同济大学计算机系

数字逻辑课程综合实验报告

学　　号 **1751740**

姓　　名 刘鲲

专　　业 计算机科学与技术

授课老师 张冬冬

一、实验内容

基于 FPGA 的 Uart 发送彩色图像并图像处理。

图像处理包括：显示原图，彩色转灰色，中值滤波，Sobel 边缘检测，腐蚀，膨胀。

使用方法：

1.准备好一张 RGB565 的图片并将其转成 16 进制格式、一个串口发送软件、FPGA、MicroUSB 线

2.加载 bit 流，打开串口发送软件，将图片的 16 进制数据发送到 USB 转串口去

3.将开关 J15 打开，可见原图缓缓出现

4.不同的开关组合对应不同的图像处理显示

令 S = <M13><L16><J15>，令 Si=1 为开关向上拨，则

S 的值	功能
001	显示原图
010	原图的灰度图
011	中值滤波
100	边缘检测
101	腐蚀
110	膨胀
其它组合	黑屏

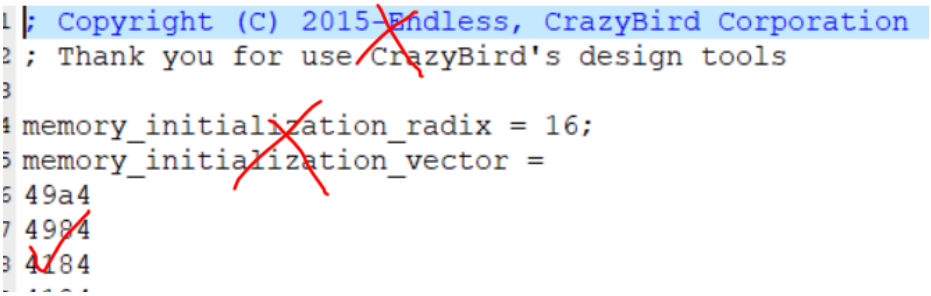
注：串口发送软件有两款：

根据笔者经验，UartAssist 虽然界面粗糙，但是不容易卡，而且能支持比左者更高的波特率。以右款为例，说明使用步骤。

1.串口号是与板子相连的 USB 端口，一般为 COM3 或 COM4，确认波特率为 9600（代码可调），为方便数据拼接，不采用校验位，一位停止位，8 位数据位。
2. 启用文件数据源，将转换好的 RGB565 图像数据的文件导入，注意要按十六进制。
3. 打开
4. 发送

注：获得一张图片的 RGB565 编码
1. 使用 BMP2MiF 将 bmp 格式的图片转为 RGB565，格式建议为. coe

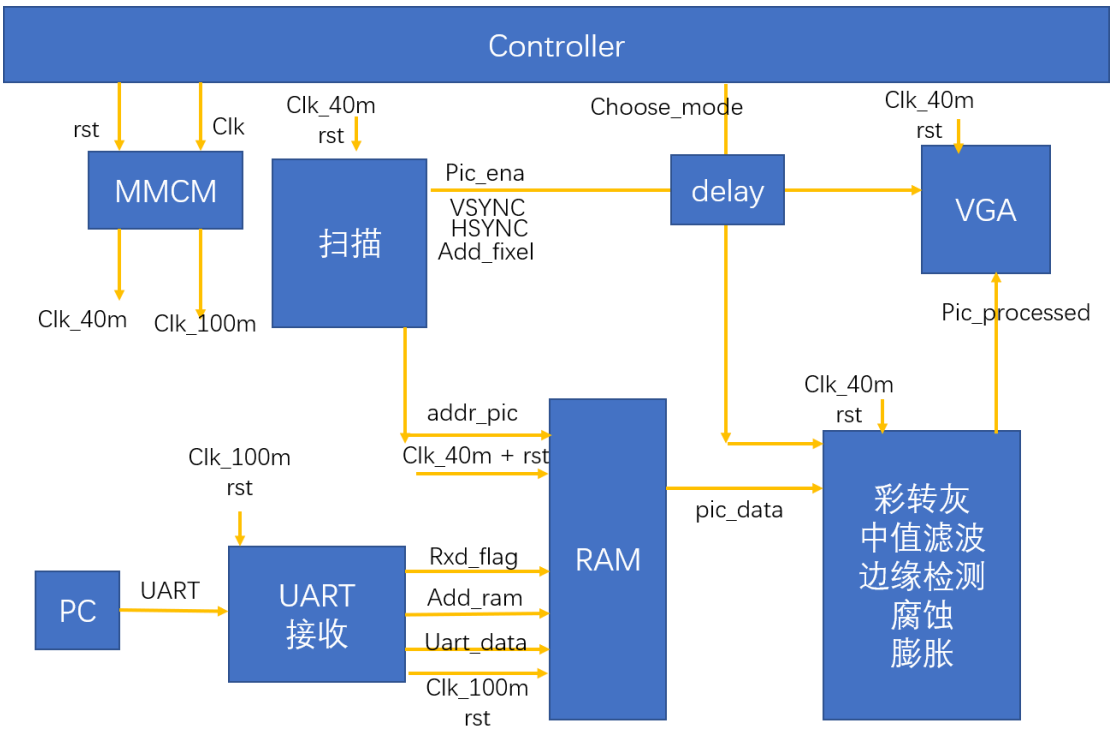
2. 打开 coe，删除所有除数据以外的所有信息，包括\r\n！可使用文本编辑器的全文替换功能，但是数据量较大时会卡死



注：在早期设计中，图像是放在 IP core ROM 里的。具体操作为将 bmp 图像用上述 BMP2Mif 转成.coe 文件，并设置好同大小的 IP core ROM 即可

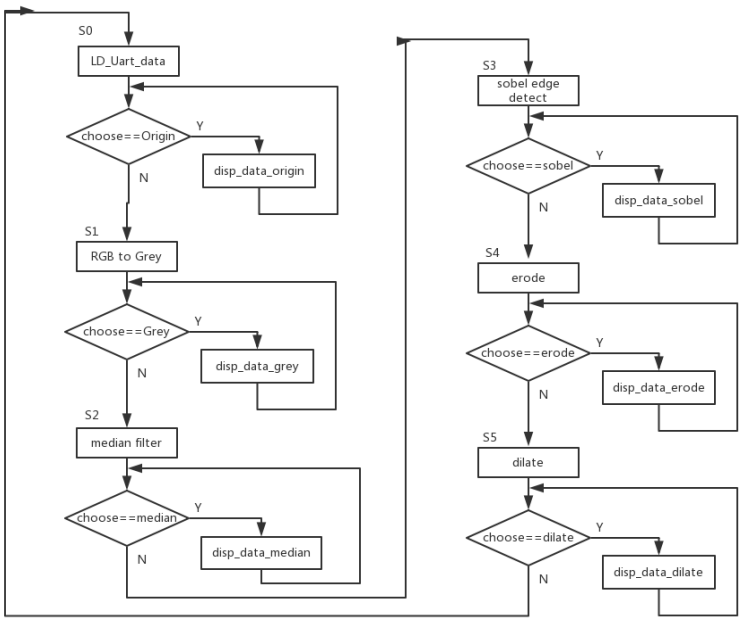
二、图像处理数字系统总框图

（按由顶向下方法进行子系统的划分，给出包含各子系统相互关系及控制信号的总框图，并对各子系统功能及实现进行概述。具体可参考教材 183 页的相关描述方法。）



三、系统控制器设计

（要求画出所设计数字系统的 ASM 流程图，列出状态转移真值表。由状态转移真值表，求出系统控制器的次态激励函数表达式和控制命令逻辑表达式，并用 Logisim 画出系统控制器逻辑方案图。）



注：由于实际分支的 ASM 过于复杂，故取最为完整的一条 ASM 路径。详细的状态转移见下方真值表。采用格雷码，有输入 choose[4:0]。

	PS					NS					choose[4:0]
	Qa	Qb	Qc	Qd	Qe	Qa(D)	Qb(D)	Qc(D)	Qd(D)	Qe(D)	
S0	0	0	0	0	0	0	0	0	0	1	S1
	0	0	0	0	0	0	0	0	1	0	S2
	0	0	0	0	0	0	0	1	0	0	S3
	0	0	0	0	0	0	1	0	0	0	S4
	0	0	0	0	0	1	0	0	0	0	S5
S1	0	0	0	0	1	0	0	0	0	1	S1
	0	0	0	0	1	0	0	0	1	0	S2
	0	0	0	0	1	0	0	1	0	0	S3
	0	0	0	0	1	0	1	0	0	0	S4
	0	0	0	0	1	1	0	0	0	0	S5
S2	0	0	0	1	0	0	0	0	0	1	S1
	0	0	0	1	0	0	0	0	1	0	S2
	0	0	0	1	0	0	0	1	0	0	S3
	0	0	0	1	0	0	1	0	0	0	S4
	0	0	0	1	0	1	0	0	0	0	S5
S3	0	0	1	0	0	0	0	0	0	1	S1
	0	0	1	0	0	0	0	0	1	0	S2
	0	0	1	0	0	0	0	1	0	0	S3
	0	0	1	0	0	0	1	0	0	0	S4
	0	0	1	0	0	1	0	0	0	0	S5
S4	0	1	0	0	0	0	0	0	0	1	S1
	0	1	0	0	0	0	0	0	1	0	S2
	0	1	0	0	0	0	0	1	0	0	S3
	0	1	0	0	0	0	1	0	0	0	S4
	0	1	0	0	0	1	0	0	0	0	S5
S5	1	0	0	0	0	0	0	0	0	1	S1
	1	0	0	0	0	0	0	0	1	0	S2
	1	0	0	0	0	0	0	1	0	0	S3
	1	0	0	0	0	0	1	0	0	0	S4
	1	0	0	0	0	1	0	0	0	0	S5

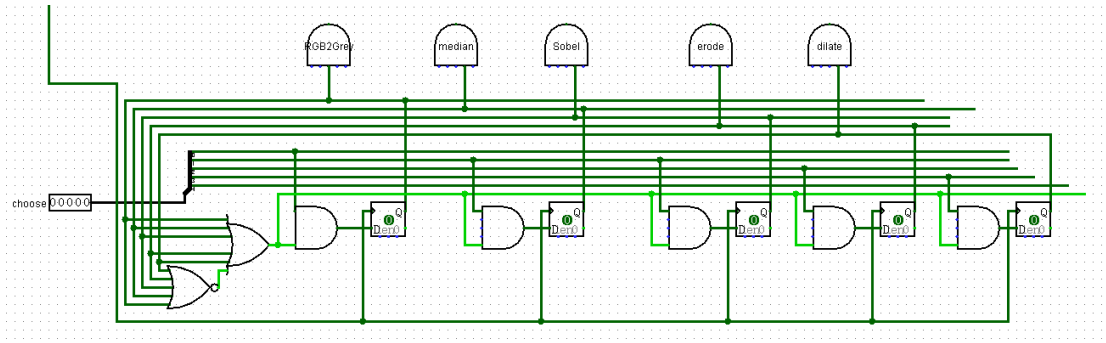
$Qa = S5 * (\overline{Qa + Qb + Qc + Qd + Qe} + Qa + Qb + Qc + Qd + Qe)$

$Qb = S4 * (\overline{Qa + Qb + Qc + Qd + Qe} + Qa + Qb + Qc + Qd + Qe)$

$Qc = S3 * (\overline{Qa + Qb + Qc + Qd + Qe} + Qa + Qb + Qc + Qd + Qe)$

$Qd = S2 * (\overline{Qa + Qb + Qc + Qd + Qe} + Qa + Qb + Qc + Qd + Qe)$

$Qe = S1 * (\overline{Qa + Qb + Qc + Qd + Qe} + Qa + Qb + Qc + Qd + Qe)$



四、子系统模块建模

（该部分要求对实验中的所有子系统模块进行描述，给出各子系统的功能框图及接口信号定义，并列出各模块建模的 verilog 代码）

4.1. 彩色转灰色

描述：将 RGB565 转换成 Grey8。

```
module RGB2Grey(                    input ena,
    input clk,                      input vs,
```

```
    input hs,
    input rst,
    input [15:0]RGB_data,
    output [7:0]Y,
    output post_vs,
    output post_hs,
    output post_ena
   );
    wire [7:0] R0, B0, G0;
    reg [15:0] Y0;//big enough
    //先从 RGB565 to RGB888
    //原因：没有直接从 RGB565 到 YCrCb 的算法

    //量化补偿
    assign R0 = {RGB_data[15:11],RGB_data[15:13]};
   assign G0 = {RGB_data[10:5],RGB_data[10:9]};
   assign B0 = {RGB_data[4:0],RGB_data[4:2]};

    //RGB888 with YCrCb
    reg [15:0] R,G,B; // have to be big enough
    always@(posedge clk or posedge rst)
    begin
        if(rst)
        begin
            R <= 0;
            G <= 0;
            B <= 0;
        end
        else
        begin
        //Verilog 没有浮点数运算，都乘整数，再取高位
            R <= R0 * 77;
            G <= G0 * 150;
            B <= B0 * 29;
        end
    end

    //get grey
    always@(posedge clk or posedge rst)
    begin
        if(rst)
        begin
            Y0 <= 0;
        end
        else
        begin
            Y0 <= R + G + B;
        end
    end


    //取高位
    assign Y = Y0[15:8];

    //计算一位的 Y 需要两个 clk 所以需要两个延迟
    delay_nclk #(.N(2))
    U_delay_2clk (
        .clk(clk),
        .rst(rst),
        .ena(ena),
        .vs(vs),
        .hs(hs),
        .post_vs(post_vs),
        .post_hs(post_hs),
        .post_ena(post_ena)
    );


endmodule
```

4.2. 中值滤波

描述：使用 IP core：shift_ram3x3 中值滤波，消除噪音，使图像平滑，利于边缘提取。

```
module median_filter(
    input clk,
    input rst,
    input vs,
    input hs,
    input [7:0]grey_data,
    input pic_ena,
    output [7:0]median_filter_data,
    output post_vs,
    output post_hs,
```

```
    output post_pic_ena
    );

wire [7:0]row1,row2;
reg   [7:0]row3;

//只用两个 shift ram 级联
//vivado 只有单行 shift ram
    shift_ram_3x3_8bits U_shift_ram1 (
        .D(row3),           // input wire [7 : 0] D
        .CLK(clk),      // input wire CLK //这里 debug 了好久好久!!!!
        .CE(pic_ena),       // input wire CE  //  改 成带使能的 时序报告的 slack 就过了!!!
        .SCLR(rst),   // input wire SCLR
        .Q(row2)            // output wire [7 : 0] Q
      );

    shift_ram_3x3_8bits U_shift_ram0 (
        .D(row2),           // input wire [7 : 0] D
        .CLK(clk),      // input wire CLK
        .CE(pic_ena),       // input wire CE
        .SCLR(rst),   // input wire SCLR
        .Q(row1)            // output wire [7 : 0] Q
        );
    //  数据进入 shift ram  消耗 1 时钟

    always@(posedge clk or posedge rst)
    begin
        if(rst)
            row3 <= 0;
        else if(pic_ena)
            row3 <= grey_data;
        else
            row3 <= row3;
    end

    //因为消耗了 1 个时钟，所以同步信号要延迟
    //延迟方法如下：
    //log 2 clocks signal sync
    reg [1:0] pic_ena_r1;
    reg [1:0] vs_r1,hs_r1;
    always@(posedge clk or posedge rst)
    begin
        if(rst)
        begin
            pic_ena_r1 <= 0;
            vs_r1 <= 0;
            hs_r1 <= 0;
        end
        else
        begin
            pic_ena_r1 <= {pic_ena_r1[0],pic_ena};
            vs_r1 <= {vs_r1[0],vs};
            hs_r1 <= {hs_r1[0],hs};
        end
end
wire read_matrix_ena = pic_ena_r1[0];//log 1
assign read_matrix_hs = hs_r1[0];
assign read_matrix_vs = vs_r1[0];

//根据行、场同步信号读取 3X3 阵列
//matrix_p_x_y
reg [7:0] mp11,mp12,mp13;
reg [7:0] mp21,mp22,mp23;
reg [7:0] mp31,mp32,mp33;

always@(posedge clk or posedge rst)
begin
    if(rst)
    begin
        {mp11,mp12,mp13}<=0;
        {mp21,mp22,mp23}<=0;
        {mp31,mp32,mp33}<=0;
    end
        else if(read_matrix_ena)
        begin

{mp11,mp12,mp13}<={mp12,mp13,row1};

{mp21,mp22,mp23}<={mp22,mp23,row2};

{mp31,mp32,mp33}<={mp32,mp33,row3};
        end
        else
        begin

{mp11,mp12,mp13}<={mp11,mp12,mp13};
```

```
{mp21,mp22,mp23}<={mp21,mp22,mp23};

{mp31,mp32,mp33}<={mp31,mp32,mp33};
                end
end

wire matrix_frame_vs;
wire matrix_frame_hs;
wire matrix_frame_ena;
assign matrix_frame_vs = vs_r1[1];
assign matrix_frame_hs = hs_r1[1];
assign matrix_frame_ena = pic_ena_r1[1];

//对读取的阵列作中值滤波运算
//算法详见：
wire [7:0]max [2:0];
wire [7:0]min [2:0];
wire [7:0]mid [2:0];

//step1:每一行排序  1clk
compare U_row1(
        .clk(clk),
        .rst(rst),
        .data_1(mp11),
        .data_2(mp12),
        .data_3(mp13),
        .data_min(min[0]),
        .data_middle(mid[0]),
        .data_max(max[0])
        );

compare U_row2(
        .clk(clk),
        .rst(rst),
        .data_1(mp21),
        .data_2(mp22),
        .data_3(mp23),
        .data_min(min[1]),
        .data_middle(mid[1]),
        .data_max(max[1])
        );

compare U_row3(
        .clk(clk),
        .rst(rst),
        .data_1(mp31),
        .data_2(mp32),
        .data_3(mp33),
        .data_min(min[2]),
        .data_middle(mid[2]),
        .data_max(max[2])
        );

//step2:小中取大 大中取小，中中取中
// maxMin == max_of_Min 数组
wire [7:0]maxMin,minMax,midMid;
compare U_col1(
        .clk(clk),
        .rst(rst),
        .data_1(min[0]),
        .data_2(min[1]),
        .data_3(min[2]),
        .data_min(),
        .data_middle(),
        .data_max(maxMin)
        );

compare U_col2(
        .clk(clk),
        .rst(rst),
        .data_1(max[0]),
        .data_2(max[1]),
        .data_3(max[2]),
        .data_min(minMax),
        .data_middle(),
        .data_max()
        );

compare U_col3(
        .clk(clk),
        .rst(rst),
        .data_1(mid[0]),
        .data_2(mid[1]),
        .data_3(mid[2]),
        .data_min(),
        .data_middle(midMid),
        .data_max()
```

```
        );

    //step3:再对三个最值取中值
    compare U_col0(
        .clk(clk),
        .rst(rst),
        .data_1(minMax),
        .data_2(maxMin),
        .data_3(midMid),
        .data_min(),
        .data_middle(median_filter_data),
        .data_max()
        );

    //取中值共需 3 步，三个延迟
    //上面的比较还需要 3clk,需要延迟 3clk
    //再次延迟
    reg [2:0] pic_ena_r3;
    reg [2:0] vs_r3,hs_r3;
    always@(posedge clk or posedge rst)
        begin
            if(rst)
            begin
                pic_ena_r3 <= 0;
                vs_r3 <= 0;
                hs_r3 <= 0;
            end
            else
            begin
                pic_ena_r3                  <=
{pic_ena_r3[1:0],matrix_frame_ena};
                vs_r3 <= {vs_r3[1:0],matrix_frame_vs};
                hs_r3 <= {hs_r3[1:0],matrix_frame_hs};
            end
        end
        assign post_hs = hs_r3[2];
        assign post_vs = vs_r3[2];
        assign post_pic_ena = pic_ena_r3[2];
endmodule
```

4.3. Sobel 边缘检测

描述：边缘是周围像素急剧变化的点的集合。利用 Sobel 算子卷积，提取图像边缘。

```
module sobel_edge_detect(
    input clk,
    input rst,
    input pic_ena,    //图片使能，表示这个 clk 在
处理图片
    input vs,
    input hs,
    input   [15:0]threshold,
    input   [7:0]median_filter_data,
    output sobel_detect_data_bit,
    output post_vs,
    output post_hs,
    output post_pic_ena
   );
    //pipline method
    //for simple, |G| = |Gx| + |Gy|

    wire [7:0]row1,row2;
    reg   [7:0]row3;

    //只用两个 shift ram 级联
    //vivado 只有单行 shift ram
    shift_ram_3x3_8bits U_shift_ram1 (
        .D(row3),             // input wire [7 : 0] D
        .CLK(clk),      // input wire CLK //这里
debug 了好久好久!!!!
        .CE(pic_ena),         // input wire CE    //
这里也换成使能的，减少 slack
        .SCLR(rst),   // input wire SCLR
        .Q(row2)              // output wire [7 : 0] Q
    );

    shift_ram_3x3_8bits U_shift_ram0 (
        .D(row2),             // input wire [7 : 0] D
        .CLK(clk),       // input wire CLK
        .CE(pic_ena),         // input wire CE
        .SCLR(rst),   // input wire SCLR
        .Q(row1)              // output wire [7 : 0] Q
    );

    //  数据进入 shift ram  消耗 1 时钟
```

```
always@(posedge clk or posedge rst)
begin
    if(rst)
        row3 <= 0;
    else if(pic_ena)
        row3 <= median_filter_data;
    else
        row3 <= row3;
end

//延迟方法如下：
//log 2 clocks signal sync
reg [1:0] pic_ena_r1;
reg [1:0] vs_r1,hs_r1;
always@(posedge clk or posedge rst)
begin
    if(rst)
    begin
        pic_ena_r1 <= 0;
        vs_r1 <= 0;
        hs_r1 <= 0;
    end
    else
    begin
        pic_ena_r1                    <=
{pic_ena_r1[0],pic_ena};
        vs_r1 <= {vs_r1[0],vs};
        hs_r1 <= {hs_r1[0],hs};
    end
end

wire  read_matrix_ena  =  pic_ena_r1[0];//取[0],
表示 log 1，是读入 shift ram 的，

//之后会取[1]表示 log 2，是读出矩阵的延迟
assign read_matrix_hs = hs_r1[0];
assign read_matrix_vs = vs_r1[0];

//根据行、场同步信号读取 3X3 阵列
//matrix_p_x_y
reg [7:0] mp11,mp12,mp13;
reg [7:0] mp21,mp22,mp23;
reg [7:0] mp31,mp32,mp33;

always @(posedge clk or posedge rst)
begin
    if(rst)begin
        {mp11,mp12,mp13}<=0;
        {mp21,mp22,mp23}<=0;
        {mp31,mp32,mp33}<=0;
    end
    else if(read_matrix_ena)begin

{mp11,mp12,mp13}<={mp12,mp13,row1};

{mp21,mp22,mp23}<={mp22,mp23,row2};

{mp31,mp32,mp33}<={mp32,mp33,row3};
    end
    else begin

{mp11,mp12,mp13}<={mp11,mp12,mp13};

{mp21,mp22,mp23}<={mp21,mp22,mp23};

{mp31,mp32,mp33}<={mp31,mp32,mp33};
    end
end

//Sobel Edge Detect
//Step 1     列，2 clk  算完一个 3X3 的 Gx
/*
Gx = | -1    0    +1 |
     | -2    0    +2 |
     | -1    0    +1 |
*/
reg [9:0] gx1, gx2, Gx; //gx1 + ; gx2 - ;
always @(posedge clk or posedge rst) begin
    if(rst)begin
        gx1 <= 0;
        gx2 <= 0;
        Gx <= 0;
    end
    else begin
        gx1 <= mp13 + (mp23 << 1) + mp33;
// +
        gx2 <= mp11 + (mp21 << 1) + mp31;
// -
```

```
                Gx <= (gx1 >= gx2) ? gx1 - gx2 :
gx2 - gx1 ; // |Gx| = |gx1 - gx2|
            end
        end

        //同理 算列
        reg [9:0] gy1, gy2, Gy; //gx1 + ; gx2 - ;
        always @(posedge clk or posedge rst) begin
            if(rst)begin
                gy1 <= 0;
                gy2 <= 0;
                Gy <= 0;
            end
            else begin
                gy1 <= mp11 + (mp12 << 1) + mp13;
        // +
                gy2 <= mp31 + (mp32 << 1) + mp33;
        // -
                Gy <= (gy1 >= gy2) ? gy1 - gy2 :
gy2 - gy1 ; // |Gx| = |gx1 - gx2|
            end
        end

        // Step 2, 0 clk
        wire [10:0] G; //宽度 为 10 ，因为 Gx 和 Gy
都只有 9 位，肯定不会超
        assign G = Gy + Gx ;//simplified

        // Step 3, threshold, 1 clk
        reg sobel_detect_data_bit_r;
        assign sobel_detect_data_bit =
sobel_detect_data_bit_r;
        always @(posedge clk or posedge rst) begin
            if(rst)
                sobel_detect_data_bit_r <= 1'b0;
            else if (G >= threshold)
                sobel_detect_data_bit_r <= 1'b1;
            else
                sobel_detect_data_bit_r <= 1'b0;
        end

        //delay 3 clk
        delay_nclk #(.N(3))
            U_delay_2clk (
            .clk(clk),
            .rst(rst),
            .ena(pic_ena_r1[1]),
            .vs(vs_r1[1]),
            .hs(hs_r1[1]),
            .post_vs(post_vs),
            .post_hs(post_hs),
            .post_ena(post_pic_ena)
        );
endmodule
```

4.4. 腐蚀

描述：消除边界点，使边界向内部收缩，消除小及无意义的像素点。

```
module erode(
        input clk,
        input rst,
        input pic_ena,
        input vs,
        input hs,
        input sobel_detect_data_bit,
        output erode_data,
        output post_vs,
        output post_hs,
        output post_pic_ena
       );

        wire row2,row1;
        reg row3;

        shift_ram_3x3_1bit U_shift_ram_3x3_1bit1 (
            .D(row3),           // input wire [0 : 0] D
            .CLK(clk),          // input wire CLK
            .CE(pic_ena),       // input wire CE
            .SCLR(rst),         // input wire SCLR
            .Q(row2)            // output wire [0 : 0] Q
        );

        shift_ram_3x3_1bit U_shift_ram_3x3_1bit0 (
            .D(row2),           // input wire [0 : 0] D
            .CLK(clk),          // input wire CLK
            .CE(pic_ena),       // input wire CE
```

```
        .SCLR(rst),        // input wire SCLR
        .Q(row1)           // output wire [0 : 0] Q
    );

    // consume 1 clk
    always@(posedge clk or posedge rst) begin
        if(rst)
            row3 <= 0;
        else if(pic_ena)
            row3 <= sobel_detect_data_bit;
        else
            row3 <= row3;
    end

    //延迟方法如下：
    //log 2 clocks signal sync
    reg [1:0] pic_ena_r1;
    reg [1:0] vs_r1,hs_r1;
    always@(posedge clk or posedge rst)
    begin
        if(rst)
        begin
            pic_ena_r1 <= 0;
            vs_r1 <= 0;
            hs_r1 <= 0;
        end
        else
        begin
            pic_ena_r1                    <=
{pic_ena_r1[0],pic_ena};
            vs_r1 <= {vs_r1[0],vs};
            hs_r1 <= {hs_r1[0],hs};
        end
    end

    wire  read_matrix_ena  =  pic_ena_r1[0];//取[0],
表示 log 1，是读入 shift ram 的，

    //之后会取[1]表示 log 2，是读出矩阵的延迟
    assign read_matrix_hs = hs_r1[0];
    assign read_matrix_vs = vs_r1[0];

    //根据行、场同步信号读取 3X3 阵列
    //matrix_p_x_y
    reg   mp11,mp12,mp13;
    reg   mp21,mp22,mp23;
    reg   mp31,mp32,mp33;

    always @(posedge clk or posedge rst) begin
        if(rst) begin
            {mp11,mp12,mp13}<=0;
            {mp21,mp22,mp23}<=0;
            {mp31,mp32,mp33}<=0;
        end
        else if(read_matrix_ena) begin

{mp11,mp12,mp13}<={mp12,mp13,row1};

{mp21,mp22,mp23}<={mp22,mp23,row2};

{mp31,mp32,mp33}<={mp32,mp33,row3};
        end
        else begin

{mp11,mp12,mp13}<={mp11,mp12,mp13};

{mp21,mp22,mp23}<={mp21,mp22,mp23};

{mp31,mp32,mp33}<={mp31,mp32,mp33};
        end
    end

    //    以上除了 IP core 的 width 是 1bit 以
外，其它都和 sobel、median 一样
    // 分两步 流水线 加快速度
    // Step 1,对每一行&
    reg line1,line2,line3;
    always @(posedge clk or posedge rst) begin
        if(rst) begin
            line1 <= 0;
            line2 <= 0;
            line3 <= 0;
        end
        else begin
            line1 <= mp11 & mp12 & mp13;
            line2 <= mp21 & mp22 & mp23;
            line3 <= mp31 & mp32 & mp33;
        end
```

```
    end

    // Step 2,对每一行&的结果再与
    reg line;
    always @(posedge clk or posedge rst) begin
        if(rst) begin
            line <= 0;
        end
        else begin
            line <= line1 & line2 & line3;
        end
    end

    assign erode_data = line;
    delay_nclk #(.N(2))
        U_delay_2clk (
        .clk(clk),
        .rst(rst),
        .ena(pic_ena_r1[1]),
        .vs(vs_r1[1]),
        .hs(hs_r1[1]),
        .post_vs(post_vs),
        .post_hs(post_hs),
        .post_ena(post_pic_ena)
    );
endmodule
```

4.5. 膨胀

描述：将与物体接触的所有背景点合并到该物体中，使边界向外部扩张的过程。

```
module dilate(
    input clk,
    input rst,
    input pic_ena,
    input vs,
    input hs,
    input erode_data,
    output dilate_data,
    output post_vs,
    output post_hs,
    output post_pic_ena
   );

    wire row2,row1;
    reg row3;

    shift_ram_3x3_1bit U_shift_ram_3x3_1bit1 (
        .D(row3),           // input wire [0 : 0] D
        .CLK(clk),          // input wire CLK
        .CE(pic_ena),       // input wire CE
        .SCLR(rst),         // input wire SCLR
        .Q(row2)            // output wire [0 : 0] Q
    );

    shift_ram_3x3_1bit U_shift_ram_3x3_1bit0 (
        .D(row2),           // input wire [0 : 0] D
        .CLK(clk),          // input wire CLK
        .CE(pic_ena),       // input wire CE
        .SCLR(rst),         // input wire SCLR
        .Q(row1)            // output wire [0 : 0] Q
    );

    // consume 1 clk
    always@(posedge clk or posedge rst) begin
        if(rst)
            row3 <= 0;
        else if(pic_ena)
            row3 <= erode_data;
        else
            row3 <= row3;
    end

    //延迟方法如下：
    //log 2 clocks signal sync
    reg [1:0] pic_ena_r1;
    reg [1:0] vs_r1,hs_r1;
    always@(posedge clk or posedge rst)
    begin
        if(rst)
        begin
            pic_ena_r1 <= 0;
            vs_r1 <= 0;
            hs_r1 <= 0;
        end
        else
        begin
```

```
                pic_ena_r1                          <=
{pic_ena_r1[0],pic_ena};
                vs_r1 <= {vs_r1[0],vs};
                hs_r1 <= {hs_r1[0],hs};
            end
        end

        wire read_matrix_ena = pic_ena_r1[0];//取[0],表示 log 1，是读入 shift ram 的，

        //之后会取[1]表示 log 2，是读出矩阵的延迟
        assign read_matrix_hs = hs_r1[0];
        assign read_matrix_vs = vs_r1[0];

        //根据行、场同步信号读取 3X3 阵列
        //matrix_p_x_y
        reg   mp11,mp12,mp13;
        reg   mp21,mp22,mp23;
        reg   mp31,mp32,mp33;

        always @(posedge clk or posedge rst) begin
            if(rst) begin
                {mp11,mp12,mp13}<=0;
                {mp21,mp22,mp23}<=0;
                {mp31,mp32,mp33}<=0;
            end
            else if(read_matrix_ena) begin

        {mp11,mp12,mp13}<={mp12,mp13,row1};

        {mp21,mp22,mp23}<={mp22,mp23,row2};

        {mp31,mp32,mp33}<={mp32,mp33,row3};
            end
            else begin

        {mp11,mp12,mp13}<={mp11,mp12,mp13};

        {mp21,mp22,mp23}<={mp21,mp22,mp23};

        {mp31,mp32,mp33}<={mp31,mp32,mp33};
            end
        end

        //    以上除了 IP core 的 width 是 1bit 以外，其它都和 sobel、median 一样
        // 分两步 流水线 加快速度
        // Step 1,对每一行&
        reg line1,line2,line3;
        always @(posedge clk or posedge rst) begin
            if(rst) begin
                line1 <= 0;
                line2 <= 0;
                line3 <= 0;
            end
            else begin
                line1 <= mp11 | mp12 | mp13;
                line2 <= mp21 | mp22 | mp23;
                line3 <= mp31 | mp32 | mp33;
            end
        end

        // Step 2,对每一行&的结果再与
        reg line;
        always @(posedge clk or posedge rst) begin
            if(rst) begin
                line <= 0;
            end
            else begin
                line <= line1 | line2 | line3;
            end
        end

        assign dilate_data = line;
        delay_nclk #(.N(2))
            U_delay_2clk (
            .clk(clk),
            .rst(rst),
            .ena(pic_ena_r1[1]),
            .vs(vs_r1[1]),
            .hs(hs_r1[1]),
            .post_vs(post_vs),
            .post_hs(post_hs),
            .post_ena(post_pic_ena)
        );
    endmodule
```

五、测试模块建模

（要求列写各建模模块的 test bench 模块代码）

笔者使用了很多 IP core，发现无论是用 Modelsim 还是 Vivado 自带的仿真器都不行，卡在加载界面。而且网上的方法都行不通。故验证大多是综合后观察 Schematic 图和下板后现象猜测 Bug 所在。

以下是不带 IP core 的模块的 test bench。

```
module delay_nclk_tb;
    reg clk = 0;
    reg [7:0]grey_data = 0;
    wire [7:0] median_filter_data;

    median_filter U_med(
        .clk(clk_40m),
        .rst(1'b0),
        .vs(1'b1),
        .hs(1'b1),
        .grey_data(grey_data),
        .pic_ena(1'b1),
        .median_filter_data(median_filter_data),
        .post_vs(),
        .post_hs(),
        .post_pic_ena()
    );

//  genvar i;
    //for(i=0;i<2000;i = i + 1)
//  begin
//      5 clk = ~clk;
//  end
    always #1 clk = ~clk;
    always @(negedge clk)
    begin
        grey_data = grey_data + 8'b0000_0001;
    end
endmodule

module Uart_receiver_tb;//(
    //input clk,
    //input rst,
    //input rxd,
    //output [15:0]led

//);
    reg clk;
    reg rst;
    reg rxd;
    wire [15:0]led;
    wire rxd_flag;

    wire divide_clken;

    clk_divider_precise
    #(
        //DEVIDE_CNT = 42.94967296 * fo
    //  parameter   DEVIDE_CNT  =
32'd175921860   //256000bps * 16
    //  parameter   DEVIDE_CNT  =
32'd87960930    //128000bps * 16
    //  parameter   DEVIDE_CNT  =
32'd79164837    //115200bps * 16
        .DEVIDE_CNT(32'd6597070)
    //9600bps * 16
    )
    U_clk_divider_precise
    (
        .clk(clk), //!!!!!原始 100MHZ 的 clk!!!!
        .rst_n(~rst),

        .divide_clk(),//禁止使用分频后的时钟
详情见网页收藏夹
        .divide_clken(divide_clken) //对于全局时
钟 clk 100MHZ 的时钟使能
    );

    Uart_receiver U_receiver
    (
        .clk(clk),
        .rst_n(~rst),
        .clken_16bps(divide_clken),//clk_bps * 16
```

```
        .rxd(rxd),          //uart txd interface

        .rxd_data(led),    //uart data received
        .rxd_flag(rxd_flag)     //uart  data  receive
done
    );

    integer j,i;
    initial
    begin
        //choose = 4'b0;
        rxd = 1'b0;
        rst = 1'b1;
        #5;
        for(j = 0; j <= 2000; j = j + 1)
        begin
            for(i = 0;i < 10; i = i + 1)
            begin
                if (i == 0)
                    rxd = 1;
                else
                    ;

            end
            clk = 0;
            #1;
            clk = 1;
            #1;
        end
    end
endmodule

module three_fsm_tb;
    reg clk;
    reg [3:0] choose;
    reg         [15:0]hs_3fsm          =
16'b1111_1111_1010_1010;
    reg         [15:0]vs_3fsm          =
16'b1111_1111_0001_0001;
    reg [15:0]pic_ena = 16'b1111_1111_0101_0101;
    wire [11:0]VGA_data;
    wire final_pic_ena;
    wire HSYNC,VSYNC;

    three_fsm U_3fsm(
        .clk(clk),
        .rst(1'b0),
        .choose(choose), //选择
        .origin_data(16'b0111_1110_1001_0011),
        .grey_data(8'b1110_1111),
        .median_data(8'b0101_1111),
        .hs(hs_3fsm),          //4 bits 的 choose
有 16 种 预留好
        .vs(vs_3fsm),
        .pic_ena(pic_ena),
        .sobel_data(1'b1),
        .erode_data(1'b1),
        .dilate_data(1'b1),
        .post_hs(HSYNC),//[15:0]
        .post_vs(VSYNC),//[15:0]
        .post_pic_ena(final_pic_ena),//[15:0]
        .VGA_data(VGA_data)
    );

    integer j;
    initial
    begin
        choose = 4'b0;
        #5;
        for(j = 0; j <= 6; j = j + 1)
        begin
            clk = 0;

            #1;
            clk = 1;
            #5;

        end
    end
endmodule

module uart_test(
    input clk,
    input rst,
    input rxd,
    output [15:0]led
    );
```

```
    wire divide_clken;
    //reg [7:0]rxd_data ;
    //assign led [7:0] =    rxd_data[7:0];

    uart_receiver U_receiver
    (
        //gobal clock
        .clk(clk),
        .rst_n(~rst),

        //user interface
        .clken_16bps(divide_clken),//clk_bps * 16

        //uart interface
        .rxd(rxd),          //uart txd interface
        .rxd_data(led [15:0]),   //uart data receive
        .rxd_flag()  //uart data receive done
    );

    precise_divider
    #(
        //DEVIDE_CNT = 42.94967296 * fo
    //  parameter       DEVIDE_CNT          =
32'd175921860   //256000bps * 16
    //  parameter       DEVIDE_CNT          =
32'd87960930    //128000bps * 16
    //  parameter       DEVIDE_CNT          =
32'd79164837    //115200bps * 16
        .DEVIDE_CNT(32'd6597070)
    //9600bps * 16
    )
    U_divider
    (
        //global clock
        .clk(clk),
        .rst_n(~rst),

        //user interface
        .divide_clk(),
        .divide_clken(divide_clken)
    );

endmodule
```

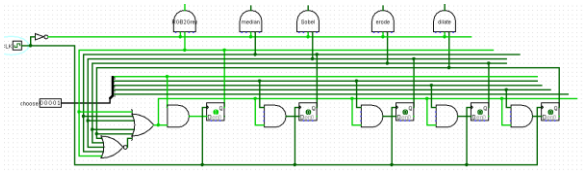
```
module fsm7_tb;

    reg    clk;
    reg    rst_n;
    parameter PERIOD = 10;     //100MHz
    initial
    begin
        clk = 0;
        forever      #(PERIOD/2)
        clk = ~clk;
    end

    task task_reset;
    begin
        rst_n = 0;
        repeat(2) @(negedge clk);
        rst_n = 1;
    end
    endtask
    wire  clk_ref = clk;
    wire  sys_rst_n = rst_n;



    //------------------------------------
    //uart interface
    reg                     fpga_rxd;
    //wire              fpga_txd;


    wire  divide_clken2;
    clk_divider_precise
    #(
        //DEVIDE_CNT = 42.94967296 * fo
        //.DEVIDE_CNT        (32'd412317)
    //9600bps
        .DEVIDE_CNT  (32'd10995116)
    //256000bps
    )
    u_precise_divider2
    (
        //global
```

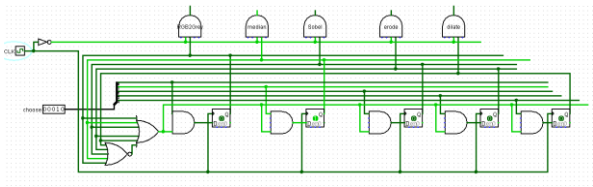
```
        .clk                (clk_ref),
    //100MHz clock
        .rst_n              (sys_rst_n),
//global reset

        //user interface
        .divide_clk         (),
        .divide_clken       (divide_clken2)
    );


    //uart data txd simulate
    wire  uart_bps_en =    divide_clken2;
    task  task_uart_txd;
    input [7:0] uart_data;
    begin
        //8'hCB = 8'b1100_1011
        @(posedge uart_bps_en);   fpga_rxd = 0;
    //Start
        @(posedge  uart_bps_en);    fpga_rxd  =
uart_data[0];
        @(posedge  uart_bps_en);    fpga_rxd  =
uart_data[1];
        @(posedge  uart_bps_en);    fpga_rxd  =
uart_data[2];
        @(posedge  uart_bps_en);    fpga_rxd  =
uart_data[3];
        @(posedge  uart_bps_en);    fpga_rxd  =
uart_data[4];
        @(posedge  uart_bps_en);    fpga_rxd  =
uart_data[5];
        @(posedge  uart_bps_en);    fpga_rxd  =
uart_data[6];
        @(posedge  uart_bps_en);    fpga_rxd  =
uart_data[7];
        @(posedge uart_bps_en);   fpga_rxd = 1;
    //End
        //#78;
    end
    endtask



    //-------------------------------------------
    //system initialization
task task_sysinit;
begin
    fpga_rxd = 1;
end
endtask

//---------------------------------------
//testbench of the RTL
integer i;
initial
begin
    task_sysinit;
    task_reset;
    for(i=0 ;i< 32'hffff_ffff; i=i+1)
    begin

    task_uart_txd(8'hCB);
    task_uart_txd(8'h0A);


    task_uart_txd(8'h09);
    task_uart_txd(8'h0F);

    end
    /*
    task_uart_txd(8'h0F);
    task_uart_txd(8'h09);


    task_uart_txd(8'h01);
    task_uart_txd(8'h02);


    task_uart_txd(8'h07);
    task_uart_txd(8'h04);   */

end

wire [11:0]VGA_data;
    wire [15:0]led;
    wire HSYNC;
    wire VSYNC;
disp_VGA U_disp_VGA(
  .clk(clk),
```

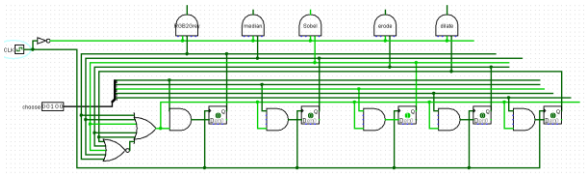
```
.rxd(fpga_rxd),                                    .led(led)
.rst(1'b0), // effective when 1                    );
.choose(4'b0001),//choose mode,binary code,
.HSYNC(HSYNC),
.VSYNC(VSYNC),
.VGA_data(VGA_data),//,                    endmodule
```

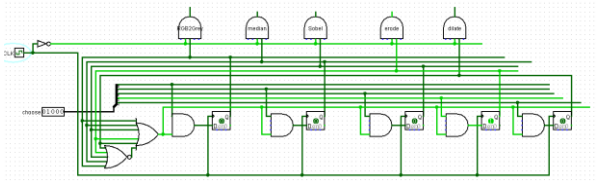
六、实验结果

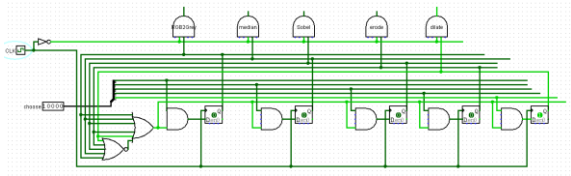
（该部分可截图说明，可包含 logisim 逻辑验证图、modelsim 仿真波形图、以及下板后的实验结果贴图）

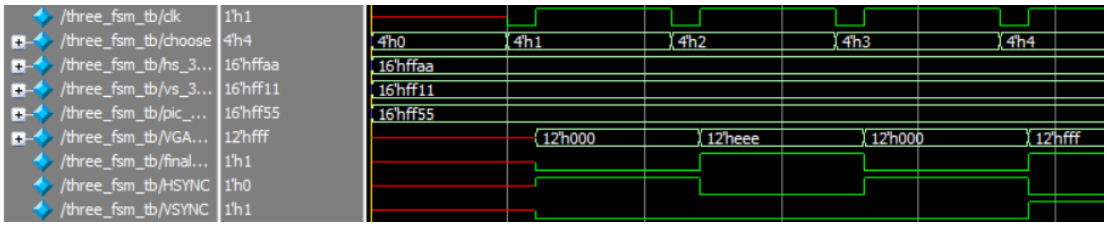


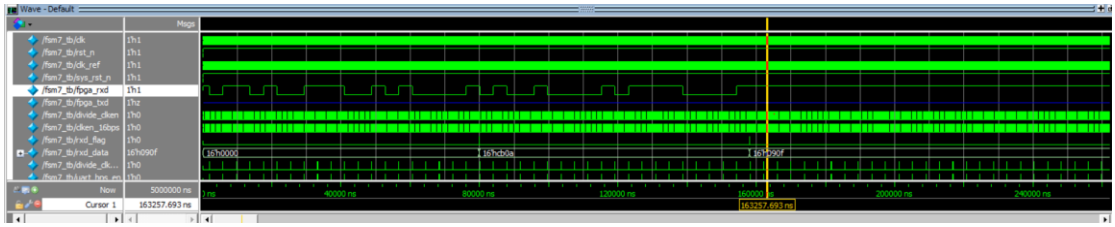


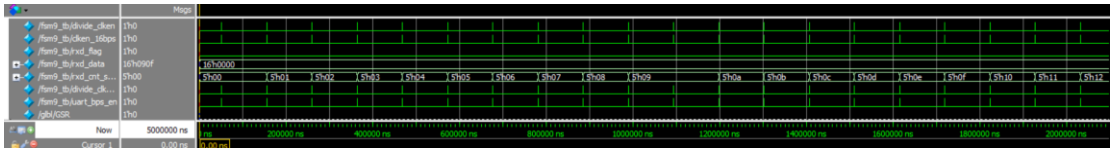


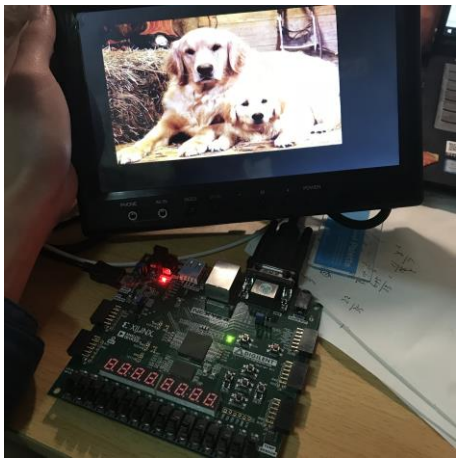


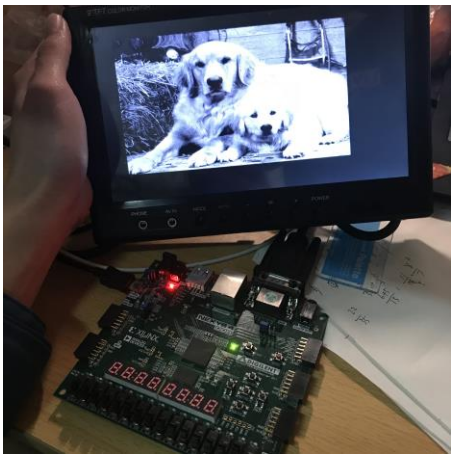


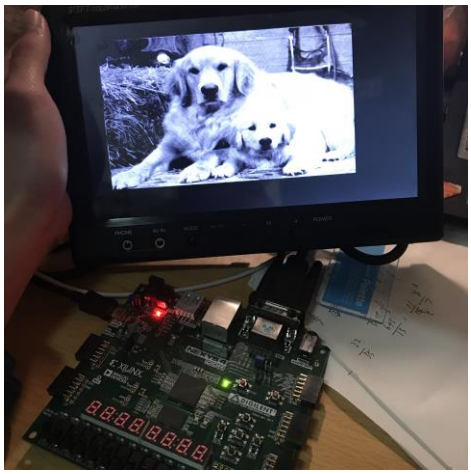


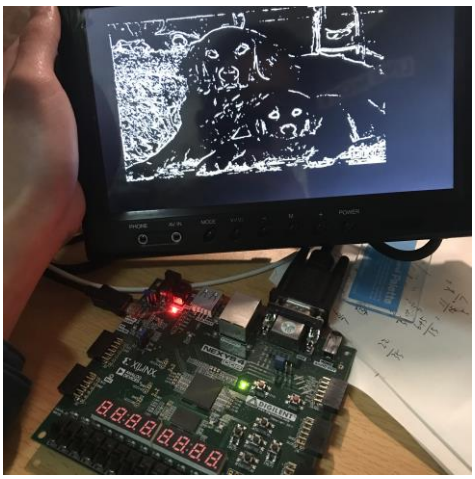

Wave - Default
Msgs
/fsm7_tb/clk
/fsm7_tb/rst_n
/fsm7_tb/clk_ref
/fsm7_tb/sys_rst_n
/fsm7_tb/fpga_rxd
/fsm7_tb/fpga_txd
/fsm7_tb/divide_clken
/fsm7_tb/clken_16bps
/fsm7_tb/rxd_flag
/fsm7_tb/rxd_data
/fsm7_tb/divide_clk...
/fsm7_tb/uart_bps_en
Now
Cursor 1
163257.693 ns

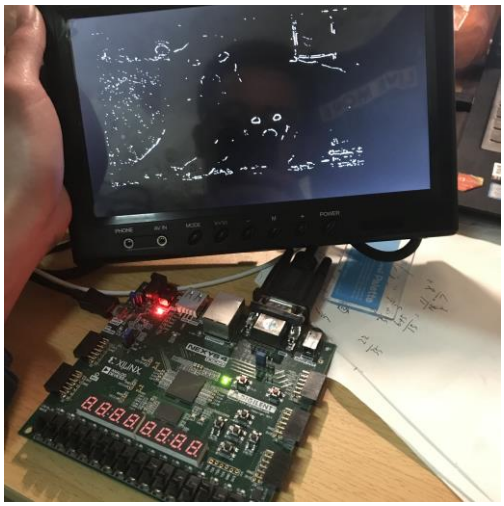

Msgs
/fsm9_tb/divide_clken
/fsm9_tb/clken_16bps
/fsm9_tb/rxd_flag
/fsm9_tb/rxd_data
/fsm9_tb/rxd_cnt_s...
/fsm9_tb/divide_clk...
/fsm9_tb/uart_bps_en
/glbl/GSR
Now
5000000 ns
Cursor 1
0.00 ns

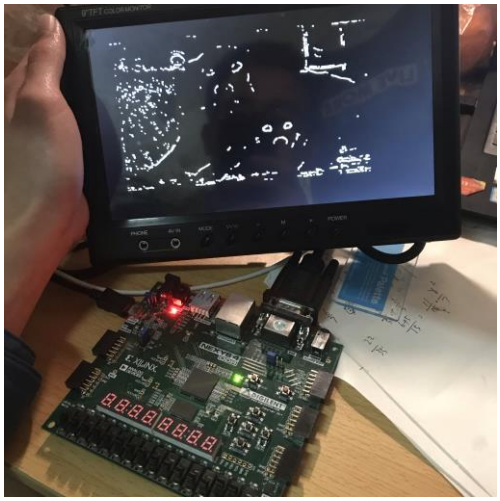












七、结论

采用流水线结构设计后，速度快，逻辑简单，占用资源少。

本次综合实验的图像处理为一般的图像处理的标准流程。对于图像的处理，实验者可在研究过程中深刻体会软件和硬件编程的差异，并对时序有了更深层次的理解，在以后的实验中加强运用流水线的设计方法。

八、心得体会及建议

8.1.Debug 调试心得体会

8.1.1. 活用时序报告

有一次怎么找也找不到问题所在，回忆起上课时老师提到过时序分析报告，于是翻看实验书，网上搜索资料，自学时序分析，将时序报告看懂 40%，根据时序报告 debug。主要是为了消除 slack。

有一次，我的模块 clk 端口是.clk(clk & ena),clk 是我的时钟，ena 是我的使能信号，这种写法会导致 clk 进入端口的延迟加大。发现时序报告说这条路径的 slack 过大标红，查看路径，发现是这里的问题，于是新添加使能端，改成:

.clk(clk)

.ena(ena)

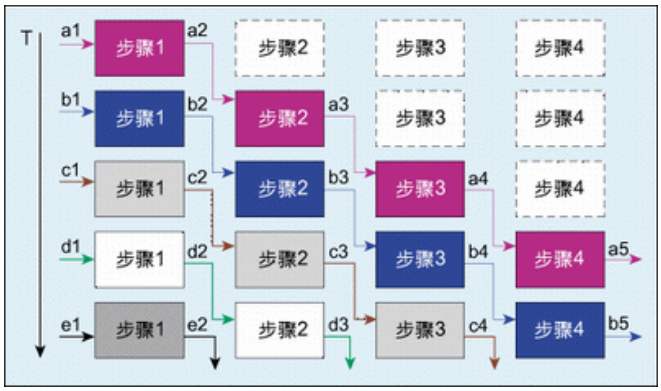
Slack 就消除了。

8.1.2. 对于自顶向下的设计的理解

我是先写好了图像处理，想再加一个 Uart 发送彩色图片。但是发现时序什么的都要改，改得乱七八糟，像是重新写一样。我的感觉是，硬件编程灵活度很差，不像软件编程，删减都很方便。其中主要的原因在于时序安排。

8.1.3. 流水线的使用和体会

其实本次图像处理可以完全用组合逻辑完成，但是速度很慢。采用流水线思想的话就会很快。



但同时也要处理好数据依赖的问题。

8.1.4. 对 Uart 的认识

一开始我没找到 Uart 接口，后来发现它和电源线是一起的，就很困惑，也就是说它一线三用，送 bitstream、输电、Uart 串口。后来我找了很多资料都没有找到解释。最后我回归本源，看了板子的手册才明白一切都在手册里。

JTAG 是用来导.bit 的，和 UART 共用一个 microUSB，并且不冲突。

The FT2232HQ is also used as the controller for the Digilent USB-JTAG circuitry, but the USB-UART and USB-JTAG functions behave entirely independent of one another. Programmers interested in using the UART functionality of the FT2232 within their design do not need to worry about the JTAG circuitry interfering with the UART data transfers, and vice-versa. The combination of these two features into a single device allows the Nexys4 to be programmed, communicated with via UART, and powered from a computer attached with a single Micro USB cable.

意思就是说一根 microUsb 既做了.bit 文件的传输（JTAG） 又做了 UART 的功能。

6 USB-UART Bridge (Serial Port)

The Nexys4 includes an FTDI FT2232HQ USB-UART bridge (attached to connector J6) that lets you use PC applications to communicate with the board using standard Windows COM port commands. Free USB-COM port drivers, available from www.ftdichip.com under the "Virtual Com Port" or VCP heading, convert USB packets to UART/serial port data. Serial port data is exchanged with the FPGA using a two-wire serial port (TXD/RXD) and optional hardware flow control (RTS/CTS). After the drivers are installed, I/O commands can be used from the PC directed to the COM port to produce serial data traffic on the C4 and D4 FPGA pins.

8.1.5. 利用 Schematic Debug

在 Schematic 中，同时打开 Report Timing Summary，能够把时序分析报告中的线路在 Schematic 标出来。

而且，如果发现综合成功但是下板不符合预期，观察 Schematic 图往往能发现问题所在。比如我打开 Schematic 图发现有模块没连上等等，立刻就找到了错误。

还有很关键的一点是，生成 bitstream 比较慢，所以我一般先看时序报告和 Schematic 图，觉得差不多了再生成 bitstream，免得浪费时间。

8.2. 认识到的一些技巧和收获的经验

8.2.1. 端口都要有 xdc 配置，否则比特流无法生成

8.2.2. ROM 里存放的 coe 文件宽度要正好，不然会提示不让过

8.2.3. 串口、COM 口、UART 口, TTL、RS-232、RS-485 区别详解

串口、COM 口是指的物理接口形式(硬件)。而 TTL、RS-232、RS-485 是指的电平标准(电信号)。

8.2.4. 禁止用计数器分频后的信号做其它模块的时钟

禁止用计数器分频后的信号做其它模块的时钟，而要用改成时钟使能的方式。否则这种时钟满天飞的方式对设计的可靠性极为不利，也大大增加了静态时序分析的复杂性。

在某系统中，前级数据输入位宽为 8 位，而后级的数据输出位宽为 32，我们需要将 8bit 数据转换为 32bit，由于后级的处理位宽为前级的 4 倍，因此后级处理的时钟频率也将下降为前级的 1/4，若不使用时钟使能，则要将前级的时钟进行 4 分频来作后级处理的时钟。这种设计方法会引入新的时钟域，处理上需要采取多时钟域处理的方式，因而在设计复杂度提高的同时系统的可靠性也将降低。为了避免以上问题，我们采用了时钟使能以减少设计复杂度。

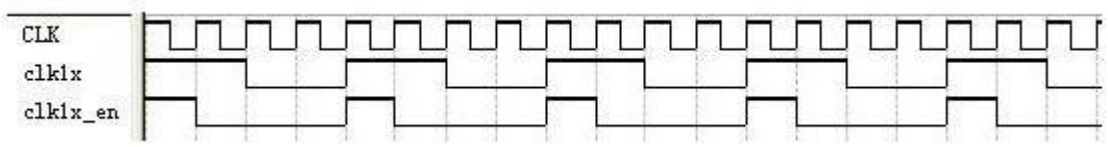
不要这样做：

```
always (posedge clk1x or negedge rst_n)
begin
    ...
end
```

而要这样做：

```
always (posedge clk or negedge rst_n)
begin
    ...
    else if (clk1x_en == 1'b1 )          //使用使能的方式。
    ...
End
```

时钟如图所示，clk1x 是 CLK 的四分频后产生的时钟，clk1x_en 是与 clk1x 同频的时钟使能信号。



参考网址：http://bbs.elecfans.com/jishu_1627880_1_1.html

8.2.5. FPGA 浮点运算

FPGA 无法进行浮点运算，所以我们采取将整个式子都扩大 256 倍，然后再右移 8 位，这样就得到了 FPGA 擅长的乘法运算和加法运算了。

FPGA 的 IP Core 中还有支持浮点运算的 IP core。

8.2.6. 一般采用下降沿复位

8.3. 尚未解决的问题

8.3.1. 对时序分析这一块还是不够熟练，脑子里还是没有电路

8.3.2. 对于 Verilog 的代码还不够规范

8.3.3. 我深刻体会到，D 是 Description 的意思了。Verilog 是硬件描述语言，但不是设计语言，设计是要我们来设计的，而且是一切尽可能完备的情况下才开始写代码。我们应当根据功能，脑海中生成各个硬件的组合，最后用 Verilog 把这一切表达出来；而不

是先写 **Verilog**，期待它能实现你想要的功能。我觉得硬件编程难就难在这里，不能走一步看一步，必须一次性走完。