       // 打三拍 消除亚稳态
    reg rxd_reg2;
    reg rxd_reg3;
    wire start_flag;    // 稳定下降沿信号
    reg [14:0] clk_cnt;
    reg [3:0] bit_cnt;
    reg work_flag;      // 开始 8bit 的串转并
    reg [7:0] rx_data;

    // 稳定下降沿 开始接收
    assign start_flag = (~rxd_reg2) & rxd_reg3;

    always @(posedge clk or negedge reset)
```

```
    begin
        if(reset)
            { rxd_reg1, rxd_reg2, rxd_reg3 } <= 3'b111;
        else
            { rxd_reg1, rxd_reg2, rxd_reg3 } <= { rxd, rxd_reg1,
rxd_reg2 };
    end

    // 开始读入 8bit 信号的设置
    always @(posedge clk or negedge reset)
    begin
        if(reset)
            work_flag <= 1'b0;
        else
        begin
            // 开始读入 8bit
            if(start_flag)
                work_flag <= 1'b1;
            // 当前 8bit 读入完毕 等待新信号
            else if((bit_cnt == 9) && (clk_cnt == BPS_CNT / 2))
                work_flag <= 1'b0;
            else
                work_flag <= work_flag;
        end
    end

    // 根据波特率与时钟频率计时
    always @(posedge clk or negedge reset)
    begin
        if(reset)
        begin
            clk_cnt <= 0;
            bit_cnt <= 0;
        end
        else if(work_flag)
        begin
            if(clk_cnt < BPS_CNT - 1)
            begin
                clk_cnt <= clk_cnt + 1'b1;
                bit_cnt <= bit_cnt;
            end
            else
            begin
                clk_cnt <= 0;
                bit_cnt <= bit_cnt + 1'b1;
            end
        end
        else
```

```
        begin
            clk_cnt <= 0;
            bit_cnt <= 0;
        end
    end

    // 串行输入转并行
    always @(posedge clk or negedge reset)
    begin
        if(reset)
            rx_data <= 8'b0;
        else if(work_flag)
            // 周期正中读入较稳定 校验位舍弃
            if(clk_cnt == BPS_CNT / 2 && bit_cnt != 9)
                rx_data <= { rxd_reg3, rx_data[7:1] };
            else
                rx_data <= rx_data;
        else
            rx_data <= 8'b0;
    end

    // 8bit 发送结束 并行输出
    always @(posedge clk or negedge reset)
    begin
        if(reset)
            data_out <= 8'b0;
        else if(bit_cnt == 9)
            data_out <= rx_data;
        else
            data_out <= data_out;
    end

    // 输出标志位拉高
    always @(posedge clk or negedge reset)
    begin
        if(reset)
            data_flag <= 0;
        else if(bit_cnt == 9)
            data_flag <= 1;
        else
            data_flag <= 0;
    end
endmodule
```

4.4 摄像头初始化模块

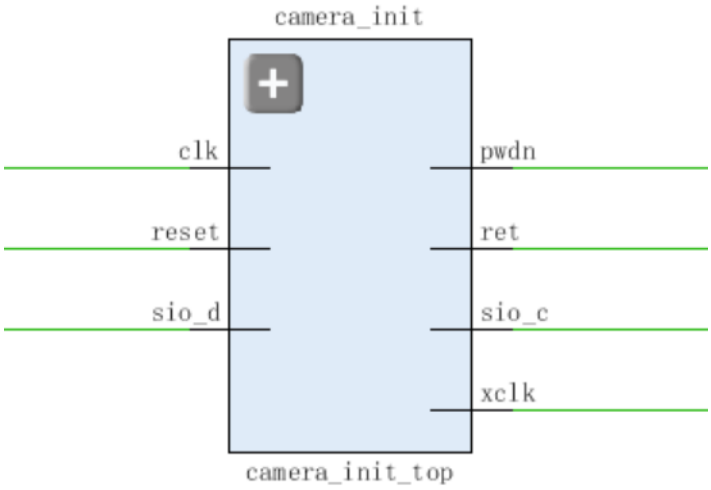

图 9

模块简介：连接配置文件模块与 sccb 发送模块（由于此部分的两个子模块代码较冗长，不在实验报告中展示）。将初始化的配置文件信息通过 sio_c 与 sio_d 写入摄像头的配置文件中；写入完毕后拉低 pwdn 并拉高 ret。

```
module camera_init_top(
    input clk,       // 25MHz 时钟
    input reset,     // 复位信号
    // 以下是与摄像头相连的管脚 名称对应相同
    output sio_c,
    inout  sio_d,
    output pwdn,
    output ret,
    output xclk
    );

    // pwdn 高电平有效 ret 低电平有效
    assign pwdn = 0;
    assign ret = 1;
    // sio_d 高阻态
    pullup up (sio_d);
    // 赋给 xclk 时钟信号
    assign xclk = clk;

    wire cfg_ok, sccb_ok;
    wire [15: 0] data_sent;
```

```
    // 实例化配置写入模块
    camera_reg_cfg reg_cfg(
        .clk(clk),
        .reset(reset),
        .data_out(data_sent),
        .cfg_ok(cfg_ok),
        .sccb_ok(sccb_ok)
    );
    // 实例化 sccb 发送模块
    camera_sccb_sender sccb_sender(
        .clk(clk),
        .reset(reset),
        .sio_c(sio_c),
        .sio_d(sio_d),
        .cfg_ok(cfg_ok),
        .sccb_ok(sccb_ok),
        .reg_addr(data_sent[15:8]),
        .value(data_sent[7:0])
    );
endmodule
```

4.5 摄像头画面传输模块

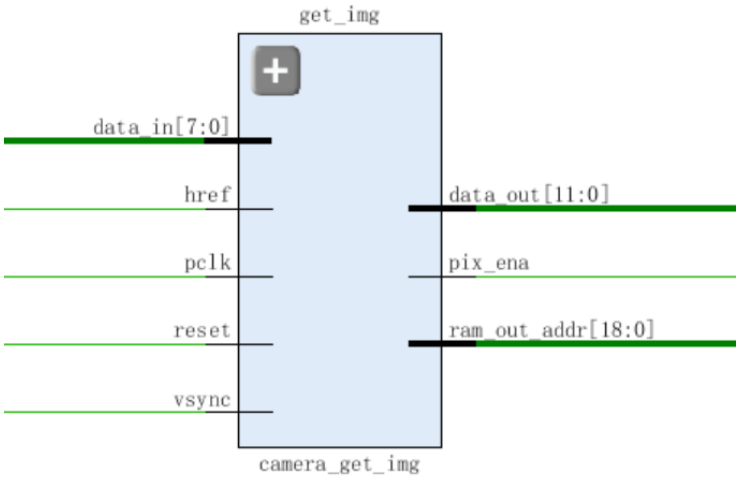

图 10

模块简介：负责处理摄像头传输的像素信息。将 href 有效时每两个 pclk 传输的 RGB565 信息整合成一个像素对应的 RGB444 信息，每两个 pclk 输出一次像素写入使能信号；并通过 pclk、href、vsync 计算出对应的 RAM 写入地址。

```
module camera_get_img(
    input                pclk,           // 像素时钟（1 像素周期为 2pclk）
```

```
    input               reset,          // 复位信号
    input               href,           // 行参考信号
    input               vsync,          // 帧同步
    input       [7:0]   data_in,        // 从摄像头读入的 RGB565 信息(两个
pclk)
    output reg [11:0]   data_out,       // 输出的一像素 RGB444
    output reg          pix_ena,        // 新的像素输出
    output reg [18:0]   ram_out_addr    // 应该写入的 RAM 地址
);


    reg [15:0] rgb565 = 0;
    reg [1:0]  bit_status = 0;      // 两个 pclk 对应一次输出
    reg [18:0] ram_next_addr;

    initial ram_next_addr <= 0;


    always@ (posedge pclk)
    begin

        // 开始输出新的一帧 从头写入 RAM
        if(vsync == 0)
        begin
            ram_out_addr <= 0;
            ram_next_addr <= 0;
            bit_status <= 0;
        end
        else
        begin
            // RGB565 取高位压缩为 RGB444
            data_out <= { rgb565[15:12], rgb565[10:7], rgb565[4:1] };
            ram_out_addr <= ram_next_addr;
            pix_ena <= bit_status[1];

            // 两个 pclk 输出一次
            bit_status <= {bit_status[0], (href && !bit_status[0])};

            // 两字节信息 拼成 16bit 的 RGB565
            rgb565 <= { rgb565[7:0], data_in };
            if(bit_status[1] == 1)
                ram_next_addr <= ram_next_addr + 1;
        end

    end

endmodule
```

4.6 图像处理模块

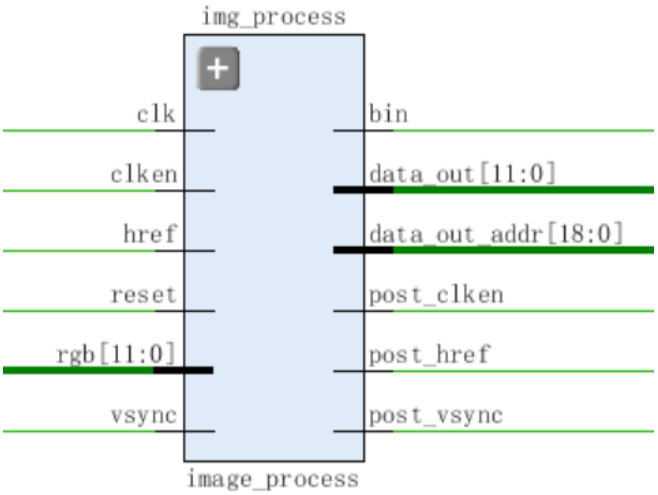

图 11

模块简介：对摄像头画面传输模块输出的像素信息进行流水线处理。分为 RGB 转灰度图、二值化两个子模块（由于报告篇幅限制不在此展示）。对实时传入的像素信息进行灰度、二值化处理。由于图像处理有周期上的延迟，对 clken、href、vsync 信号进行对应的延迟处理，并计算对应写入的 RAM 地址。

```
module image_process(
    input                clk,                // 时钟
    input                reset,              // 复位信号
    input                href,               // 处理前的行参考信号
    input                vsync,              // 处理前的场同步信号
    input                clken,              // 处理前像素有效信号
    input        [11:0]  rgb,                // 处理前的 RGB444
    output               bin,                // 二值化后的数值
    output               post_href,          // 处理后的行参考信号
    output               post_vsync,         // 处理后的场同步信号
    output               post_clken,         // 处理后像素有效信号
    output       [11:0]  data_out,           // 二值化后的数值(0/255)
    output reg   [18:0]  data_out_addr       // 写入 RAM 的地址
    );

    // 线网定义
    wire [7:0] grey;
    wire post_href0,  post_href1;
    wire post_vsync0, post_vsync1;
    wire post_clken0, post_clken1;

    initial data_out_addr <= 0;
```

```
    // 线网连接
    assign data_out = bin ? 12'hFFF : 12'h000;
    assign post_href  = post_href1;
    assign post_vsync = post_vsync1;
    assign post_clken = post_clken1;

    always @ (posedge clk)
    begin
        if(post_vsync1 == 0)
        begin
            data_out_addr <= 0;
        end
        else if(post_clken1)
        begin
            data_out_addr <= data_out_addr + 1;
        end
    end

    // rgb 转灰度模块实例化
    rgb2grey rgb2grey_inst(
        .clk(clk),
        .reset(reset),
        .org_href(href),
        .org_vsync(vsync),
        .org_clken(clken),
        .org_rgb(rgb),
        .grey(grey),
        .out_href(post_href0),
        .out_vsync(post_vsync0),
        .out_clken(post_clken0)
    );

    // 灰度图二值化模块实例化
    binarization bin_inst(
        .clk(clk),
        .reset(reset),
        .org_href(post_href0),
        .org_vsync(post_vsync0),
        .org_clken(post_clken0),
        .grey(grey),
        .bin(bin),
        .out_href(post_href1),
        .out_vsync(post_vsync1),
        .out_clken(post_clken1)
    );

endmodule
```

4.7 垂直投影模块

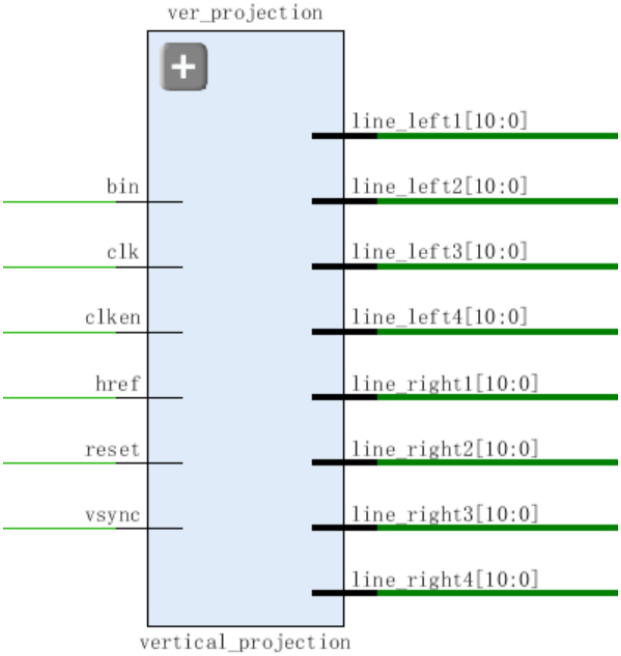

图 12

模块简介：对二值化处理后的图像画面进行垂直投影。根据投影后数据的上升沿和下降沿判断数字的左右边界（最多可检测到 4 个数字）；若未检测到上升下降沿则框线值为 0。输出的框线信息用以后续的水平投影模块。

```
module vertical_projection(
    input               clk,            // 像素时钟
    input               reset,          // 复位信号
    input               vsync,          // 帧同步
    input               href,           // 行参考
    input               clken,          // 像素有效信号
    input               bin,            // 输入的二值化 bit 信号（黑色为 1，白色为 0）

    output reg [10:0]   line_left1,     // 数字的左右框线
    output reg [10:0]   line_right1,
    output reg [10:0]   line_left2,
    output reg [10:0]   line_right2,
    output reg [10:0]   line_left3,
    output reg [10:0]   line_right3,
    output reg [10:0]   line_left4,
    output reg [10:0]   line_right4
);
```

```
    parameter   DISPLAY_WIDTH  = 10'd640;
    parameter   DISPLAY_HEIGHT = 10'd480;

    /*********************
        线网与寄存器定义
    *********************/

    reg [10:0]      x_cnt;           // 行场计数
    reg [10:0]      y_cnt;
    reg [10:0]      reg_x_cnt;       // 行场计数寄存
    reg [10:0]      reg_y_cnt;

    reg             ram_wr_ena;
    wire [9:0]      ram_wr_data;
    wire [9:0]      ram_rd_data;

    reg             reg_bin;        // 寄存 bit 信号

    reg [9:0]       rd_data1;       // 读到的信号寄存数个周期
    reg [9:0]       rd_data2;
    reg [9:0]       rd_data3;
    reg [9:0]       rd_data4;
    reg [9:0]       rd_data5;
    reg [9:0]       rd_data6;
    reg [9:0]       rd_data7;

    reg [10:0]      leftline1;      // 寄存左右框线
    reg [10:0]      rightline1;
    reg [10:0]      leftline2;
    reg [10:0]      rightline2;
    reg [10:0]      leftline3;
    reg [10:0]      rightline3;
    reg [10:0]      leftline4;
    reg [10:0]      rightline4;
    reg [10:0]      lastposline;
    reg [10:0]      lastnegline;

    reg [3:0]       posedge_cnt;    // 上升沿计数
    reg [3:0]       negedge_cnt;    // 下降沿计数

    // 输入的二值化信号打一排 用于写入 ram
    always@(posedge clk or posedge reset)
    begin
        if(reset)
            reg_bin <= 0;
        else
            reg_bin <= bin;
    end
```

```
// 对行场方向分别计数用于投影
always@(posedge clk or posedge reset)
begin
    if(reset)
        begin
            x_cnt <= 10'd0;
            y_cnt <= 10'd0;
        end
    else
        if(vsync == 0)
        begin
            x_cnt <= 10'd0;
            y_cnt <= 10'd0;
        end
        else if(clken)
        begin
            if(x_cnt < DISPLAY_WIDTH - 1)
            begin
                x_cnt <= x_cnt + 1'b1;
                y_cnt <= y_cnt;
            end
            else
            begin
                x_cnt <= 10'd0;
                y_cnt <= y_cnt + 1'b1;
            end
        end
end

// 行场计数寄存一个时钟周期 用于写
always@(posedge clk or posedge reset)
begin
    if(reset)
    begin
        reg_x_cnt <= 10'd0;
        reg_y_cnt <= 10'd0;
    end
    else
    begin
        reg_x_cnt <= x_cnt;
        reg_y_cnt <= y_cnt;
    end
end

// 像素使能信号打一拍 延迟用于写入
always @ (posedge clk or posedge reset)
begin
```

```
        if(reset)
            ram_wr_ena <= 1'b0;
        else
            ram_wr_ena <= clken;
    end

    // 新的一帧开始时 清空已有的 ram 信息
    assign ram_wr_data = (y_cnt == 10'd0) ? 10'd0 : (ram_rd_data +
reg_bin);

    ram_vertical_projection ram_ver (
      .clka(clk),
      .wea(ram_wr_ena),
      .addra(reg_x_cnt),
      .dina(ram_wr_data),
      .clkb(clk),
      .addrb(x_cnt),
      .doutb(ram_rd_data)
    );

    // 寄存 3 级 用于后续判断上升下降沿
    always @ (posedge clk or posedge reset)
    begin
        if(reset || vsync == 0)
        begin
            rd_data1 <= 10'd0;
            rd_data2 <= 10'd0;
            rd_data3 <= 10'd0;
            rd_data4 <= 10'd0;
            rd_data5 <= 10'd0;
            rd_data6 <= 10'd0;
            rd_data7 <= 10'd0;
        end
        else if(clken)
        begin
            rd_data1 <= ram_rd_data;
            rd_data2 <= rd_data1;
            rd_data3 <= rd_data2;
            rd_data4 <= rd_data3;
            rd_data5 <= rd_data4;
            rd_data6 <= rd_data5;
            rd_data7 <= rd_data6;
        end
    end

    always @ (posedge clk or posedge reset)
    begin
        if(reset)
```

```
        begin
            leftline1  <= 10'b0;
            leftline2  <= 10'b0;
            leftline3  <= 10'b0;
            leftline4  <= 10'b0;
            rightline1 <= 10'b0;
            rightline2 <= 10'b0;
            rightline3 <= 10'b0;
            rightline4 <= 10'b0;
            posedge_cnt <= 4'd0;
            negedge_cnt <= 4'd0;
        end
        else if(x_cnt == 1 && y_cnt == 1)
        begin
            leftline1  <= 10'b0;
            leftline2  <= 10'b0;
            leftline3  <= 10'b0;
            leftline4  <= 10'b0;
            rightline1 <= 10'b0;
            rightline2 <= 10'b0;
            rightline3 <= 10'b0;
            rightline4 <= 10'b0;
            posedge_cnt <= 4'd0;
            negedge_cnt <= 4'd0;
        end
        else if(clken)
        begin
            // 最后一行开始统计上升下降沿
            if(y_cnt == DISPLAY_HEIGHT - 1'b1)
            begin
                if((rd_data7 == 10'd0) && (ram_rd_data > 10'd10)
                && (posedge_cnt == 0 || reg_x_cnt - lastposline >
10'd50))
                begin
                    posedge_cnt = posedge_cnt + 1;
                    lastposline <= reg_x_cnt - 2;
                    case(posedge_cnt)
                        4'd1 : leftline1  <= reg_x_cnt - 2;
                        4'd2 : leftline2  <= reg_x_cnt - 2;
                        4'd3 : leftline3  <= reg_x_cnt - 2;
                        4'd4 : leftline4  <= reg_x_cnt - 2;
                    endcase
                end

                if((rd_data7 > 10'd10) && (ram_rd_data == 10'd0)
                && (negedge_cnt == 0 || reg_x_cnt - lastnegline >
10'd50))
                begin
```

```
                    negedge_cnt = negedge_cnt + 1;
                    lastnegline <= reg_x_cnt - 2;
                    case(negedge_cnt)
                        4'd1 : rightline1  <= reg_x_cnt - 2;
                        4'd2 : rightline2  <= reg_x_cnt - 2;
                        4'd3 : rightline3  <= reg_x_cnt - 2;
                        4'd4 : rightline4  <= reg_x_cnt - 2;
                    endcase
                end
            end
        end
    end

    // 一帧写入完毕后 再更新线条位置
    always @ (posedge clk or posedge reset)
    begin
        if(reset)
        begin
            line_left1  <= 10'd0;
            line_right1 <= 10'd0;
            line_left2  <= 10'd0;
            line_right2 <= 10'd0;
            line_left3  <= 10'd0;
            line_right3 <= 10'd0;
            line_left4  <= 10'd0;
            line_right4 <= 10'd0;
        end
        else if(vsync == 0)
        begin
            line_left1  <= leftline1;
            line_right1 <= rightline1;
            line_left2  <= leftline2;
            line_right2 <= rightline2;
            line_left3  <= leftline3;
            line_right3 <= rightline3;
            line_left4  <= leftline4;
            line_right4 <= rightline4;
        end
    end

endmodule
```

4.8 水平投影模块

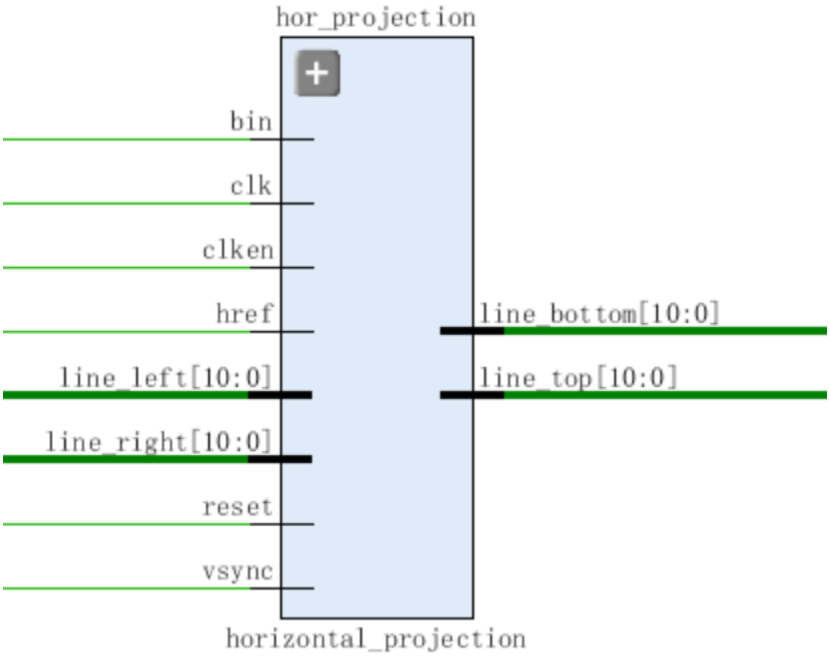

图 13

模块简介：该模块共 4 个，分别对应 4 个数字的垂直投影。即对二值化处理后的图像画面在垂直投影模块确定的竖直范围内进行水平投影。根据投影后数据的上升沿和下降沿判断数字的上下边界；若未检测到上升下降沿则框线值为 0。输出的框线信息用以后续的交点特征提取模块。

```
module horizontal_projection(
    input               clk,                // 像素时钟
    input               reset,              // 复位信号
    input               vsync,              // 帧同步
    input               href,               // 行参考
    input               clken,              // 像素使能
    input               bin,                // 输入的二值化 bit 信号（黑色
为 1，白色为 0）
    input      [10:0]   line_left,          // 数字左右框线
    input      [10:0]   line_right,
    output reg [10:0]   line_top,           // 数字上下框线
    output reg [10:0]   line_bottom
);

    parameter   DISPLAY_WIDTH  = 10'd640;
    parameter   DISPLAY_HEIGHT = 10'd480;
```

```
/*********************
    线网与寄存器定义
*********************/

// 行场计数
reg [10:0]      x_cnt;
reg [10:0]      y_cnt;

// 寄存行水平投影
reg [9:0]       tot;
reg [9:0]       tot1;
reg [9:0]       tot2;
reg [9:0]       tot3;

// 寄存上升沿下降沿位置
reg [10:0]      topline1;
reg [10:0]      bottomline1;


// 对行场方向分别计数用于投影
always@(posedge clk or posedge reset)
begin
    if(reset)
        begin
            x_cnt <= 10'd0;
            y_cnt <= 10'd0;
        end
    else
        if(vsync == 0)
        begin
            x_cnt <= 10'd0;
            y_cnt <= 10'd0;
        end
        else if(clken)
        begin
            if(x_cnt < DISPLAY_WIDTH - 1)
            begin
                x_cnt <= x_cnt + 1'b1;
                y_cnt <= y_cnt;
            end
            else
            begin
                x_cnt <= 10'd0;
                y_cnt <= y_cnt + 1'b1;
            end
        end
end
```

```
    // 每一行进行计数
    always@(posedge clk or posedge reset)
    begin
        if(reset)
            tot <= 10'b0;
        else if(clken)
        begin
            if(x_cnt == 0)
                tot <= 10'b0;
            // 只在数字范围内进行投影
            else if(x_cnt > line_left && x_cnt < line_right)
                tot <= tot + bin;
        end
    end

    // 寄存 3 级 用于后续判断上升下降沿
    always @ (posedge clk or posedge reset)
    begin
        if(reset)
        begin
            tot1 <= 10'd0;
            tot2 <= 10'd0;
            tot3 <= 10'b0;
        end
        else if(clken && x_cnt == DISPLAY_WIDTH - 1)
        begin
            tot1 <= tot;
            tot2 <= tot1;
            tot3 <= tot2;
        end
    end

    always @ (posedge clk or posedge reset)
    begin
        if(reset)
        begin
            topline1    <= 10'd0;
            bottomline1 <= 10'd0;
        end
        else if(clken)
        begin
            // 最后一行开始统计上升下降沿
            if(x_cnt == DISPLAY_WIDTH - 1'b1)
            begin
                if((tot3 == 10'd0) && (tot > 10'd10))
                    topline1 <= y_cnt - 3;

                if((tot3 > 10'd10) && (tot == 10'd0))
```

```
                    bottomline1 <= y_cnt - 3;
            end
        end
    end

    // 一帧写入完毕后 再更新线条位置
    always @ (posedge clk or posedge reset)
    begin
        if(reset)
        begin
            line_top    <= 10'd0;
            line_bottom <= 10'd0;
        end
        else if(vsync == 0)
        begin
            line_top    <= topline1;
            line_bottom <= bottomline1;
        end
    end

endmodule
```

4.9 交点特征提取模块

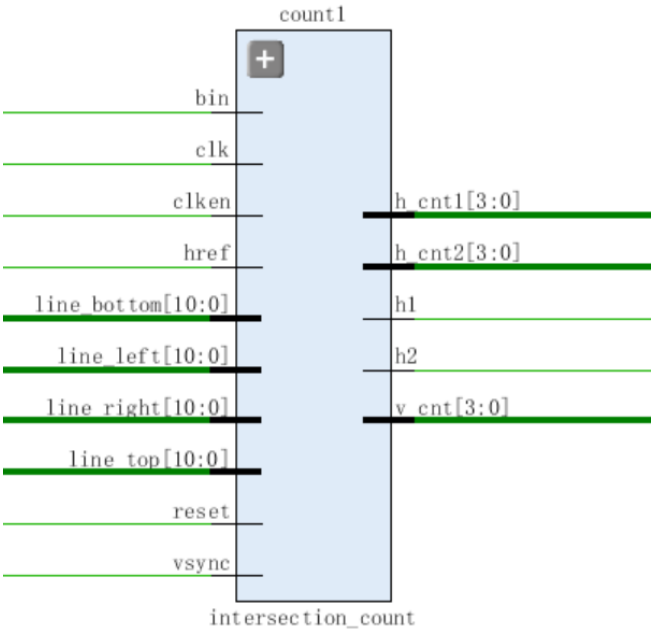

图 14

模块简介：该模块共 4 个，分别对应 4 个数字的交点特征提取。即在根据垂直、水平所判断的数字边界内，求算印刷体数字与竖直（1 条）、水平（2 条）

分割线的交点个数与位置，将所提取到底交点特征进行输出。

```
module intersection_count(
    input               clk,                // 时钟
    input               reset,              // 复位信号
    input               vsync,              // 场同步信号
    input               href,               // 行参考信号
    input               clken,              // 像素有效信号
    input               bin,                // 输入的二值化 bit 信号（黑色
为 1，白色为 0）
    input   [10:0]      line_top,           // 识别出的数字定位框线
    input   [10:0]      line_bottom,
    input   [10:0]      line_left,
    input   [10:0]      line_right,
    output reg [3:0]    v_cnt,              // 与竖直分割线的交点个数
    output reg [3:0]    h_cnt1,             // 与水平分割线 1 的交点个数
    output reg [3:0]    h_cnt2,             // 与水平分割线 2 的交点个数
    output reg          h1,                 // 最后一个与水平分割线 1 的交
点在中轴线左侧还是右侧
    output reg          h2                  // 最后一个与水平分割线 2 的交
点在中轴线左侧还是右侧
    );

    // 参数定义 画面宽高
    parameter    DISPLAY_WIDTH  = 10'd640;
    parameter    DISPLAY_HEIGHT = 10'd480;

    // 行场计数
    reg  [10:0] x_cnt;
    reg  [10:0] y_cnt;

    reg         v_reg0, v_reg1, v_reg2, v_reg3;
    reg         h1_reg0, h1_reg1, h1_reg2, h1_reg3;
    reg         h2_reg0, h2_reg1, h2_reg2, h2_reg3;

    // 寄存与割线的交点个数
    reg [3:0]   vcnt;
    reg [3:0]   hcnt1;
    reg [3:0]   hcnt2;
    reg         h1_pst;
    reg         h2_pst;

    // 辅助计算的参考数值
    wire [10:0] fig_width  = line_right - line_left;
    wire [10:0] fig_height = line_bottom - line_top;
    wire [10:0] fig_vdiv   = line_left + fig_width * 8 / 15;
    wire [10:0] fig_hdiv1  = line_top + fig_height * 3 / 10;
    wire [10:0] fig_hdiv2  = line_top + fig_height * 7 / 10;
```

```
// 对行场方向分别计数用于投影
always@(posedge clk or posedge reset)
begin
    if(reset)
        begin
            x_cnt <= 10'd0;
            y_cnt <= 10'd0;
        end
    else
        if(vsync == 0)
        begin
            x_cnt <= 10'd0;
            y_cnt <= 10'd0;
        end
        else if(clken)
        begin
            if(x_cnt < DISPLAY_WIDTH - 1)
            begin
                x_cnt <= x_cnt + 1'b1;
                y_cnt <= y_cnt;
            end
            else
            begin
                x_cnt <= 10'd0;
                y_cnt <= y_cnt + 1'b1;
            end
        end
end

// 寄存竖直中轴割线上的像素
always@(posedge clk or posedge reset)
begin
    if(reset)
    begin
        v_reg0 <= 1'b0;
        v_reg1 <= 1'b0;
        v_reg2 <= 1'b0;
        v_reg3 <= 1'b0;
    end
    // 新的一帧开始 清空计数
    else if(x_cnt == 1 && y_cnt == 1)
    begin
        v_reg0 <= 1'b0;
        v_reg1 <= 1'b0;
        v_reg2 <= 1'b0;
        v_reg3 <= 1'b0;
    end
```

```
        else if(clken)
        begin
            // 新的一帧开始
            if((x_cnt == fig_vdiv) && (y_cnt > line_top) && (y_cnt <
line_bottom))
            begin
                v_reg0 <= v_reg1;
                v_reg1 <= v_reg2;
                v_reg2 <= v_reg3;
                v_reg3 <= bin;
            end
        end
    end


    // 寄存水平第一条割线上的像素
    always@(posedge clk or posedge reset)
    begin
        if(reset)
        begin
            h1_reg0 <= 1'b0;
            h1_reg1 <= 1'b0;
            h1_reg2 <= 1'b0;
            h1_reg3 <= 1'b0;
        end
        // 新的一帧开始 清空计数
        else if(x_cnt == 1 && y_cnt == 1)
        begin
            h1_reg0 <= 1'b0;
            h1_reg1 <= 1'b0;
            h1_reg2 <= 1'b0;
            h1_reg3 <= 1'b0;
        end
        else if(clken)
        begin
            // 新的一帧开始
            if((y_cnt == fig_hdiv1) && (x_cnt > line_left) && (x_cnt <
line_right))
            begin
                h1_reg0 <= h1_reg1;
                h1_reg1 <= h1_reg2;
                h1_reg2 <= h1_reg3;
                h1_reg3 <= bin;
            end
        end
    end

    // 寄存水平第二条割线上的像素
```

```
    always@(posedge clk or posedge reset)
    begin
        if(reset)
        begin
            h2_reg0 <= 1'b0;
            h2_reg1 <= 1'b0;
            h2_reg2 <= 1'b0;
            h2_reg3 <= 1'b0;
        end
        // 新的一帧开始 清空计数
        else if(x_cnt == 1 && y_cnt == 1)
        begin
            h2_reg0 <= 1'b0;
            h2_reg1 <= 1'b0;
            h2_reg2 <= 1'b0;
            h2_reg3 <= 1'b0;
        end
        else if(clken)
        begin
            // 新的一帧开始
            if((y_cnt == fig_hdiv2) && (x_cnt > line_left) && (x_cnt <
line_right))
            begin
                h2_reg0 <= h2_reg1;
                h2_reg1 <= h2_reg2;
                h2_reg2 <= h2_reg3;
                h2_reg3 <= bin;
            end
        end
    end

    // 竖直中轴线交点计数
    always@(posedge clk or posedge reset)
    begin
        if(reset)
            vcnt <= 4'b0;
        // 新的一帧开始
        else if((x_cnt == 1) && (y_cnt == 1))
            vcnt <= 4'b0;
        else if(clken)
        begin
            // 多加几个像素点是以减小干扰
            if(v_reg0 == 0 && v_reg1 == 0 && v_reg2 == 0 && v_reg3 == 1
            && (x_cnt == fig_vdiv)
            && (y_cnt > line_top) && (y_cnt < line_bottom))
            begin
                vcnt <= vcnt + 1;
            end
```

```
        end
    end

    // 水平第一条割线交点计数
    always@(posedge clk or posedge reset)
    begin
        if(reset)
            hcnt1 <= 4'b0;
        // 新的一帧开始
        else if((x_cnt == 1) && (y_cnt == 1))
            hcnt1 <= 4'b0;
        else if(clken)
        begin
           //多加几个像素点是以减小干扰
           if(h1_reg0 == 0 && h1_reg1 == 0 && h1_reg2 == 0 && h1_reg3
== 1
           && (y_cnt == fig_hdiv1)
           && (x_cnt > line_left) && (x_cnt < line_right))
           begin
               hcnt1 <= hcnt1 + 1;
               h1_pst = (x_cnt < fig_vdiv);
           end
        end
    end

    // 水平第二条割线交点计数
    always@(posedge clk or posedge reset)
    begin
        if(reset)
            hcnt2 <= 4'b0;
        // 新的一帧开始
        else if((x_cnt == 1) && (y_cnt == 1))
            hcnt2 <= 4'b0;
        else if(clken)
        begin
           //多加几个像素点是以减小干扰
           if(h2_reg0 == 0 && h2_reg1 == 0 && h2_reg2 == 0 && h2_reg3
== 1
           && (y_cnt == fig_hdiv2)
           && (x_cnt > line_left) && (x_cnt < line_right))
           begin
               hcnt2 <= hcnt2 + 1;
               h2_pst = (x_cnt < fig_vdiv);
           end
        end
    end

    always@(posedge clk or posedge reset)
```

```
    begin
        if(reset)
        begin
            v_cnt  <= 4'b0;
            h_cnt1 <= 4'b0;
            h_cnt2 <= 4'b0;
            h1     <= 1'b0;
            h2     <= 1'b0;
        end
        else if(vsync == 0)
        begin
            v_cnt  <= vcnt;
            h_cnt1 <= hcnt1;
            h_cnt2 <= hcnt2;
            h1     <= h1_pst;
            h2     <= h2_pst;
        end
    end
endmodule
```

4.10 数字判断模块

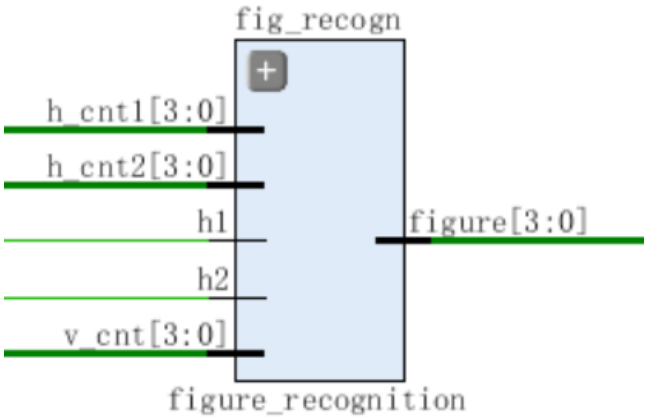

图 15

模块简介：该模块共 4 个，分别对应 4 个数字的判断。即分别通过交点特征提取模块所获得的信息进行数字的判断，并传输给 7 段数码管模块进行显示。

```
module figure_recognition(
    input [3:0]          v_cnt,    // 数字与竖直中轴线交点个数
    input [3:0]          h_cnt1,   // 数字与水平分割线 1 交点个数
    input [3:0]          h_cnt2,   // 数字与水平分割线 2 交点个数
    input                h1,       // 最后与水平线 1 的交点在中轴线左还是右
    input                h2,       // 最后与水平线 2 的交点在中轴线左还是右
    output reg [3:0]     figure     // 判断出的数字
```

```
    );
    always @ (*)
    begin
        case({ v_cnt, h_cnt1, h_cnt2 })
            {4'd2, 4'd2, 4'd2}: figure <= 4'd0;
            {4'd1, 4'd1, 4'd1}: figure <= 4'd1;
            {4'd2, 4'd2, 4'd1}: figure <= 4'd4;
            {4'd3, 4'd1, 4'd2}:
            begin
                case(h1)
                    1'b0:       figure <= 4'd3;
                    1'b1:       figure <= 4'd6;
                endcase
            end
            {4'd2, 4'd1, 4'd1}: figure <= 4'd7;
            {4'd3, 4'd2, 4'd2}: figure <= 4'd8;
            {4'd3, 4'd2, 4'd1}: figure <= 4'd9;
            {4'd3, 4'd1, 4'd1}:
            begin
                case({ h1, h2 })
                    2'b01:      figure <= 4'd2;
                    2'b00:      figure <= 4'd3;
                    2'b10:      figure <= 4'd5;
                    default:    figure <= 4'hF;
                endcase
            end
            default:            figure <= 4'hF;
        endcase
    end
endmodule
```

4.11 7 段数码管模块

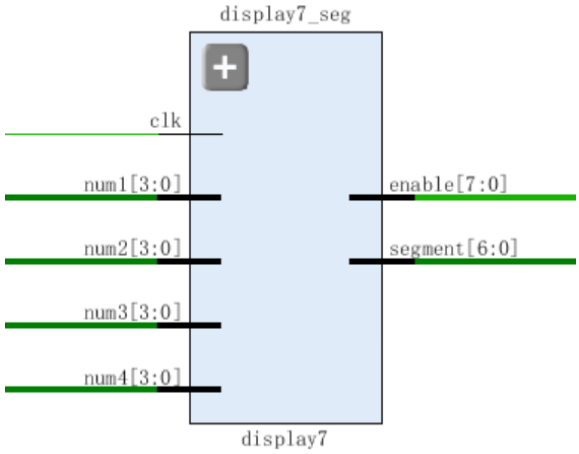

图 16

模块简介：该模块负责将识别出的最多 4 位数字在 7 段数码管上进行显示。查阅开发板官方手册可了解到 7 段数码管显示原理；本实验 8 位数字中的高四位默认不做显示，低四位在时钟周期中轮流闪烁，通过视觉暂留效应达到 4 位不同数字的同时显示。若某位数字不存在或识别不出则不显示。

```
module display7(
    input               clk,        // 时钟
    input       [3:0]   num1,       // 右侧 4 位数字显示
    input       [3:0]   num2,       // 从左到右为 num1 ~ num4
    input       [3:0]   num3,       // 非 0~9 则不亮灯
    input       [3:0]   num4,
    output      [7:0]   enable,     // 八个数字的使能
    output reg  [6:0]   segment     // 7 段数码管
    );

    // 寄存器定义
    reg [16:0]  cnt = 0;        // 计数器 用于分频
    reg [3:0]   reg_ena;        // 控制当前点亮的数码管
    reg [3:0]   num_current;    // 当前显示的数字

    // 左侧四位数字不亮
    assign enable = { 4'b1111, reg_ena };

    always @(posedge clk)
    begin
        cnt = cnt + 1;      // 计数
        // 分频 100MHz 无法正常显示
        case(cnt[16:15])
            2'b00:
            begin
                reg_ena     <= 4'b0111;
                num_current <= num1;
            end
            2'b01:
            begin
                reg_ena     <= 4'b1011;
                num_current <= num2;
            end
            2'b10:
            begin
                reg_ena     <= 4'b1101;
                num_current <= num3;
            end
            2'b11:
            begin
                reg_ena     <= 4'b1110;
```

```
                num_current <= num4;
            end
        endcase
    end

    always @(*)
    begin
        case(num_current)
            4'd0    : segment <= 7'b1000000;
            4'd1    : segment <= 7'b1111001;
            4'd2    : segment <= 7'b0100100;
            4'd3    : segment <= 7'b0110000;
            4'd4    : segment <= 7'b0011001;
            4'd5    : segment <= 7'b0010010;
            4'd6    : segment <= 7'b0000010;
            4'd7    : segment <= 7'b1111000;
            4'd8    : segment <= 7'b0000000;
            4'd9    : segment <= 7'b0010000;
            default : segment <= 7'b1111111;
        endcase
    end

endmodule
```

4.12 VGA 驱动模块

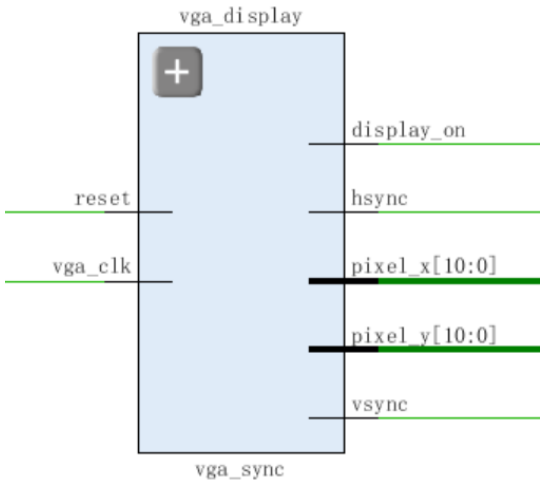

图 17

模块简介：该模块负责以 640*480 的分辨率输出 VGA 时序逻辑以供屏幕显示。VGA 时序参数等查阅相关资料得，具体在代码段参数部分列出。该模块以 25.175MHz 的时钟输入；一行像素输出后拉高行同步信号以同步数据，场时序

同理。在有效像素输出时拉高 display_on 信号以显示像素信息。pixel_x 与 pixel_y 代表此时的像素位置。

```
module vga_sync(
    input vga_clk,                  // VGA 时钟
    input reset,                    // 复位信号

    output hsync,                   // 行同步信号
    output vsync,                   // 场同步信号

    output display_on,              // 是否显示输出
    output [10:0] pixel_x,          // 当前像素点横坐标
    output [10:0] pixel_y           // 当前像素点纵坐标
);

    // VGA 同步参数
    parameter H_DISPLAY       = 640;    // 行有效显示
    parameter H_L_BORDER      =  48;    // 行左边框
    parameter H_R_BORDER      =  16;    // 行右边框
    parameter H_SYNC          =  96;    // 行同步
    parameter H_MAX           = H_DISPLAY + H_L_BORDER + H_R_BORDER +
H_SYNC - 1;

    parameter V_DISPLAY       = 480;    // 列有效显示
    parameter V_T_BORDER      =  33;    // 列上边框
    parameter V_B_BORDER      =  10;    // 列下边框
    parameter V_SYNC          =   2;    // 列同步
    parameter V_MAX           = V_DISPLAY + V_T_BORDER + V_B_BORDER +
V_SYNC - 1;

    // VGA 行场同步信号
    assign hsync = (h_cnt < H_SYNC) ? 1'b0 : 1'b1;
    assign vsync = (v_cnt < V_SYNC) ? 1'b0 : 1'b1;

    // 是否显示输出
    assign display_on = (h_cnt >= H_SYNC + H_L_BORDER) && (h_cnt <
H_SYNC + H_L_BORDER + H_DISPLAY)
                        && (v_cnt >= V_SYNC + V_T_BORDER) && (v_cnt <
V_SYNC + V_T_BORDER + V_DISPLAY);

    // 横纵坐标计算
    reg [10:0] h_cnt;
    reg [10:0] v_cnt;
    assign pixel_x = h_cnt - H_SYNC - H_L_BORDER;
    assign pixel_y = v_cnt - V_SYNC - V_T_BORDER;

    always @ (posedge vga_clk)
    begin
```

```
        if(reset)
        begin
            h_cnt <= 0;
            v_cnt <= 0;
        end
        else
        begin
            if(h_cnt == H_MAX)
            begin
                h_cnt <= 0;
                if(v_cnt == V_MAX)
                    v_cnt <= 0;
                else
                    v_cnt <= v_cnt + 1;
            end
            else
                h_cnt <= h_cnt + 1;
        end
    end

endmodule
```

4.13 VGA 框线显示模块

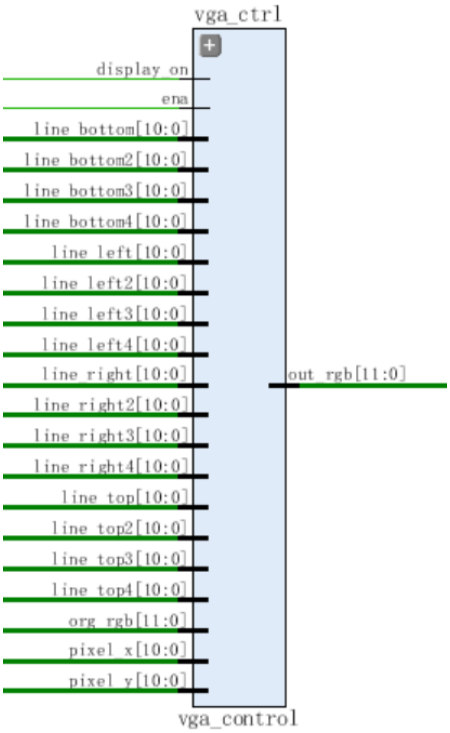

图 18

模块简介：在框线显示模式下，该模块负责显示用红色线段标注出数字的边缘框线，并用蓝色线段标注出数字的切割辅助线以便于 VGA 显示观察。

```
module vga_control(
    input       [11:0]    org_rgb,           // 原始 rgb 像素值
    input                 display_on,        // 是否显示
    input       [10:0]    pixel_x,           // 当前输出像素的横纵坐标
    input       [10:0]    pixel_y,
    input                 ena,               // 是否显示框线
    input       [10:0]    line_left,         // 各个数字的框线
    input       [10:0]    line_right,
    input       [10:0]    line_top,
    input       [10:0]    line_bottom,
    input       [10:0]    line_left2,
    input       [10:0]    line_right2,
    input       [10:0]    line_top2,
    input       [10:0]    line_bottom2,
    input       [10:0]    line_left3,
    input       [10:0]    line_right3,
    input       [10:0]    line_top3,
    input       [10:0]    line_bottom3,
    input       [10:0]    line_left4,
    input       [10:0]    line_right4,
    input       [10:0]    line_top4,
    input       [10:0]    line_bottom4,
    output reg  [11:0]    out_rgb
    );
    // 数字辅助线
    wire [10:0] fig_width  = line_right - line_left;
    wire [10:0] fig_height = line_bottom - line_top;
    wire [10:0] fig_vdiv   = line_left + fig_width * 8 / 15;
    wire [10:0] fig_hdiv1  = line_top + fig_height * 3 / 10;
    wire [10:0] fig_hdiv2  = line_top + fig_height * 7 / 10;

    wire [10:0] fig_width_2  = line_right2 - line_left2;
    wire [10:0] fig_height_2 = line_bottom2 - line_top2;
    wire [10:0] fig_vdiv_2   = line_left2 + fig_width_2 * 8 / 15;
    wire [10:0] fig_hdiv1_2  = line_top2 + fig_height_2 * 3 / 10;
    wire [10:0] fig_hdiv2_2  = line_top2 + fig_height_2 * 7 / 10;

    wire [10:0] fig_width_3  = line_right3 - line_left3;
    wire [10:0] fig_height_3 = line_bottom3 - line_top3;
    wire [10:0] fig_vdiv_3   = line_left3 + fig_width_3 * 8 / 15;
    wire [10:0] fig_hdiv1_3  = line_top3 + fig_height_3 * 3 / 10;
    wire [10:0] fig_hdiv2_3  = line_top3 + fig_height_3 * 7 / 10;

    wire [10:0] fig_width_4  = line_right4 - line_left4;
    wire [10:0] fig_height_4 = line_bottom4 - line_top4;
```

```
    wire [10:0] fig_vdiv_4   = line_left4 + fig_width_4 * 8 / 15;
    wire [10:0] fig_hdiv1_4  = line_top4 + fig_height_4 * 3 / 10;
    wire [10:0] fig_hdiv2_4  = line_top4 + fig_height_4 * 7 / 10;


    always @ (*)
    begin
        if(display_on)
        begin
            if(ena)
            begin
                if((pixel_x == line_left || pixel_x == line_right)
                    && (pixel_y > line_top)
                    && (pixel_y < line_bottom))
                    out_rgb <= 12'hF00;
                else if((pixel_y == line_top || pixel_y == line_bottom)
                        && (pixel_x > line_left)
                        && (pixel_x < line_right))
                    out_rgb <= 12'hF00;
                else if((pixel_x == fig_vdiv)
                        && (pixel_y > line_top)
                        && (pixel_y < line_bottom))
                    out_rgb <= 12'h00F;
                else if((pixel_y == fig_hdiv1 || pixel_y == fig_hdiv2)
                        && (pixel_x > line_left)
                        && (pixel_x < line_right))
                    out_rgb <= 12'h00F;
                else if((pixel_x == line_left2 || pixel_x ==
line_right2)
                    && (pixel_y > line_top2)
                    && (pixel_y < line_bottom2))
                    out_rgb <= 12'hF00;
                else if((pixel_y == line_top2 || pixel_y ==
line_bottom2)
                        && (pixel_x > line_left2)
                        && (pixel_x < line_right2))
                    out_rgb <= 12'hF00;
                else if((pixel_x == fig_vdiv_2)
                        && (pixel_y > line_top2)
                        && (pixel_y < line_bottom2))
                    out_rgb <= 12'h00F;
                else if((pixel_y == fig_hdiv1_2 || pixel_y ==
fig_hdiv2_2)
                        && (pixel_x > line_left2)
                        && (pixel_x < line_right2))
                    out_rgb <= 12'h00F;
                else if((pixel_x == line_left3 || pixel_x ==
line_right3)
```

```
                        && (pixel_y > line_top3)
                        && (pixel_y < line_bottom3))
                    out_rgb <= 12'hF00;
                else if((pixel_y == line_top3 || pixel_y ==
line_bottom3)
                        && (pixel_x > line_left3)
                        && (pixel_x < line_right3))
                    out_rgb <= 12'hF00;
                else if((pixel_x == fig_vdiv_3)
                        && (pixel_y > line_top3)
                        && (pixel_y < line_bottom3))
                    out_rgb <= 12'h00F;
                else if((pixel_y == fig_hdiv1_3 || pixel_y ==
fig_hdiv2_3)
                        && (pixel_x > line_left3)
                        && (pixel_x < line_right3))
                    out_rgb <= 12'h00F;
                else if((pixel_x == line_left4 || pixel_x ==
line_right4)
                        && (pixel_y > line_top4)
                        && (pixel_y < line_bottom4))
                    out_rgb <= 12'hF00;
                else if((pixel_y == line_top4 || pixel_y ==
line_bottom4)
                        && (pixel_x > line_left4)
                        && (pixel_x < line_right4))
                    out_rgb <= 12'hF00;
                else if((pixel_x == fig_vdiv_4)
                        && (pixel_y > line_top4)
                        && (pixel_y < line_bottom4))
                    out_rgb <= 12'h00F;
                else if((pixel_y == fig_hdiv1_4 || pixel_y ==
fig_hdiv2_4)
                        && (pixel_x > line_left4)
                        && (pixel_x < line_right4))
                    out_rgb <= 12'h00F;
                else
                    out_rgb <= org_rgb;
            end
            else
                out_rgb <= org_rgb;
        end
        else
            out_rgb <= 12'b0;
    end
endmodule
```

五、测试模块建模

5.1 时钟分频模块测试

测试模块代码：

```
`timescale 1ns / 1ps

// 测试时钟分频模块
module clk_div_tb(
    );

    reg clk_in;
    wire clk_vga;
    wire clk_sccb;

    initial
    begin
        clk_in = 0;
        forever #5 clk_in = ~clk_in;
    end

    clk_divider clk_div(
    .clk_in1(clk_in),
    .clk_vga(clk_vga),
    .clk_sccb(clk_sccb)
    );

endmodule
```

仿真波形：

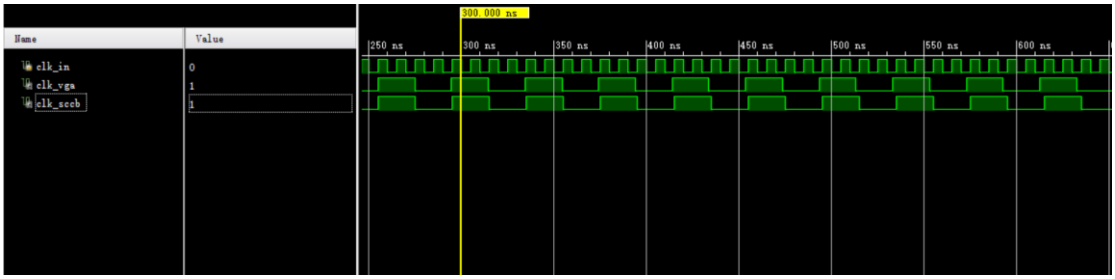

图 19

5.2 7 段数码管模块测试

测试模块代码：

```
`timescale 1ns / 1ps

// 7 段数码管测试模块
module display7_tb(
    );

    reg clk;
    wire [7:0] ena;
    wire [6:0] segment;

    initial
    begin
        clk = 0;
        forever #1 clk = ~clk;
    end

    display7 dis7(
        .clk,
        .num1(4'h7),
        .num2(4'h3),
        .num3(4'hF),
        .num4(4'h0),
        .enable(ena),
        .segment(segment)
   );

endmodule
```

仿真波形：

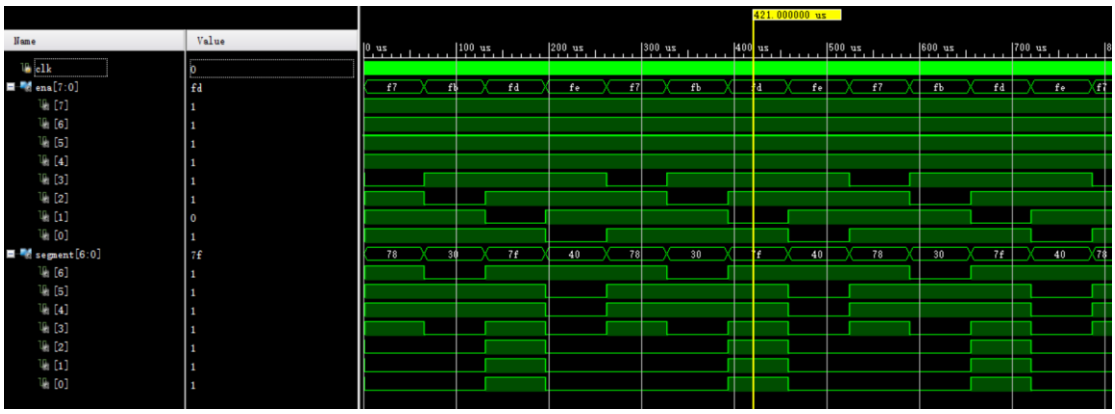

图 20

5.3 VGA 模块测试

由于 VGA 模块的特殊性，直接下板测试相较撰写 testbench 文件更为直观便捷。测试 VGA 模块时，我将 bmp 格式图片转换为 coe 文件用于初始化 ROM，从而使屏幕显示图片以测试 VGA 模块的正确性（如图 21、图 22 所示）。

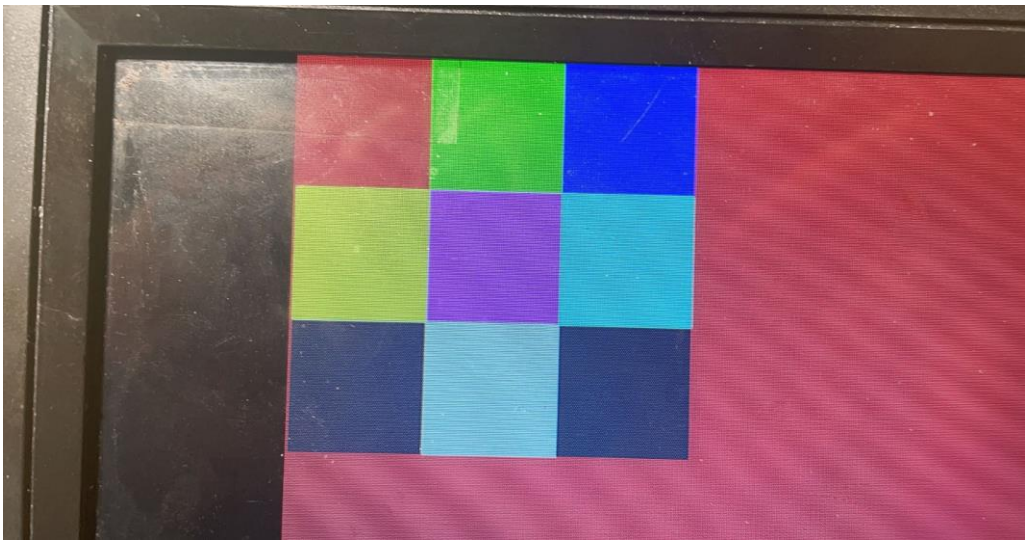

图 21

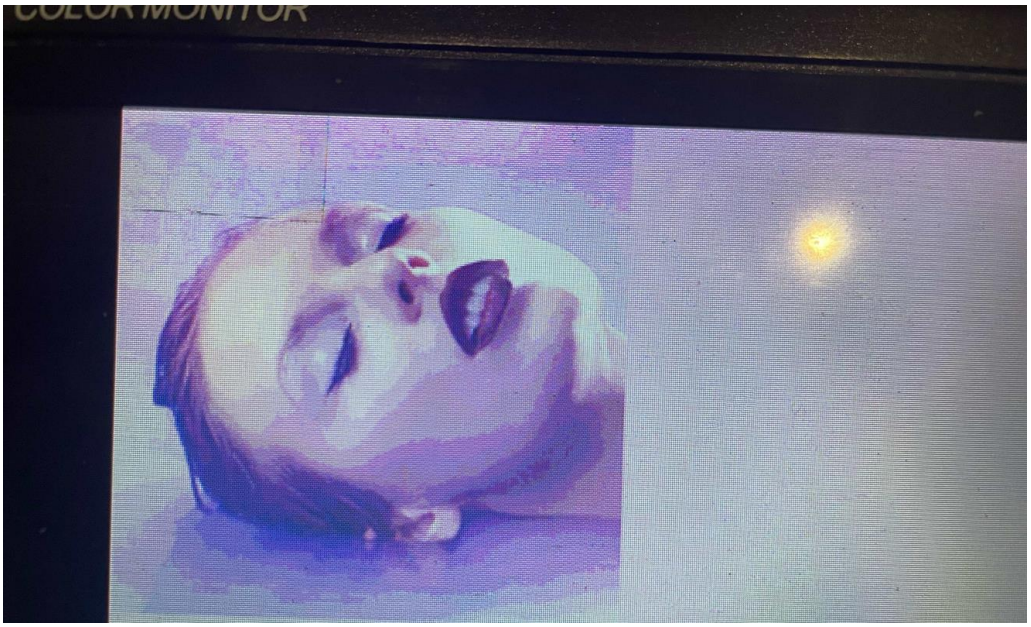

图 22

六、实验结果

6.1 单个数字识别

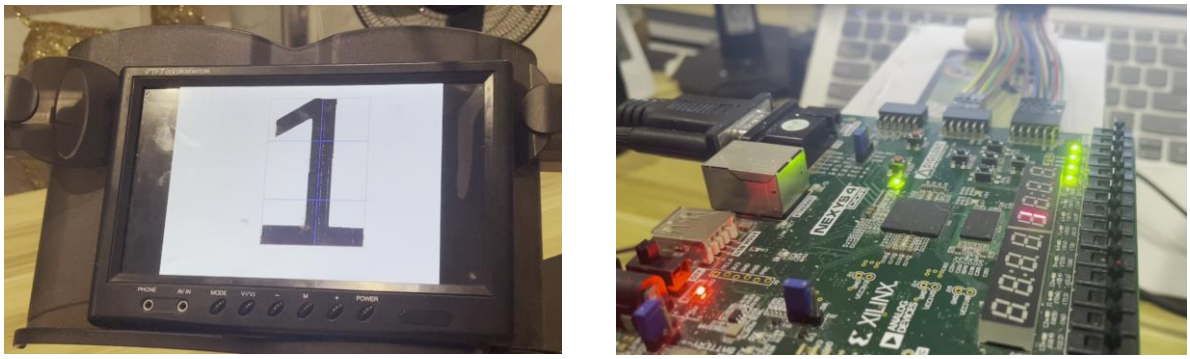

图 23

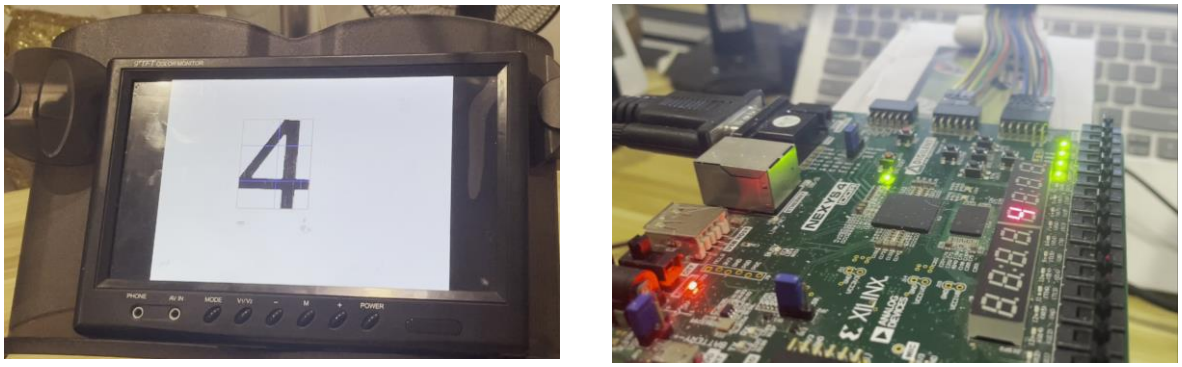

图 24

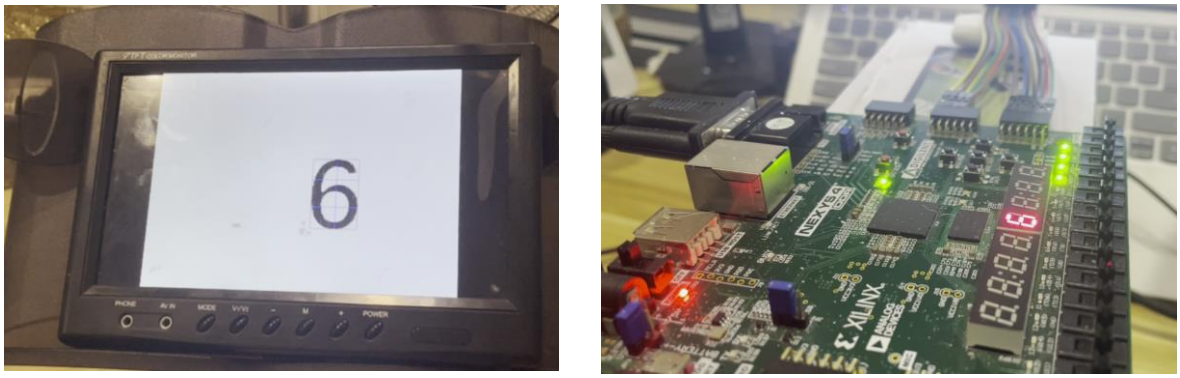

图 25

6.2 多个数字识别

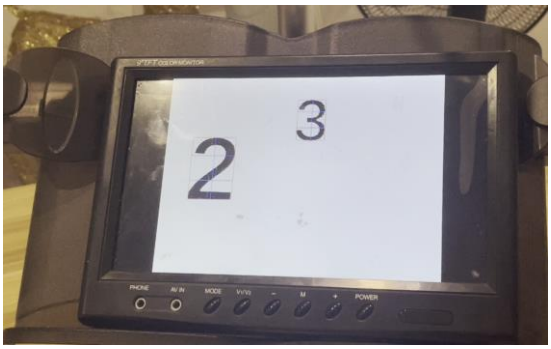
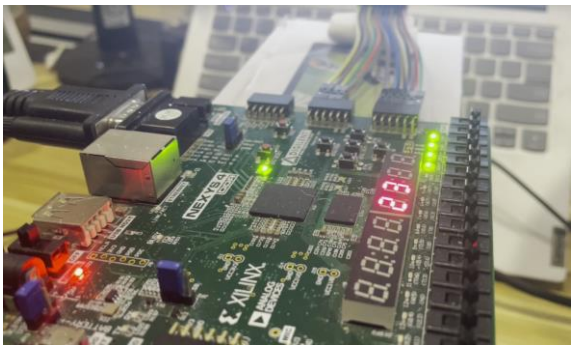

图 26

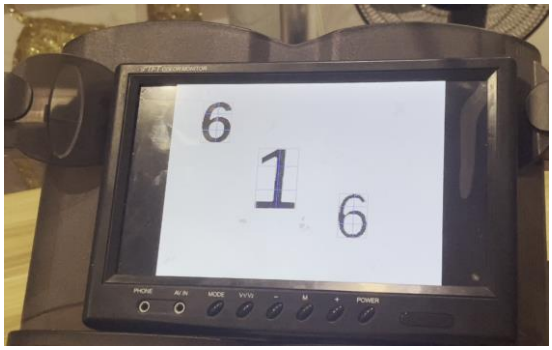
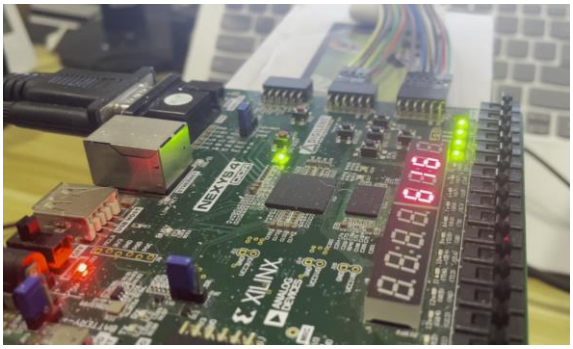

图 27

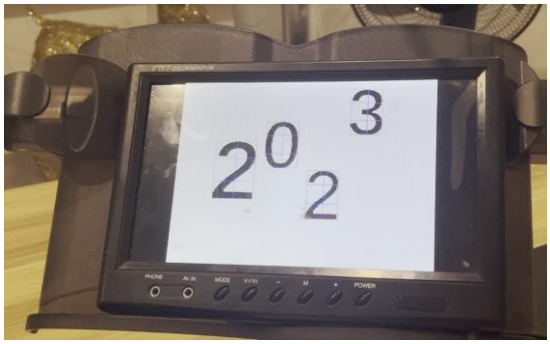
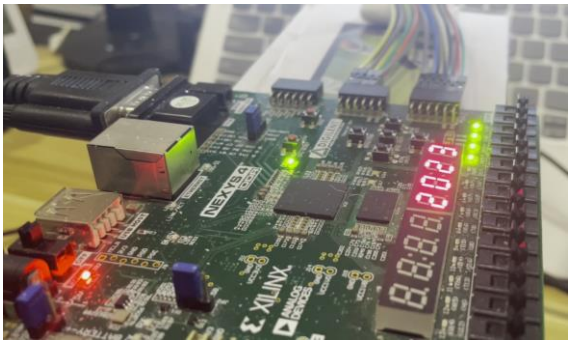

图 28

七、结论

各个子模块均能够按照预期协调正常运作，数字系统工作正常。在摄像头晃动幅度较小的稳定状态下，对于单个、两个、三个、四个印刷体数字的定位基本准确无误，数字判别正确率较高且能够成功进行显示。

八、心得体会及建议

8.1 本课程收获

在为期一学期的数字逻辑理论课与实验课学习过程中，我从对数字逻辑相关知识知之甚少到能够顺利完成比较完整的大作业项目，可以说是收获颇丰。在不断的学习与实践过程中，我认为自己能够比较认真地学习数字逻辑的知识，也端正地对待每一次的实验课程与课后作业。通过一学期以来的实践学习，我逐渐提升了自己对 verilog 语言的编程能力，也不断在仿真、下板过程中提高自己的调试能力。知识点方面，从时序逻辑、组合逻辑再到数字系统的设计，我在循序渐进的学习中逐步掌握了数字逻辑相关的许多知识，也为大作业的完成奠定了基础。对于一些课内没有涉及的知识，希望自己在未来的学习过程中能够更有自主学习的意识与动力。

在开始着手进行大作业的起初，尽管有实验课的一些基础，但我对各个外设的连接与使用可以说是一窍不通。在开始构架大作业的项目之前，我先逐一学习了各个外设的使用方式，查看了相关的各种文档与手册，也阅读了老师们所推荐的相关资料。首先我学习了 VGA 显示的时序逻辑，查阅了相关的资料；在确定了 640x480 的分辨率之后，我通过 ROM 初始化的方式显示图片来调试该模块。对于 HC-06 蓝牙模块，我了解了 uart 协议并了解了通过打拍的方式来消除亚稳态，也在相关代码的撰写过程中通过 LED 灯的明灭进行调试。OV2640 摄像头的初始化与使用是外设中难度最大的，在此过程中我既查阅了 OV2640 的软件官方文档，也参考了学长的摄像头相关代码。其中既涉及到 sccb 协议传输的初始化寄存器信息，也涉及到将摄像头传入图像的 RGB565 整合成 RGB444 像素的写入 RAM。学习外设相关知识后，我自顶向下进行了各个子系统的划分，并分别对各个模块进行代码撰写与调试。尽管遭遇了不少困难（如对数字左右框线的判断不稳定、7 段数码管无法正常显示等情况），但它们最终都在调试后被逐个攻破。总的来说，本次数字逻辑综合实验大作业既是对我数字逻辑能力很好的检验过程，也是对我自学能力的一次锻炼。

8.2 对本课程的建议

在本学期数字逻辑的课程当中，重点以开关理论基础、组合逻辑、时序逻辑、数字系统为脉络学习了课本相关章节的知识。我认为在课程安排上，可以适当减少第一章节的教学时长。因为计算机科学导论、离散数学等课程在教学内容上都与该章节有所重合，所以我认为重点讲解卡诺图等部分即可。在数字系统方面的教学由于临近期末周较为仓促，我认为可以适当增加这一部分的教学时间，让同学们更深刻地了解到数字系统设计的方法和思想，对综合性大作业也会有所帮助。除此之外，我觉得对大作业外设的介绍也可以在实验课之中穿插进行。这样同学们能够了解到不同外设不同协议与开发板的数据传输方式，在期末开始综合作业的设计与实施时也能够更为得心应手。

8.3 对数字芯片设计国内外现状的认识与体会

揆诸当下，我国在半导体产业仍与世界顶尖水平有着不得不正视的差距。从光刻胶到 EDA 软件，还要许多技术壁垒亟待打破、工程难关等待翻越。以 FPGA 芯片为例，其市场的 95%都被美国公司所占据，尤其是中高端的 FPGA 芯片，国产替代品的研发与制造仍然苦难重重。在许多关键敏感的领域，自主可控的国产数字芯片对于国家安全的维护至关重要，因此数字芯片领域的技术攻关与研发制造任重而道远。回顾我国科技数十年来的发展史，许多领域的科研已然从跟跑者成为领跑与开拓者。时代的接力棒已然交到当代青年的手中，芯片领域的人才培养与技术突破还需要我们的学习钻研。相信在未来半导体领域的发展过程中，也必然能够突破“卡脖子”的技术限制，实现弯道超车。