小梅哥 AC620 FPGA 开发板进阶设计教程

全功能高性价比 FPGA 开发板 AC620 火热销售中，配套 60 天实地培训视频。

购买链接：

https://item.taobao.com/item.htm?spm=a1z10.1-c-s.w4004-13687301132.6.8uPVJ6&id=544830995588

目录

说　　明

本教程为小梅哥 **AC620 FPGA** 开发板适配的图像处理相关教程，旨在展示使用 **FPGA** 进行图像处理的一般方法，例程源码基于《**FPGA** 自学笔记——设计与验证》一书第 **6.6** 节内容，通过加入图像处理算法得到。具体例程使用方法，与书中板级验证方法一致。教程内容将会不定期更新新的算法内容，所有例程源码，均可在芯航线 **FPGA** 客户专属群“**286285840**”群文件下载。如果文档有修正或更新，也将及时上传在群文件。

基于 FPGA 图像处理之 RGB 转灰度算法的实现

背景知识

在正式入题之前先给大家讲解一下 gray 图像，YUV 图像以及 Ycbcr 图像。

Gray 图像：灰度（gray）图像就是我们常说的黑白图像，由黑到白为灰阶为 0-255(8bit)。

YUV 是被欧洲电视系统所采用的一种颜色编码方法（属于 PAL），是 PAL 和 SECAM 模拟彩色电视制式采用的颜色空间。在现代彩色电视系统中，通常采用三管彩色摄影机或彩色 CCD 摄影机进行取像，然后把取得的彩色图像信号经分色、分别放大校正后得到 RGB，再经过矩阵变换电路得到亮度信号 Y 和两个色差信号 B—Y（即 U）、R—Y（即 V），最后发送端将亮度和色差三个信号分别进行编码，用同一信道发送出去。这种色彩的表示方法就是所谓的 YUV 色彩空间表示。采用 YUV 色彩空间的重要性是它的亮度信号 Y 和色度信号 U、V 是分离的。YUV 主要用于优化彩色视频信号的传输，使其向后相容老式黑白电视。与 RGB 视频信号传输相比，它最大的优点在于只需占用极少的频宽（RGB 要求三个独立的视频信号同时传输）。其中“Y”表示明亮度（Luminance 或 Luma），也就是灰阶值；而“U”和“V” 表示的则是色度（Chrominance 或 Chroma），作用是描述影像色彩及饱和度，用于指定像素的颜色。“亮度”是透过 RGB 输入信号来建立的，方法是将 RGB 信号的特定部分叠加到一起。“色度”则定义了颜色的两个方面—色调与饱和度，分别用 Cr 和 Cb 来表示。其中，Cr 反映了 RGB 输入信号红色部分与 RGB 信号亮度值之间的差异。而 Cb 反映的是 RGB 输入信号蓝色部分与 RGB 信号亮度值之间的差异。

Ycbcr 或 Y'CbCr 有的时候会被写作：YCBCR 或是 Y'CBCR，是色彩空间的一种，通常会用于影片中的影像连续处理，或是数字摄影系统中。Y'为颜色的亮度(luma)成分、而 CB 和 CR 则为蓝色和红色的浓度偏移量成份。Y'和 Y 是不同的，而 Y 就是所谓的流明(luminance)，表示光的浓度且为非线性，使用伽马

修正(gamma correction)编码处理。

FPGA 实现 RGB 图像转 Gray 图像方法

一般 RGB 像转灰度（gray）图像有两种方法:

1)使用 RGB 图像的单通道去显示图像（R，G 或 B）。

2)RGB 图像转换成 Ycbcr 图像，使用 Y 分量去显示图像，来实现彩色图像转灰度图。

RGB 单通道实现灰度图像的转换

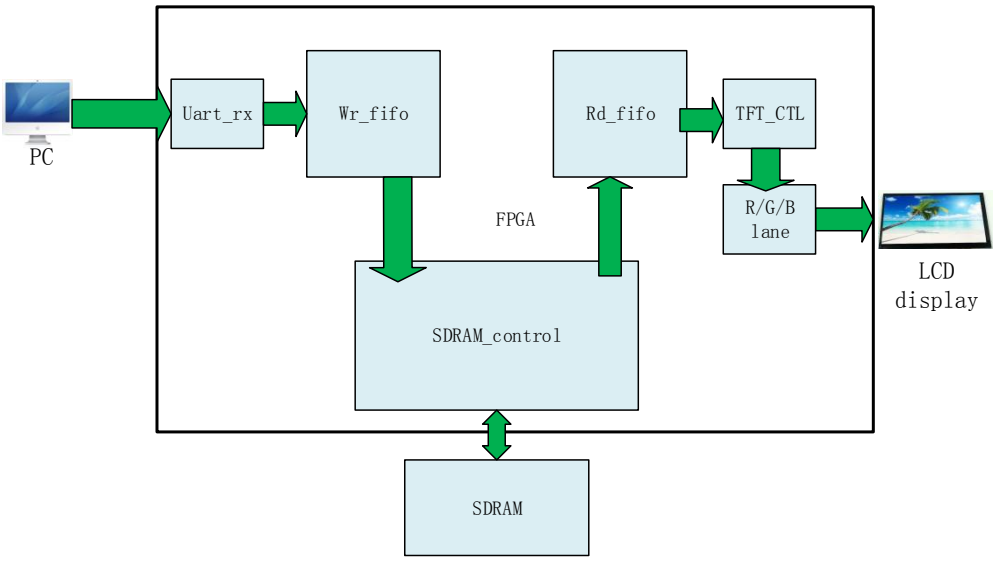


图 1 R/G/B 单通道实现灰度图像的显示

由图 1 所示，在基于 uart 传图的基础上我们增加了 R/G/B lane 模块来实现 rgb 转 gray 图像的显示。

RGB 单通道实现灰度图像 FPGA 源码：

```
//----------------------------------------------------------------------

//   R G B to gray

//----------------------------------------------------------------------
```

```
wire [15:0] rgb;

assign TFT_rgb = {rgb[15:11],rgb[15:11],1'b0,rgb[15:11]};    //red

//assign TFT_rgb = {rgb[10:6],rgb[10:5],rgb[10:6]};                //green

//assign TFT_rgb = {rgb[4:0],rgb[4:0],1'b0,rgb[4:0]};                //blue
```

实现结果：

图 2 莲花原图

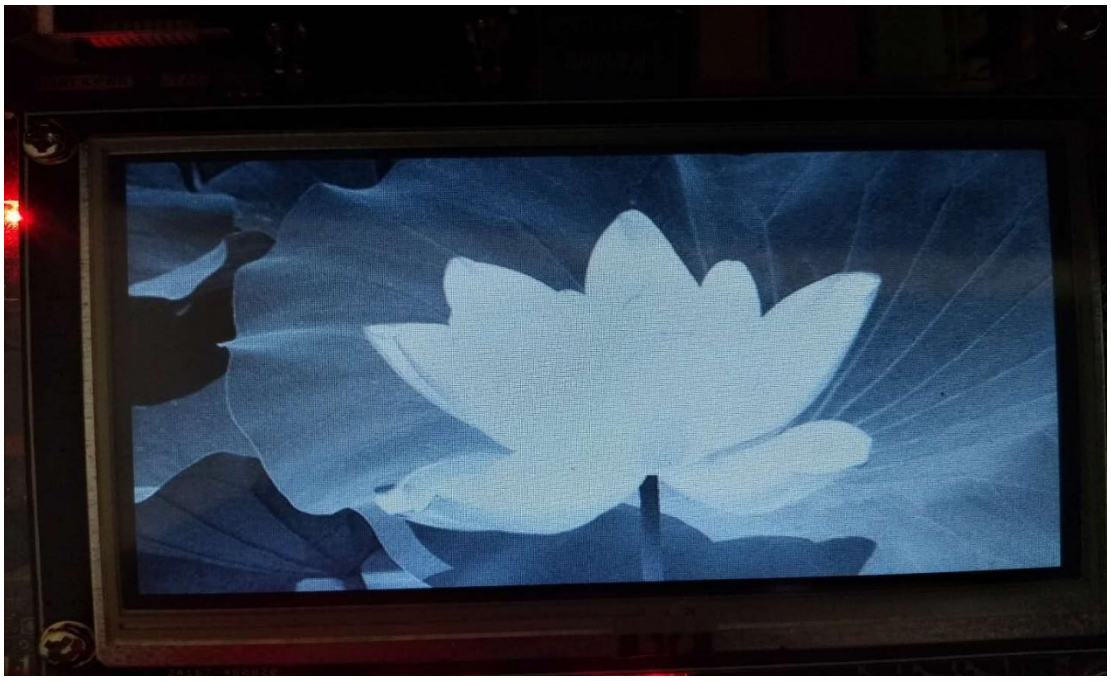


图 3 莲花 Red 分量灰度图像

图 4 莲花 Green 分量灰度图像

图 5 莲花 Blue 分量灰度图像

由图 3,4,5 三个分量显示图像来看，Green 分量显示效果较好，脉络清晰，红色分量灰度图像偏亮，蓝色分量灰度图像偏暗。大家可以多试其他图像，这种方法比较简单，容易实现。

RGB 图像转 Ycbcr 图像实现 gray 图像。

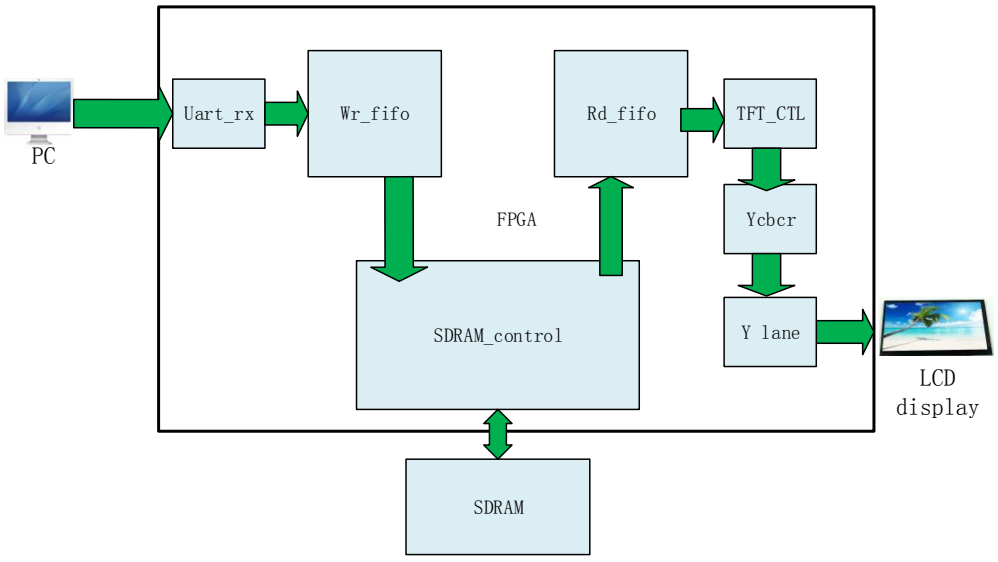


图 6 Y lane 分量实现灰度图像

RGB 转 Ycbcr 算法：

计算公式： Y = 0.183R + 0.614G + 0.062B + 16;

CB = -0.101R - 0.338G + 0.439B + 128;

CR = 0.439R - 0.399G - 0.040B + 128;

其中，时序在计算过程中完全没有用到

输入到输出有三个 clock 的时延。

第一级流水线计算所有乘法；

第二级流水线计算所有加法，把正的和负的分开进行加法；

第三级流水线计算最终的和，若为负数取 0；

RGB 转 Ycbcr FPGA 源码：

```
/*
    RGB 转 YUV 算法
    计算公式： Y = 0.183R + 0.614G + 0.062B + 16;
                CB = -0.101R - 0.338G + 0.439B + 128;
                CR = 0.439R - 0.399G - 0.040B + 128;
    其中，时序在计算过程中完全没有用到
    输入到输出有三个 clock 的时延。
    第一级流水线计算所有乘法；
    第二级流水线计算所有加法，把正的和负的分开进行加法；
    第三级流水线计算最终的和，若为负数取 0；
    仿真通过
*/
`timescale 1ns/1ps
module rgb_to_ycbcr(
                input                clk,
```

```
input               [7 : 0]     i_r_8b,
input               [7 : 0]     i_g_8b,
input               [7 : 0]     i_b_8b,

input                           i_h_sync,
input                           i_v_sync,
input                           i_data_en,

output      [7 : 0]     o_y_8b,
output      [7 : 0]     o_cb_8b,
output      [7 : 0]     o_cr_8b,

output                      o_h_sync,
output                      o_v_sync,
output                     o_data_en
);

/*************************************parameters*********************************/
//multiply 256
parameter   para_0183_10b = 10'd47;      //0.183  定点数
parameter   para_0614_10b = 10'd157;
parameter   para_0062_10b = 10'd16;
```

```
parameter   para_0101_10b = 10'd26;

parameter   para_0338_10b = 10'd86;

parameter   para_0439_10b = 10'd112;

parameter   para_0399_10b = 10'd102;

parameter   para_0040_10b = 10'd10;

parameter   para_16_18b = 18'd4096;

parameter   para_128_18b = 18'd32768;

/*****************************************************************************************/

/*********************************signals*********************************************/

wire                sign_cb;

wire                sign_cr;

reg[17: 0]    mult_r_for_y_18b;

reg[17: 0]    mult_r_for_cb_18b;

reg[17: 0]    mult_r_for_cr_18b;


reg[17: 0]    mult_g_for_y_18b;

reg[17: 0]    mult_g_for_cb_18b;

reg[17: 0]    mult_g_for_cr_18b;


reg[17: 0]    mult_b_for_y_18b;

reg[17: 0]    mult_b_for_cb_18b;

reg[17: 0]    mult_b_for_cr_18b;
```

```
reg[17: 0]    add_y_0_18b;

reg[17: 0]    add_cb_0_18b;

reg[17: 0]    add_cr_0_18b;


reg[17: 0]    add_y_1_18b;

reg[17: 0]    add_cb_1_18b;

reg[17: 0]    add_cr_1_18b;


reg[17: 0]    result_y_18b;

reg[17: 0]    result_cb_18b;

reg[17: 0]    result_cr_18b;


reg[9:0] y_tmp;

reg[9:0] cb_tmp;

reg[9:0] cr_tmp;


reg                   i_h_sync_delay_1;

reg                   i_v_sync_delay_1;

reg                   i_data_en_delay_1;


reg                   i_h_sync_delay_2;
```

```
reg                 i_v_sync_delay_2;
reg                 i_data_en_delay_2;

reg                 i_h_sync_delay_3;
reg                 i_v_sync_delay_3;
reg                 i_data_en_delay_3;
/*********************************************************************************/
/***********************************initial***************************************/
initial
begin
    mult_r_for_y_18b <= 18'd0;
    mult_r_for_cb_18b <= 18'd0;
    mult_r_for_cr_18b <= 18'd0;

    mult_g_for_y_18b <= 18'd0;
    mult_g_for_cb_18b <= 18'd0;
    mult_g_for_cr_18b <= 18'd0;

    mult_b_for_y_18b <= 18'd0;
    mult_g_for_cb_18b <= 18'd0;
    mult_b_for_cr_18b <= 18'd0;
    add_y_0_18b <= 18'd0;
```

```
    add_cb_0_18b <= 18'd0;
    add_cr_0_18b <= 18'd0;

    add_y_1_18b <= 18'd0;
    add_cb_1_18b <= 18'd0;
    add_cr_1_18b <= 18'd0;

    result_y_18b <= 18'd0;
    result_cb_18b <= 18'd0;
    result_cr_18b <= 18'd0;

    i_h_sync_delay_1 <= 1'd0;
    i_v_sync_delay_1 <= 1'd0;
    i_data_en_delay_1 <= 1'd0;

    i_h_sync_delay_2 <= 1'd0;
    i_v_sync_delay_2 <= 1'd0;
    i_data_en_delay_2 <= 1'd0;

end

/*********************************************************************************************/
```

```
/*********************************arithmetic*****************************************/

//LV1 pipeline : mult

always @ (posedge      clk)

begin

      mult_r_for_y_18b <= i_r_8b * para_0183_10b;

      mult_r_for_cb_18b <= i_r_8b * para_0101_10b;

      mult_r_for_cr_18b <= i_r_8b * para_0439_10b;

end


always @ (posedge      clk)

begin

      mult_g_for_y_18b <= i_g_8b * para_0614_10b;

      mult_g_for_cb_18b <= i_g_8b * para_0338_10b;

      mult_g_for_cr_18b <= i_g_8b * para_0399_10b;

end


always @ (posedge      clk)

begin

      mult_b_for_y_18b <= i_b_8b * para_0062_10b;

      mult_b_for_cb_18b <= i_b_8b * para_0439_10b;

      mult_b_for_cr_18b <= i_b_8b * para_0040_10b;

end
```

```
//LV2 pipeline : add
always @ (posedge      clk)
begin
      add_y_0_18b <= mult_r_for_y_18b + mult_g_for_y_18b;
      add_y_1_18b <= mult_b_for_y_18b + para_16_18b;

      add_cb_0_18b <= mult_b_for_cb_18b + para_128_18b;
      add_cb_1_18b <= mult_r_for_cb_18b + mult_g_for_cb_18b;

      add_cr_0_18b <= mult_r_for_cr_18b + para_128_18b;
      add_cr_1_18b <= mult_g_for_cr_18b + mult_b_for_cr_18b;
end
//LV3 pipeline : y + cb + cr

assign  sign_cb = (add_cb_0_18b >= add_cb_1_18b);
assign  sign_cr = (add_cr_0_18b >= add_cr_1_18b);
always @ (posedge      clk)
begin
      result_y_18b <= add_y_0_18b + add_y_1_18b;
      result_cb_18b <= sign_cb ? (add_cb_0_18b - add_cb_1_18b) : 18'd0;
      result_cr_18b <= sign_cr ? (add_cr_0_18b - add_cr_1_18b) : 18'd0;
end
```

```
always @ (posedge      clk)
begin
    y_tmp <= result_y_18b[17:8] + {9'd0,result_y_18b[7]};
    cb_tmp <= result_cb_18b[17:8] + {9'd0,result_cb_18b[7]};
    cr_tmp <= result_cr_18b[17:8] + {9'd0,result_cr_18b[7]};
end

//output
assign  o_y_8b      = (y_tmp[9:8] == 2'b00) ? y_tmp[7 : 0] : 8'hFF;
assign  o_cb_8b     = (cb_tmp[9:8] == 2'b00) ? cb_tmp[7 : 0] : 8'hFF;
assign  o_cr_8b     = (cr_tmp[9:8] == 2'b00) ? cr_tmp[7 : 0] : 8'hFF;
/*************************************************************************************/
/*********************************timing*********************************************/
always @ (posedge      clk)
begin
    i_h_sync_delay_1 <= i_h_sync;
    i_v_sync_delay_1 <= i_v_sync;
    i_data_en_delay_1 <= i_data_en;

    i_h_sync_delay_2 <= i_h_sync_delay_1;
    i_v_sync_delay_2 <= i_v_sync_delay_1;
```

```
    i_data_en_delay_2 <= i_data_en_delay_1;


    i_h_sync_delay_3 <= i_h_sync_delay_2;
    i_v_sync_delay_3 <= i_v_sync_delay_2;
    i_data_en_delay_3 <= i_data_en_delay_2;
end
//-----------------------------------
//timing
//-----------------------------------
assign  o_h_sync = i_h_sync_delay_3;
assign  o_v_sync = i_v_sync_delay_3;
assign  o_data_en = i_data_en_delay_3;


/*********************************************************************************************/
Endmodule
/*
代码 2：Ycbcr 转灰度图像
*/
wire [15:0] rgb;
wire hs;
wire vs;
wire de;
```

```
wire[7 : 0]        o_y_8b;

wire[7 : 0]        o_cb_8b;

wire[7 : 0]        o_cr_8b;


assign TFT_rgb = {o_y_8b[7:3],o_y_8b[7:2],o_y_8b[7:3]};        //Y

//assign TFT_rgb = {o_cb_8b[7:3],o_cb_8b[7:2],o_cb_8b[7:3]};    //cb

//assign TFT_rgb = {o_cr_8b[7:3],o_cr_8b[7:2],o_cr_8b[7:3]};    //cr
```

实现效果：

图 7 莲花原图

图 8 Y 分量实现效果

通过图 3,4,5 与图 8 相比，Y 分量显示的灰度图像更加具有层次感，灰度显示更加真实。

基于 FPGA 的中值滤波算法的实现

背景知识

中值滤波法是一种非线性平滑技术，它将每一像素点的灰度值设置为该点某邻域窗口内的所有像素点灰度值的中值.

中值滤波是基于排序统计理论的一种能有效抑制噪声的非线性信号处理技术，中值滤波的基本原理是把数字图像或数字序列中一点的值用该点的一个邻域中各点值的中值代替，让周围的像素值接近的真实值，从而消除孤立的噪声点。方法是用某种结构的二维滑动模板，将板内像素按照像素值的大小进行排序，生成单调上升（或下降）的为二维数据序列。二维中值滤波输出为 g（x,y）=med{f(x-

k,y-l),(k,l∈W)}，其中，f(x,y)，g(x,y)分别为原始图像和处理后图像。W 为二维模板，通常为 3*3，5*5 区域，也可以是不同的的形状，如线状，圆形，十字形，圆环形等。

中值滤波法对消除椒盐噪声非常有效，在光学测量条纹图象的相位分析处理方法中有特殊作用，但在条纹中心分析方法中作用不大。中值滤波在图像处理中，常用于保护边缘信息,是经典的平滑噪声的方法。

要得到模板中数据的中间值，首先要将数据按大小排序，然后根据有序的数字序列来找中间值。中值滤波排序的过程有很多成熟的算法，如冒泡排序、二分排序等，大多是基于微机平台的软件算法，而适合硬件平台的排序算法则比较少。

FPGA 中值滤波实现方法

L(1,1)	L(1,2)	L(1,3)
L(2,1)	L(2,2)	L(2,3)
L(3,1)	L(3,2)	L(3,3)

如上所示，为一个 3x3 的图像模板，

第一步：

分别对三行像素进行排序：

由 L11,L12,L13 得到 L1max，L1mid，L1min；

由 L21,L22,L23 得到 L2max，L2mid，L2min；

3）由 L31,L32,L33 得到 L3max，L3mid，L3min。

第二步：

分别对三行像素中的 3 个最大，3 个中间和 3 个最小分别进行排序：

由 L1max，L2max，L3max 得到 Lmax_max，Lmax_mid，Lmax_min；

2）由 L1mid，L2mid，L3mid 得到 Lmid_max，Lmid_mid，Lmid_min；

3）由 Lmin，L2min，L3min 得到 Lmin_max，Lmin_mid，Lmin_min；

第三步：对最大的最小(Lmax_min)，中间的中间(Lmid_mid)以及最小的最大(Lmin_max)进行排序（例：由 Lmax_min，Lmid_mid，Lmin_max 得到 midian）。

FPGA 的算法实现步骤基本如此。

FPGA 实现

FPGA 平台搭建：

方法 1:

如下图 1 所示，通过 R/G/B 通道形成单色通道进入中值滤波器实现灰度图像的中值滤波的系统构建。

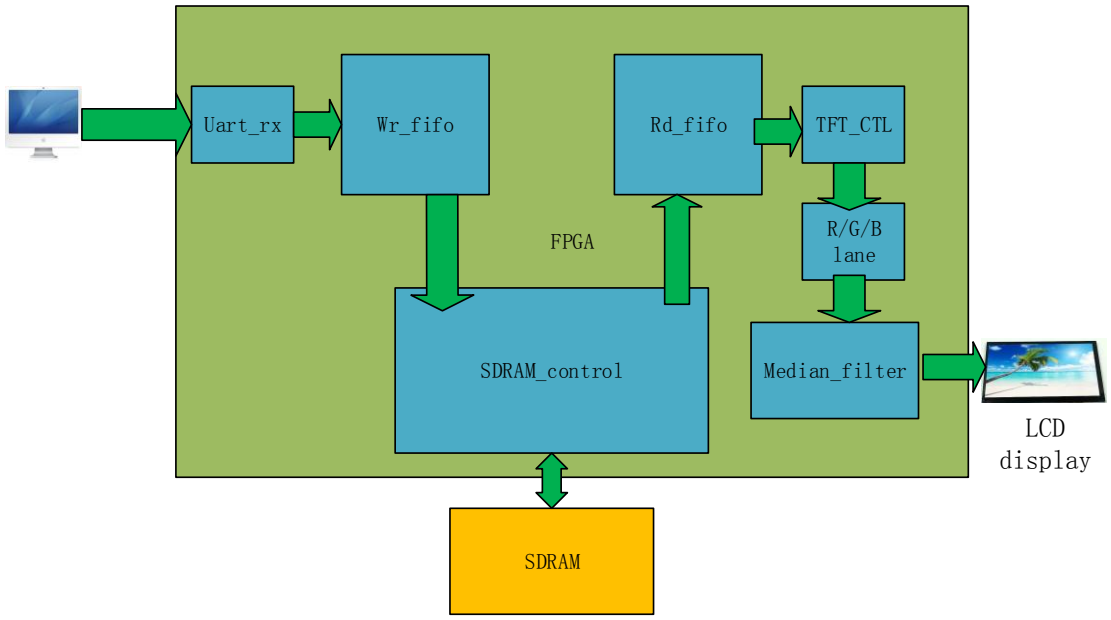


图 1 通过 R/G/B 单通道实现灰度图像

方法 2:

如下图 2 所示，首先将 RGB 图像转换成 Ycbcr 图像，Y 通道进入中值滤波器实现灰度图像的中值滤波。

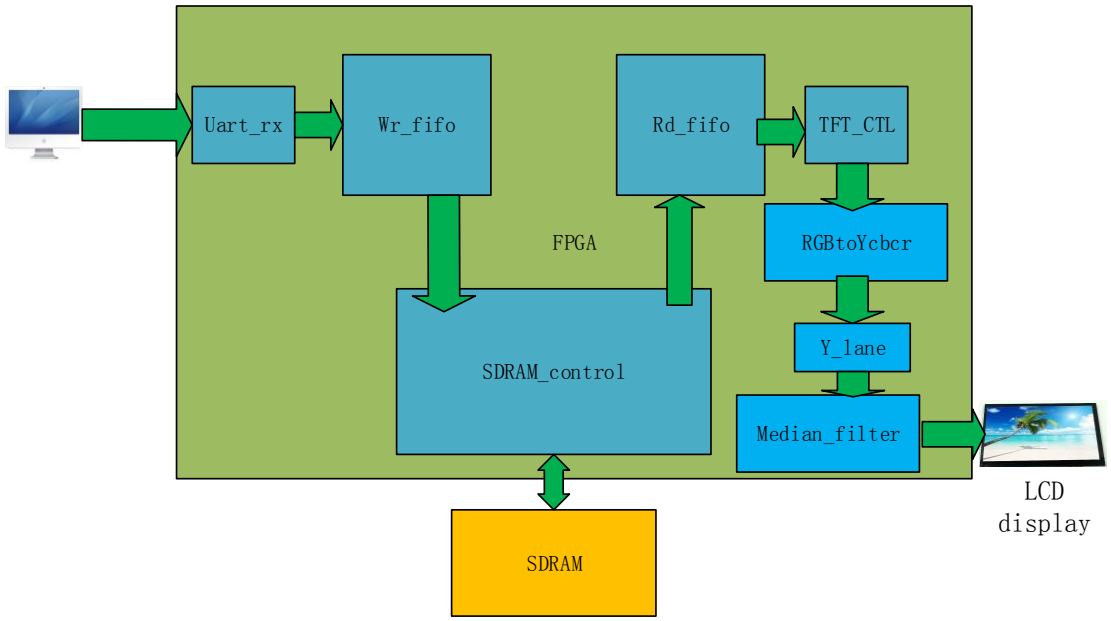


图 2 通过 Y 通道实现灰度图像

模块接口设计：

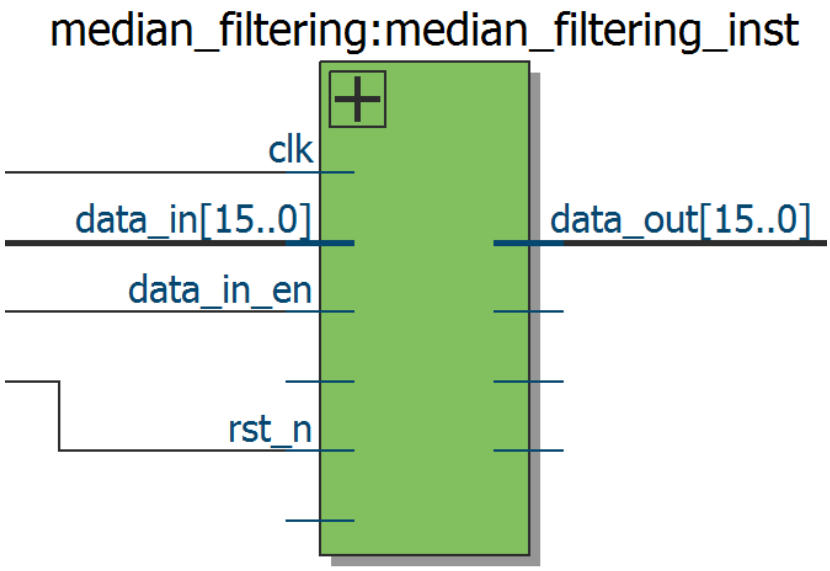


图 3 中值滤波模块

如图 4 所示，中值滤波器的端口如表 1 解释：

表 1 中值滤波器端口描述

端口名称	I/O	端口描述
clk	I	clk 为像素时钟(4.3 寸 lcd 为 9Mhz,5 寸为 33MHZ)
rst_n	I	rst_n 为系统复位信号
data_in	I [15:0]	data_in 为 16 位灰度像素输入通道
data_in_en	I	data_in_en 为 lcd 平显示有效区域使能信号
data_out	O	data_out 为 16 位灰度像素输出通道

源码：

```
/*
Module name:    median_filtering.v
```

```
Description:

Date:           2017/01/02
Engineer:       lipu
e-mail:         137194782@qq.com
*/
`timescale 1ns/1ps

module median_filtering(
        input               clk,
        input               rst_n,

        input [15:0]        data_in,
        input               data_in_en,
        input               hs_in,
        input               vs_in,

        output[15:0]        data_out,
        output              data_out_en,

        output              hs_out,
        output              vs_out
        );

wire [15:0] line0;
wire [15:0] line1;
wire [15:0] line2;

//-------------------------------------
//pipeline control signal
//-------------------------------------
reg         hs0;
reg         hs1;
reg         hs2;

reg         vs0;
reg         vs1;
reg         vs2;

reg         de0;
reg         de1;
reg         de2;
//-------------------------------------
//pipeline data
```

```
//-------------------------------------
reg [15:0] line0_data0;
reg [15:0] line0_data1;
reg [15:0] line0_data2;

reg [15:0] line1_data0;
reg [15:0] line1_data1;
reg [15:0] line1_data2;

reg [15:0] line2_data0;
reg [15:0] line2_data1;
reg [15:0] line2_data2;

//-------------------------------------
//define line max mid min
//-------------------------------------
reg [15:0] line0_max;
reg [15:0] line0_mid;
reg [15:0] line0_min;

reg [15:0] line1_max;
reg [15:0] line1_mid;
reg [15:0] line1_min;

reg [15:0] line2_max;
reg [15:0] line2_mid;
reg [15:0] line2_min;

//-------------------------------------------------------------
// define //max of min //mid of mid// min of max
//-------------------------------------------------------------

reg [15:0] max_max;
reg [15:0] max_mid;
reg [15:0] max_min;

reg [15:0] mid_max;
reg [15:0] mid_mid;
reg [15:0] mid_min;

reg [15:0] min_max;
reg [15:0] min_mid;
reg [15:0] min_min;
```

```
//-----------------------------------------------
// define mid of mid
//-----------------------------------------------

reg [15:0] mid;

line3x3 line3x3_inst(
            .clken(data_in_en),
             .clock(clk),
             .shiftin(data_in),
             .shiftout(),
             .taps0x(line0),
             .taps1x(line1),
             .taps2x(line2)
              );

//-------------------------------------------------
//delay control signal
//-------------------------------------------------
always @(posedge clk or negedge rst_n) begin
   if(!rst_n) begin
      hs0 <= 1'b0;
      hs1 <= 1'b0;
      hs2 <= 1'b0;

      vs0 <= 1'b0;
      vs1 <= 1'b0;
      vs2 <= 1'b0;

      de0 <= 1'b0;
      de1 <= 1'b0;
      de2 <= 1'b0;
   end
   else if(data_in_en) begin
      hs0 <= hs_in;
      hs1 <= hs0;
      hs2 <= hs1;

      vs0 <= vs_in;
      vs1 <= vs0;
      vs2 <= vs1;

      de0 <= data_in_en;
      de1 <= de0;
```

```
        de2 <= de1;
    end
end
//-------------------------------------------------------------------
// Form an image matrix of three multiplied by three
//-------------------------------------------------------------------
always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        line0_data0 <= 16'b0;
        line0_data1 <= 16'b0;
        line0_data2 <= 16'b0;

        line1_data0 <= 16'b0;
        line1_data1 <= 16'b0;
        line1_data2 <= 16'b0;

        line2_data0 <= 16'b0;
        line2_data1 <= 16'b0;
        line2_data2 <= 16'b0;
    end
    else if(data_in_en) begin
        line0_data0 <= line0;
        line0_data1 <= line0_data0;
        line0_data2 <= line0_data1;

        line1_data0 <= line1;
        line1_data1 <= line1_data0;
        line1_data2 <= line1_data1;

        line2_data0 <= line2;
        line2_data1 <= line2_data0;
        line2_data2 <= line2_data1;
    end
    else ;
end
//-------------------------------------------------------------------------------
//(line0 line1 line2) of (max mid min)
//-------------------------------------------------------------------------------
always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        line0_max <= 16'd0;
        line0_mid <= 16'd0;
        line0_min <= 16'd0;
    end
```

```
    else if(data_in_en) begin
        if((line0_data0 >= line0_data1) && (line0_data0 >= line0_data2)) begin
            line0_max <= line0_data0;
            if(line0_data1 >= line0_data2) begin
                line0_mid <= line0_data1;
                line0_min <= line0_data2;
            end
            else begin
                line0_mid <= line0_data2;
                line0_min <= line0_data1;
            end
        end
        else if((line0_data1 > line0_data0) && (line0_data1 >= line0_data2)) begin
            line0_max <= line0_data1;
            if(line0_data0 >= line0_data2) begin
                line0_mid <= line0_data0;
                line0_min <= line0_data2;
            end
            else begin
                line0_mid <= line0_data2;
                line0_min <= line0_data0;
            end
        end
        else if((line0_data2 > line0_data0) && (line0_data2 > line0_data1)) begin
            line0_max <= line0_data2;
            if(line0_data0 >= line0_data1) begin
                line0_mid <= line0_data0;
                line0_min <= line0_data1;
            end
            else begin
                line0_mid <= line0_data1;
                line0_min <= line0_data0;
            end
        end
    end
end

always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        line1_max <= 16'd0;
        line1_mid <= 16'd0;
        line1_min <= 16'd0;
    end
    else if(data_in_en) begin
```

```
        if((line1_data0 >= line1_data1) && (line1_data0 >= line1_data2)) begin
            line1_max <= line1_data0;
             if(line1_data1 >= line1_data2) begin
                line1_mid <= line1_data1;
                line1_min <= line1_data2;
             end
             else begin
                line1_mid <= line1_data2;
                line1_min <= line1_data1;
             end
        end
        else if((line1_data1 > line1_data0) && (line1_data1 >= line1_data2)) begin
            line1_max <= line1_data1;
             if(line1_data0 >= line1_data2) begin
                line1_mid <= line1_data0;
                line1_min <= line1_data2;
             end
             else begin
                line1_mid <= line1_data2;
                line1_min <= line1_data0;
             end
        end
        else if((line1_data2 > line1_data0) && (line1_data2 > line1_data1)) begin
            line1_max <= line1_data2;
             if(line1_data0 >= line1_data1) begin
                line1_mid <= line1_data0;
                line1_min <= line1_data1;
             end
             else begin
                line1_mid <= line1_data1;
                line1_min <= line1_data0;
             end
        end
    end
end

always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
         line2_max <= 16'd0;
        line2_mid <= 16'd0;
        line2_min <= 16'd0;
    end
    else if(data_in_en) begin
        if((line2_data0 >= line2_data1) && (line2_data0 >= line2_data2)) begin
```

```
            line2_max <= line2_data0;
             if(line2_data1 > line2_data2) begin
                line2_mid <= line2_data1;
                line2_min <= line2_data2;
             end
             else begin
                line2_mid <= line2_data2;
                line2_min <= line2_data1;
             end
          end
          else if((line2_data1 > line2_data0) && (line2_data1 >= line2_data2)) begin
             line2_max <= line2_data1;
             if(line2_data0 >= line2_data2) begin
                line2_mid <= line2_data0;
                line2_min <= line2_data2;
             end
             else begin
                line2_mid <= line2_data2;
                line2_min <= line2_data0;
             end
          end
          else if((line2_data2 > line2_data0) && (line2_data2 > line2_data1)) begin
             line2_max <= line2_data2;
             if(line2_data0 >= line2_data1) begin
                line2_mid <= line2_data0;
                line2_min <= line2_data1;
             end
             else begin
                line2_mid <= line2_data1;
                line2_min <= line2_data0;
             end
          end
       end
    end
    //-------------------------------------------------------------------------
    // (max_max max_mid max_min) of ((line0 line1 line2) of max)
    //-------------------------------------------------------------------------
    always @(posedge clk or negedge rst_n) begin
       if(!rst_n) begin
          max_max <= 16'd0;
         max_mid <= 16'd0;
         max_min <= 16'd0;
       end
       else if(data_in_en) begin
```

```
      if((line0_max >= line1_max) && (line0_max >= line2_max)) begin
         max_max <= line0_max;
          if(line1_max >= line2_max) begin
             max_mid <= line1_max;
             max_min <= line2_max;
          end
          else begin
             max_mid <= line2_max;
             max_min <= line1_max;
          end
       end
       else if((line1_max > line0_max) && (line1_max >= line2_max)) begin
         max_max <= line1_max;
          if(line0_max >= line2_max) begin
             max_mid <= line0_max;
             max_min <= line2_max;
          end
          else begin
             max_mid <= line2_max;
             max_min <= line0_max;
          end
       end
       else if((line2_max > line0_max) && (line2_max > line1_max)) begin
         max_max <= line2_max;
          if(line0_max >= line1_max) begin
             max_mid <= line0_max;
             max_min <= line1_max;
          end
          else begin
             max_mid <= line1_max;
             max_min <= line0_max;
          end
       end
   end
end
//----------------------------------------------------------------------------
// (mid_max mid_mid mid_min) of ((line0 line1 line2)of mid)
//----------------------------------------------------------------------------
always @(posedge clk or negedge rst_n) begin
   if(!rst_n) begin
       mid_max <= 16'd0;
      mid_mid <= 16'd0;
      mid_min <= 16'd0;
   end
```

```
    else if(data_in_en) begin
        if((line0_mid >= line1_mid) && (line0_mid >= line2_mid)) begin
            mid_max <= line0_mid;
             if(line1_mid >= line2_mid) begin
                mid_mid <= line1_mid;
                mid_min <= line2_mid;
             end
             else begin
                mid_mid <= line2_mid;
                mid_min <= line1_mid;
             end
        end
        else if((line1_mid > line0_mid) && (line1_mid >= line2_mid)) begin
            mid_mid <= line1_mid;
             if(line0_mid >= line2_mid) begin
                mid_mid <= line0_mid;
                mid_min <= line2_mid;
             end
             else begin
                mid_mid <= line2_mid;
                mid_min <= line0_mid;
             end
        end
        else if((line2_mid > line0_mid) && (line2_mid > line1_mid)) begin
            mid_max <= line2_mid;
             if(line0_mid >= line1_mid) begin
                mid_mid <= line0_mid;
                mid_min <= line1_mid;
             end
             else begin
                mid_mid <= line1_mid;
                mid_min <= line0_mid;
             end
        end
    end
end
//-------------------------------------------------------------------------
// (min_max min_mid min_min) of ((line0 line1 line2)of min)
//-------------------------------------------------------------------------
always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
         min_max <= 16'd0;
        min_mid <= 16'd0;
        min_min <= 16'd0;
```

```
    end
    else if(data_in_en) begin
        if((line0_min >= line1_min) && (line0_min >= line2_min)) begin
            min_max <= line0_min;
             if(line1_min >= line2_min) begin
                min_mid <= line1_min;
                min_min <= line2_min;
             end
             else begin
                min_mid <= line2_min;
                min_min <= line1_min;
             end
        end
        else if((line1_min > line0_min) && (line1_min >= line2_min)) begin
            min_max <= line1_min;
             if(line0_min >= line2_min) begin
                min_mid <= line0_min;
                min_min <= line2_min;
             end
             else begin
                min_mid <= line2_min;
                min_min <= line0_min;
             end
        end
        else if((line2_min > line0_min) && (line2_min > line1_min)) begin
            min_max <= line2_min;
             if(line0_min >= line1_min) begin
                min_mid <= line0_min;
                min_min <= line1_min;
             end
             else begin
                min_mid <= line1_min;
                min_min <= line0_min;
             end
        end
    end
end
//-------------------------------------------------------------------------
// middle
//-------------------------------------------------------------------------
always @(posedge clk or negedge rst_n) begin
    if(!rst_n)
        mid <= 16'd0;
    else if(data_in_en) begin
```

```
      if(((max_mid >= mid_mid) && (max_mid < min_mid)) || ((max_mid >= min_mid) &&
(max_mid < mid_mid)))
          mid <= max_mid;
       else if(((mid_mid > max_mid) && (mid_mid < min_mid)) || ((min_mid >= min_mid) &&
(mid_mid < max_mid)))
          mid <= mid_mid;
       else if(((min_mid > max_mid) && (min_mid < mid_mid)) || ((min_mid > mid_mid) &&
(mid_min < max_mid)))
          mid <= min_mid;
   end
   else ;
end
//----------------------------------------------------------------------------------------------------
//result
//----------------------------------------------------------------------------------------------------
assign data_out = mid;
assign data_out_en = de2;
assign hs_out = hs2;
assign vs_out = vs2;
endmodule
```

测试代码：

```
 /*
Module name:    median_filter_3x3_tb.v
Description:

Date:            2017/01/02
Engineer:        lipu
e-mail:          137194782@qq.com
*/
`timescale 1ns/1ps
`define CLK_PERIOD 20
module median_filter_3x3_tb();

reg              clk;
reg              rst_n;
reg [15:0]     data_in;
wire             data_in_en;

wire [15:0]    data_out;
wire             data_out_en;
```

```
median_filtering   median_filtering_inst(
                       .clk(clk),
                       .rst_n(rst_n),

                       .data_in(data_in),
                       .data_in_en(data_in_en),

                       .data_out(data_out),
                       .data_out_en(data_out_en)
            );

initial begin
   clk = 0;
   rst_n = 0;
   #(`CLK_PERIOD*10);
   rst_n = 1;
   #(`CLK_PERIOD*10000);
   $stop;
end

always #(`CLK_PERIOD/2) clk = ~clk;
assign data_in_en = rst_n;
always @(posedge clk or negedge rst_n) begin
   if(!rst_n)
      data_in <= 16'd0;
   else if(data_in_en)
       if(data_in == 16'd479)
          data_in <= 16'd0;
       else
          data_in <= data_in + 16'd1;
   else
     ;
end
endmodule
```

IP 设置

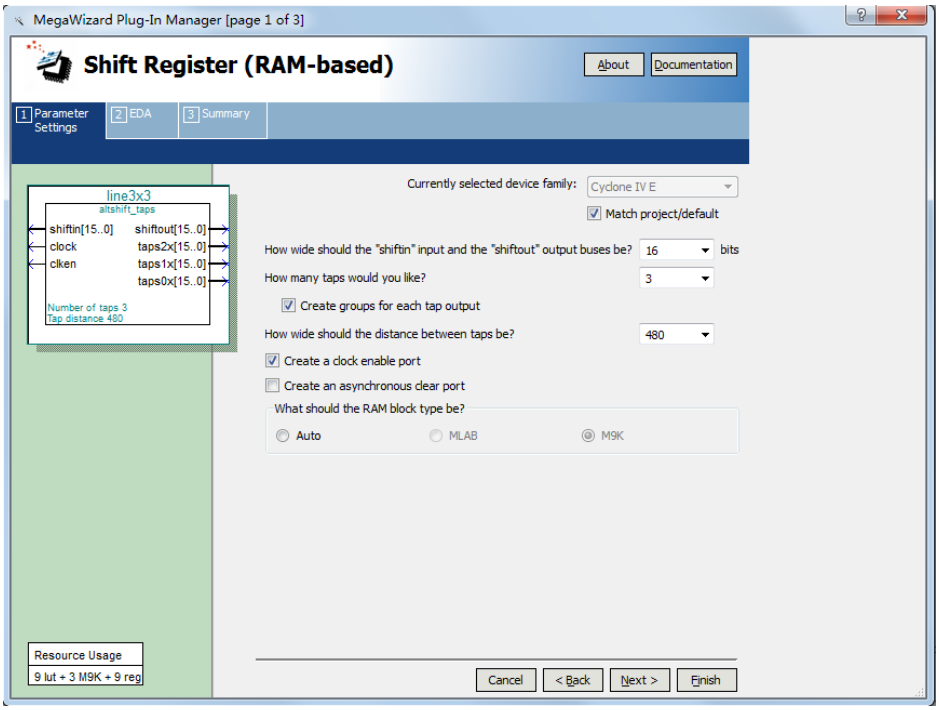


图 4 line3x3 IP

如图 4 所示，shift register(RAM-based)ip 主要为了形成三行像素缓存。

4.3 寸 TFT 显示屏的行缓存大小为 480，5 寸 TFT 显示屏的行缓存大小为 800。

仿真结果分析

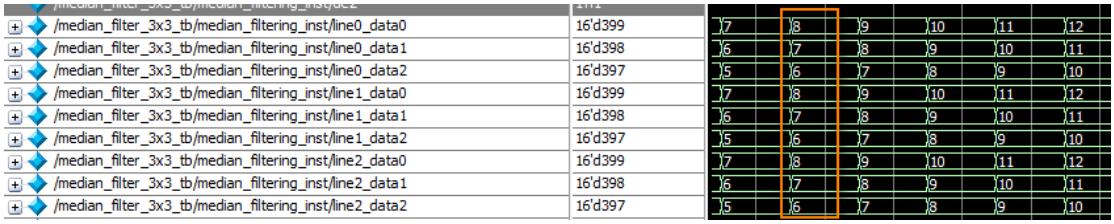


图 5 形成 3x3 像素矩阵

由图 5 可知，形成了以（x,y）为中心的 3x3 像素矩阵。

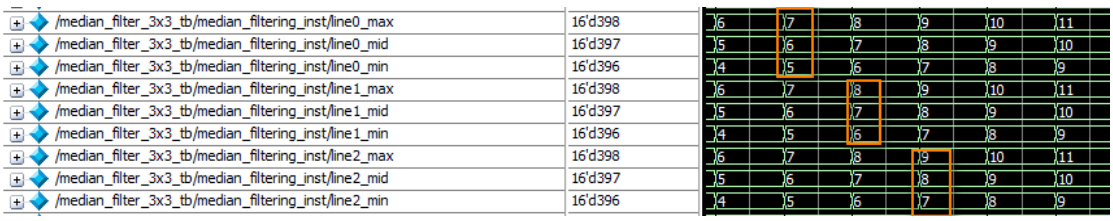


图 6 不同时间段的排序结果

由图 6 可知，对三行像素分别进行排序得到最大，中间和最小值。

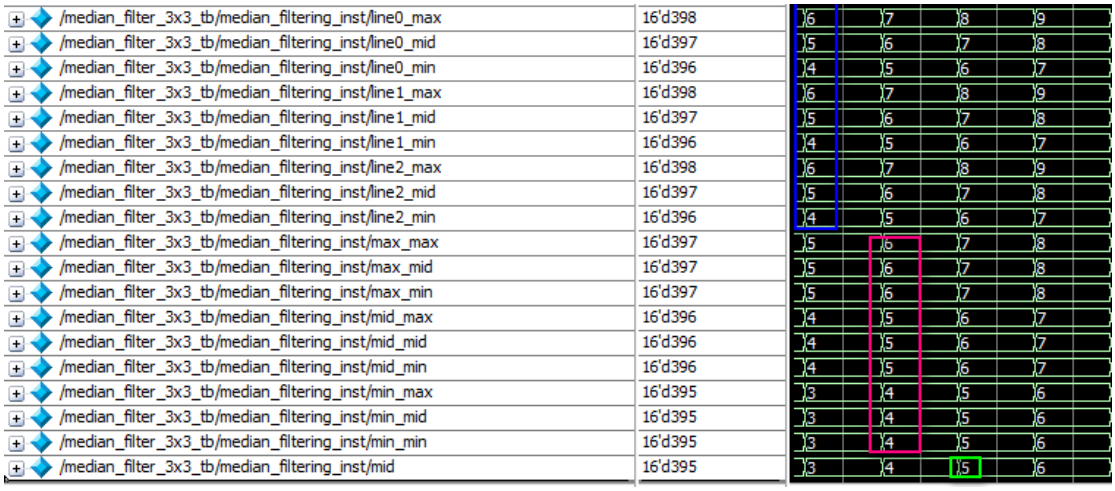


图 7 完成最终的计算结果

由图 7 可知，我们得到了最大中的最小，中间的中间，最小的最大。最后得到这三个的中间值完成中值的查找（由 Lmax_min，Lmid_mid，Lmin_max 得到 midian）。

实验结果：

图 8 未经处理的原始图像

图 9 加入椒盐噪声的灰度图像显示

图 10 经过一次中值滤波后的灰度图像显示

结果分析：

由图 9 和图 10 比较，图 10 去除了大部分椒盐噪声，并没有完全去除，这是因为有些椒盐噪声噪点比较大的原因。我们可以采取二次中值滤波，或者三次中值滤波，这样去除椒盐噪声的效果会更好。

此外中值滤波是可以处理 RGB 图像格式的，失真不是很严重，相比较均值和高斯滤波就不能处理 RGB 图像，大家可以多试验。

大家也可以尝试 5x5 中值滤波模板，看看效果是否会更好。

基于 FPGA 的灰度图像均值滤波算法的实现

作者：lee 神

背景知识

均值滤波是典型的线性滤波算法，它是指在图像上对目标像素给一个模板，该模板包括了其周围的临近像素（以目标像素为中心的周围 8 个像素，构成一个滤波模板，即去掉目标像素本身），再用模板中的全体像素的平均值来代替原来像素值。

均值滤波也称为线性滤波，其采用的主要方法为邻域平均法。线性滤波的基本原理是用均值代替原图像中的各个像素值，即对待处理的当前像素点（x，y），选择一个模板，该模板由其近邻的若干像素组成，求模板中所有像素的均值，再把该均值赋予当前像素点（x，y），作为处理后图像在该点上的灰度 g（x，y），即 g（x，y）=1/m ∑f（x，y） m 为该模板中包含当前像素在内的像素总个数。

均值滤波本身存在着固有的缺陷，即它不能很好地保护图像细节，在图像去噪的同时也破坏了图像的细节部分，从而使图像变得模糊，不能很好地去除噪声点。

FPGA 的均值滤波算法实现步骤

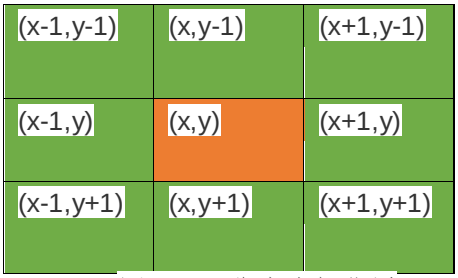


图 1 3x3 像素坐标位置

如图 1 所示，中心点（x，y）为均值滤波将要处理的位置，

f(x,y)表示（x,y）点的像素值，

g(x,y)表示（x,y）点经过均值处理后的值。

均值滤波公式表示如下：

g(x,y)=1/8*(f(x-1,y-1)+f(x,y-1)+f(x+1,y-1)
+f(x-1,y)+f(x+1,y)+
f(x-1,y+1)+f(x,y+1)+f(x+1,y+1))----------(1)

由（1）式我们看出(x,y)点的 3x3 像素点的均值等于其周围邻域的八个点的像素值之和除以 8。

FPGA 实现步骤：

1>形成 3x3 矩阵像素

2>求周围邻域八个点的像素值之和

3>将结果右移三位（相当于除以 8）得到结果。

FPGA 实现

FPGA 平台搭建

方法 1:

如下图 2 所示，通过 R/G/B 通道形成单色通道进入均值滤波器实现灰度图像的均值滤波的系统构建。

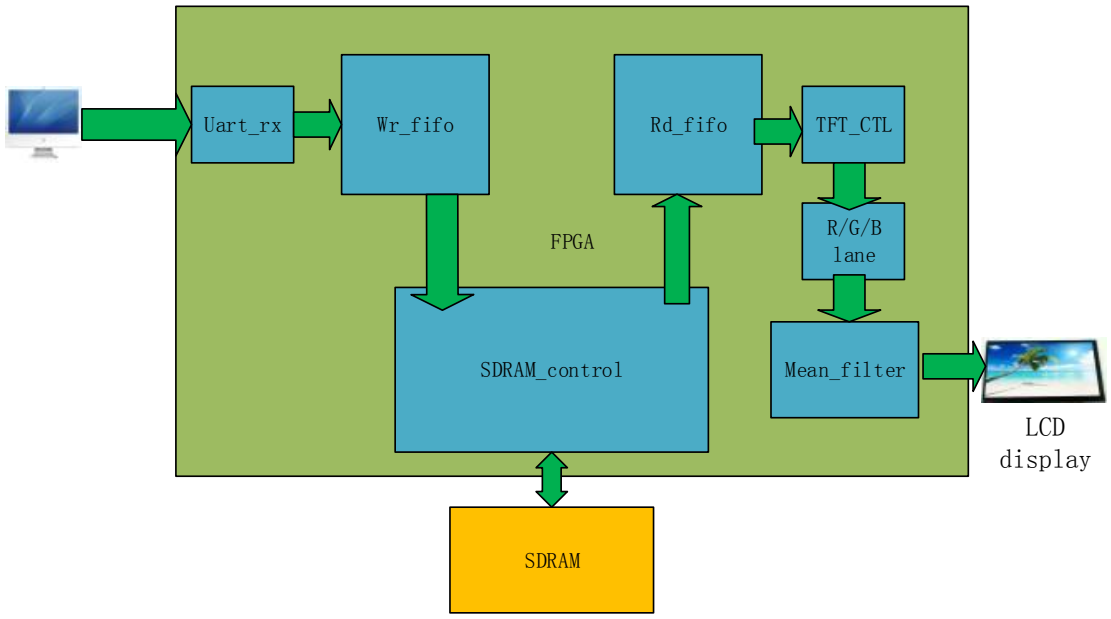


图 2 通过 R/G/B 单通道实现灰度图像

方法 2：

如下图 3 所示，首先将 RGB 图像转换成 Ycbcr 图像，Y 通道进入均值滤波器实现灰度图像的均值滤波。

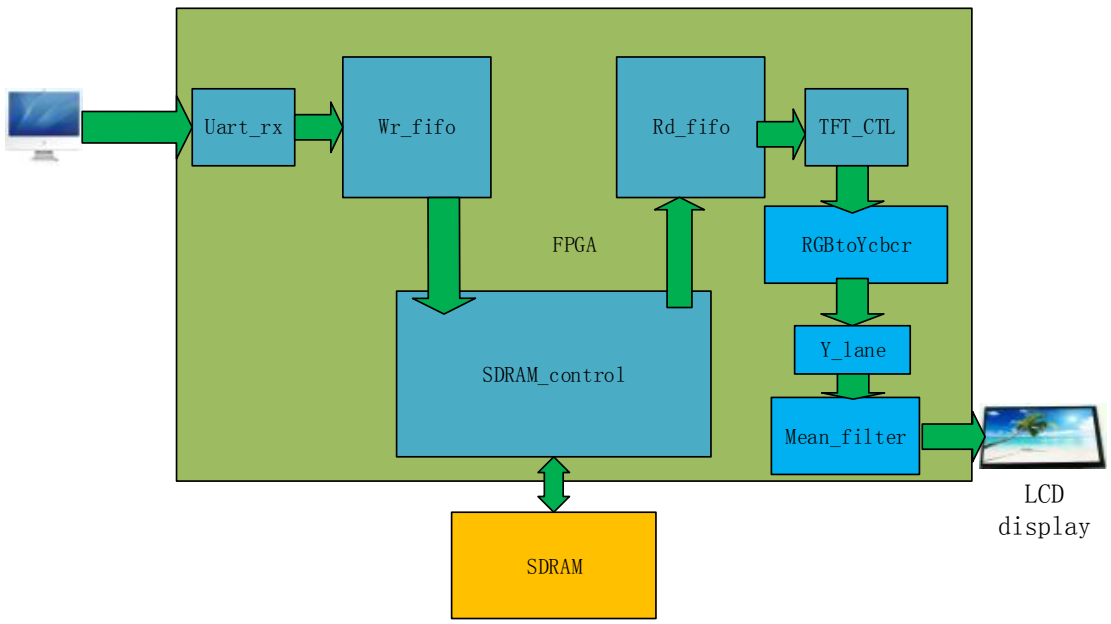


图 3 通过 Y 通道实现灰度图像

模块接口设计：

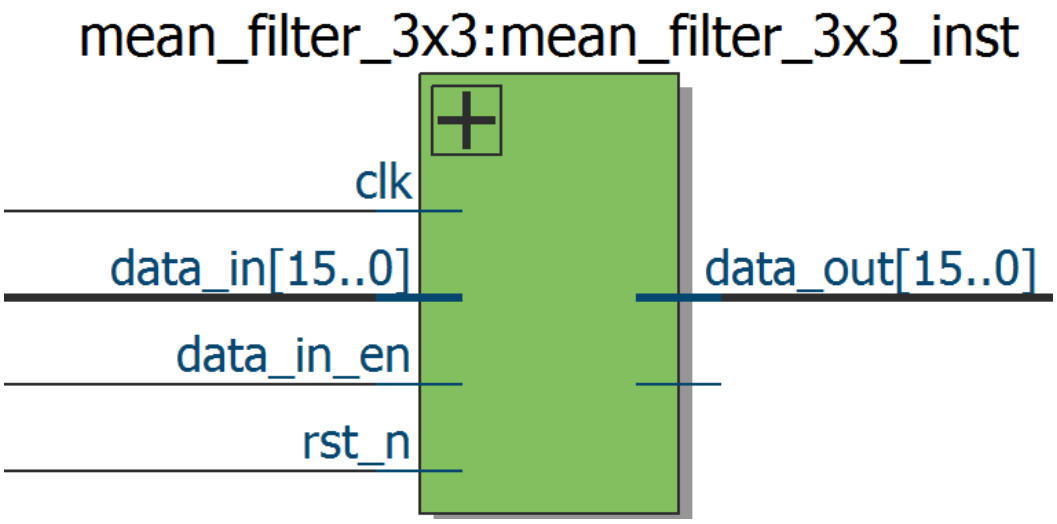


图 4 均值滤波模块

如图 4 所示，均值滤波器的端口如表 1 解释：

表 1 均值滤波器端口描述

端口名称	I/O	端口描述
clk	I	clk 为像素时钟（4.3 寸 lcd 为 9Mhz,5 寸为 33MHZ）
rst_n	I	rst_n 为系统复位信号
data_in	I [15:0]	data_in 为 16 位灰度像素输入通道
data_in_en	I	data_in_en 为 lcd 平显示有效区域使能信号
data_out	O	data_out 为 16 位灰度像素输出通道

均值滤波源码：

```
/*
Module name:    mean_filter_3x3.v
Description:    3x3 matrix operator mean filter

Date:           2017/12/22
Engineer:       lipu
*/
`timescale 1ns/1ps

module mean_filter_3x3(
        input           clk,    //lcd 控制时钟
        input           rst_n,
```

```
            input [15:0]          data_in, //16bit rgb 像素输入
            input                 data_in_en,//显示区域使能

            output   reg [15:0]data_out,//16bit rgb 像素输出
            output                data_out_en
            );
//-------------------------
//三行像素
//-------------------------
wire [15:0] line0;
wire [15:0] line1;
wire [15:0] line2;
//-------------------------
//3x3 像素矩阵
//-----------------------
reg [15:0] line0_data0;
reg [15:0] line0_data1;
reg [15:0] line0_data2;

reg [15:0] line1_data0;
reg [15:0] line1_data1;
reg [15:0] line1_data2;

reg [15:0] line2_data0;
reg [15:0] line2_data1;
reg [15:0] line2_data2;


wire [18:0]   result_data;//8 个点像素和
//-----------------------------------------------
//形成三行像素
//-----------------------------------------------
line3x3 line3x3_inst(
          .clken(data_in_en),
           .clock(clk),
           .shiftin(data_in),
           .shiftout(),
           .taps0x(line0),
           .taps1x(line1),
           .taps2x(line2)
            );
//---------------------------------------------------------------
// Form an image matrix of three multiplied by three
//---------------------------------------------------------------
```

```
always @(posedge clk or negedge rst_n) begin
   if(!rst_n) begin
      line0_data0 <= 16'b0;
       line0_data1 <= 16'b0;
       line0_data2 <= 16'b0;

       line1_data0 <= 16'b0;
       line1_data1 <= 16'b0;
       line1_data2 <= 16'b0;

       line2_data0 <= 16'b0;
       line2_data1 <= 16'b0;
       line2_data2 <= 16'b0;

       data_out_en0 <= 1'b0;
       data_out_en1 <= 1'b0;
       data_out_en2 <= 1'b0;
   end
   else if(data_in_en) begin
      line0_data0 <= line0;
       line0_data1 <= line0_data0;
       line0_data2 <= line0_data1;

       line1_data0 <= line1;
       line1_data1 <= line1_data0;
       line1_data2 <= line1_data1;

       line2_data0 <= line2;
       line2_data1 <= line2_data0;
       line2_data2 <= line2_data1;
   end
end

//------------------------------
// 8 个点平行加 IP
//------------------------------
pa_8 pa_8_inst(
         .clken(data_in_en),
         .clock(clk),
         .data0x(line0),
         .data1x(line0_data0),
         .data2x(line0_data1),
```

```
        .data3x(line1),
        .data4x(line1_data1),
        .data5x(line2),
        .data6x(line2_data0),
        .data7x(line2_data1),
        .result(result_data)
        );

//------------------------------------------------------------
// 结果输出，8 点之和除以 8，求得平均值
//------------------------------------------------------------
always @(posedge clk or negedge rst_n) begin
   if(!rst_n)
      data_out <= 16'b0;
   else if(data_in_en)
      data_out <= result_data[18:3];
       //data_out <= line2_data1;
   else ;
end

endmodule
```

测试源码：

```
/*
Module name:    mean_filter_3x3_tb.v
Description:

Date:           2017/12/22
Engineer:       lipu
e-mail:         137194782@qq.com
*/
`timescale 1ns/1ps
`define CLK_PERIOD 20
module mean_filter_3x3_tb();

reg clk;
reg rst_n;
reg [15:0] data_in;
reg data_in_en;

wire [15:0] data_out;
```

```
wire data_out_en;

mean_filter_3x3    mean_filter_3x3_inst(
                        .clk(clk),
                        .rst_n(rst_n),

                        .data_in(data_in),
                        .data_in_en(data_in_en),

                        .data_out(data_out),
                        .data_out_en(data_out_en)
                        );

initial begin
   clk = 0;
   rst_n = 0;
   data_in_en = 0;
   #(`CLK_PERIOD*10);
   rst_n = 1;
   #(`CLK_PERIOD*10);
   data_in_en = 1;
   #(`CLK_PERIOD*10000);
   $stop;
end

always #(`CLK_PERIOD/2)        clk = ~clk;
//-------------------------------------------------------------
//生成一行 480 个递增像素点
//-------------------------------------------------------------
always @(posedge clk or negedge rst_n) begin
   if(!rst_n)
      data_in <= 16'd0;
   else if(data_in_en)
      if(data_in == 16'd479)
         data_in <= 16'd0;
      else
         data_in <= data_in + 16'd1;
   else
      ;
end
endmodule
```

IP 设置

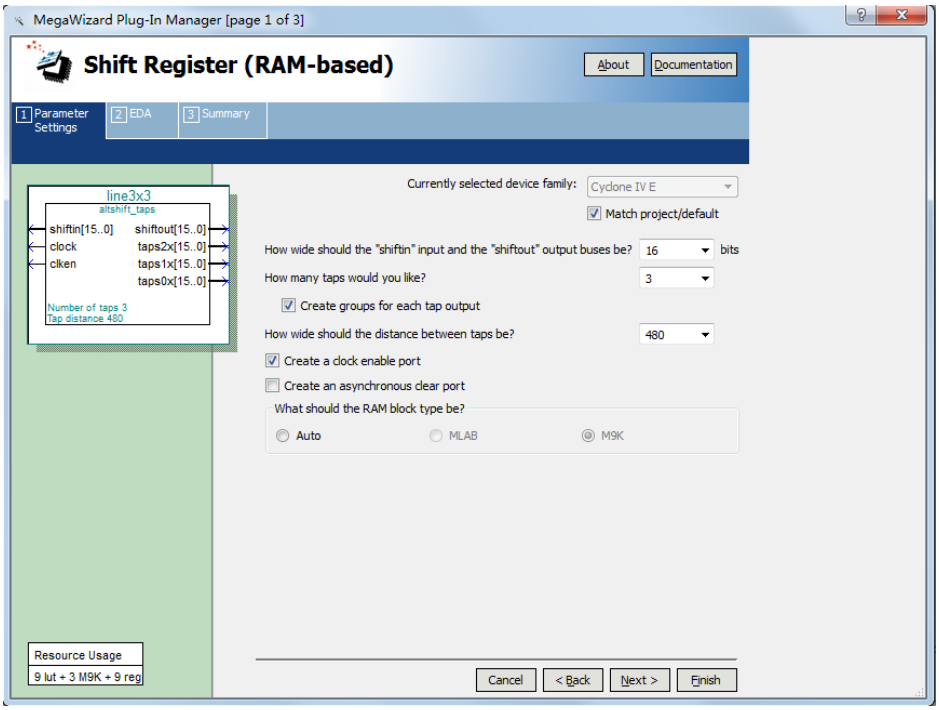


图 5 line3x3 IP

如图 5 所示，shift register(RAM-based)ip 主要为了形成三行像素缓存。4.3 寸 TFT 显示屏的行缓存大小为 480，5 寸 TFT 显示屏的行缓存大小为 800。

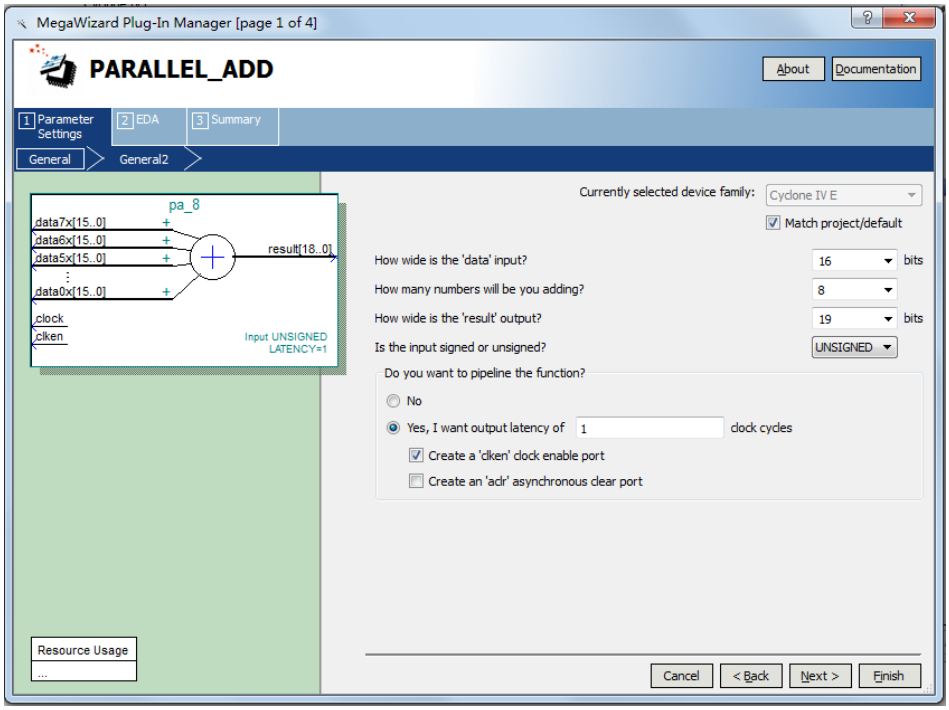


图 6 pa_8 IP

如图 6 所示，pa_8 为八个像素点点平行相加 IP。

仿真过程

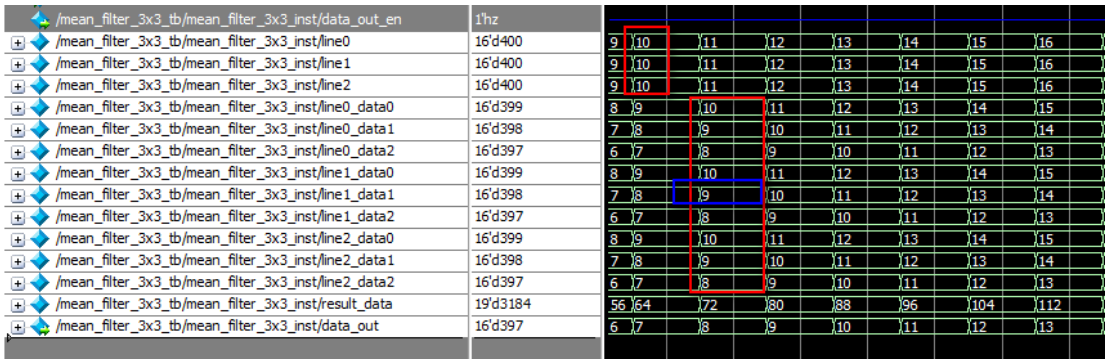


图 7　形成 3x3 矩阵像素仿真结果

如图 7，第一个小红框里边表示形成 3 行像素缓存，第二个大红框加小蓝框表示形成（x,y）周围的八个像素点。小蓝框表示（x,y）点像素值。

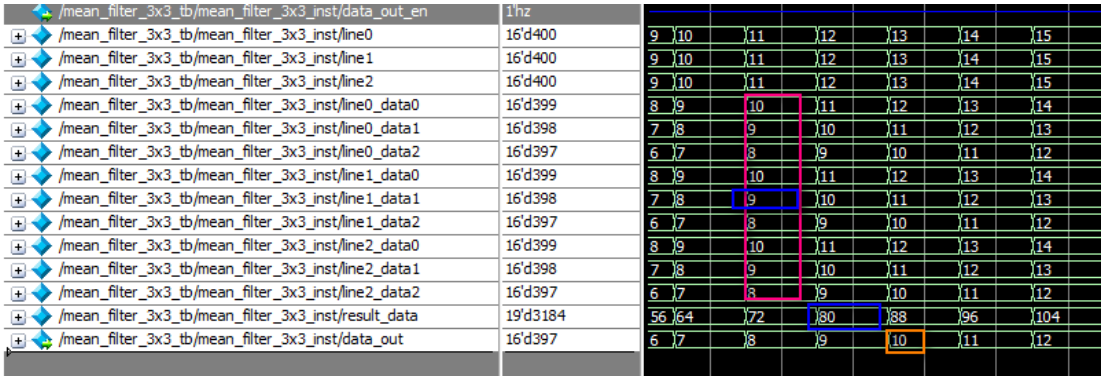


图 8　得到（x,y）点像素的均值

如图 8 所示，modelsim 仿真实现了像素 3x3 矩阵的输出，以（x,y）为中心，利用公式（1）10 + 9 + 8+10+8+10+9+8 = 80 最终求得（x,y）点的均值为 10，与图 8 结果相符。

实验结果

图 9　lena 原图

图 10 lena 未经均值滤波的灰度图像

图 11 经过均值滤波后的灰度图像

结果分析：

由图 9,10,11 实验结果来看，原始灰度图像的甚多细节被模糊化，实现了灰度图像的均值滤波。

基于 FPGA 灰度图像高斯滤波算法的实现

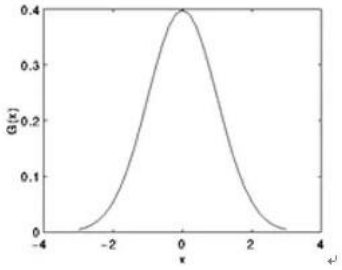
作者：lee 神

内容概要

高斯滤波是一种线性平滑滤波，适用于消除高斯噪声，广泛应用于图像处理的减噪过程。通俗的讲，高斯滤波就是对整幅图像进行加权平均的过程，每一个像素点的值，都由其本身和邻域内的其他像素值经过加权平均后得到。高斯滤波的具体操作是：用一个模板（或称卷积、掩模）扫描图像中的每一个像素，用模板确定的邻域内像素的加权平均灰度值去替代模板中心像素点的值。

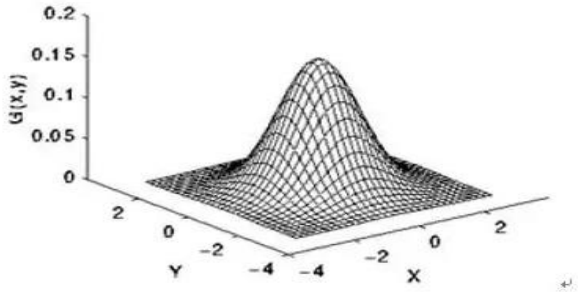
一位高斯分布：

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$



二位高斯分布：

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



高斯滤波后图像被平滑的程度取决于标准差。它的输出是临域像素的加权平均，同时离中心越近的像素权重越高。因此，相对于均值滤波（mean filter）它的平滑效果更柔和，而且边缘保留的也更好。

高斯滤波被用作为平滑滤波器的本质原因是因为它是一个低通滤波器，而且大部份基于卷积平滑滤波器都是低通滤波器。

GAUSS 滤波算法克服了边界效应，因而滤波后的图像较好。

高斯滤波算法实现步骤

(1/273) *

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

高斯滤波 5x5 算子

(1/16)*

1	2	1
2	4	2
1	2	1

高斯滤波 3x3 算子

1>串行像素形成 3x3 矩阵

(x-1,y-1)	(x,y-1)	(x+1,y-1)
(x-1,y)	(x,y)	(x+1,y)
(x-1,y+1)	(x,y+1)	(x+1,y+1)

f(x,y)表示（x,y）点的像素值；

g(x,y)表示（x,y）点经过高斯滤波处理后的值；

2>用模板确定的邻域内像素的加权平均灰度值去替代模板中心像素点的值

g(x,y)=（1/16）* (f(x-1,y-1)+2f(x,y-1)+f(x+1,y-1)

+2f(x-1,y)+4f(x,y) + 2f(x+1,y)+

f(x-1,y+1)+2f(x,y+1)+f(x+1,y+1))----------(1)

3> 用模板扫描图像中的每一个像素

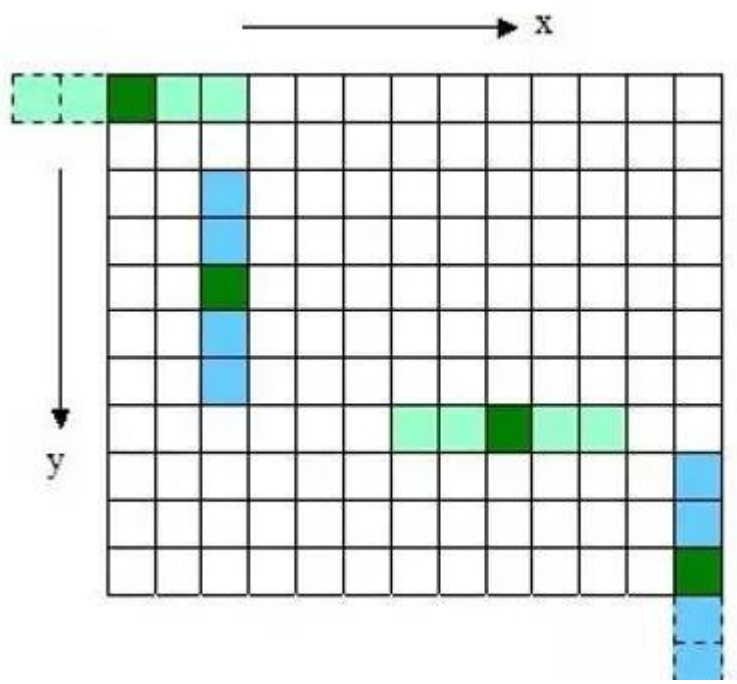


图 1 模板扫描像素点示意图

如图 1 所示，使用高斯滤波模板从屏幕的左上角（从左到右），没扫完一行换下一行（从上到下）扫完整个屏幕，最终完成一帧图的高斯滤波。

FPGA 实现

首先将 RGB 图像转换成 Gray 图像

方法 1：

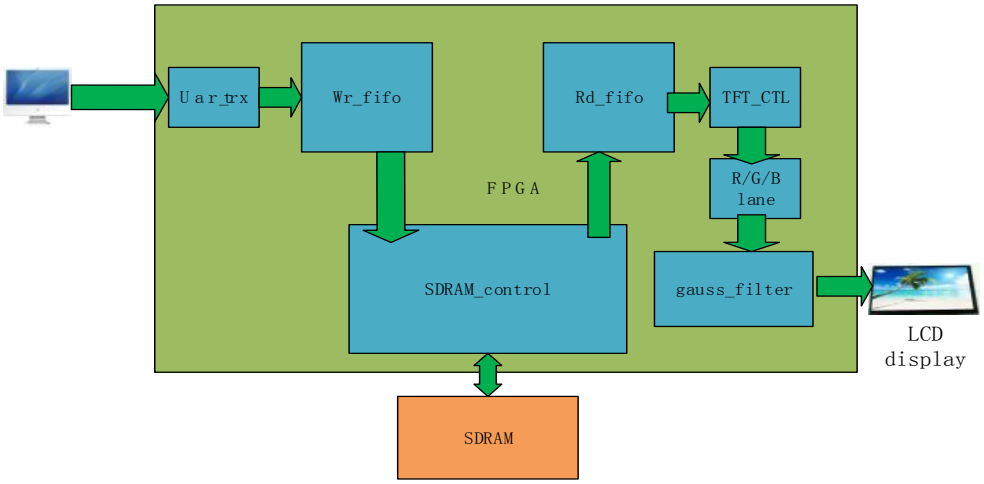


图 2 R/G/B lane 形成灰度图像进行高斯滤波

如图 2 所示，使用 R/G/B 单通道形成灰度图像进入高斯滤波模块。

方法二：

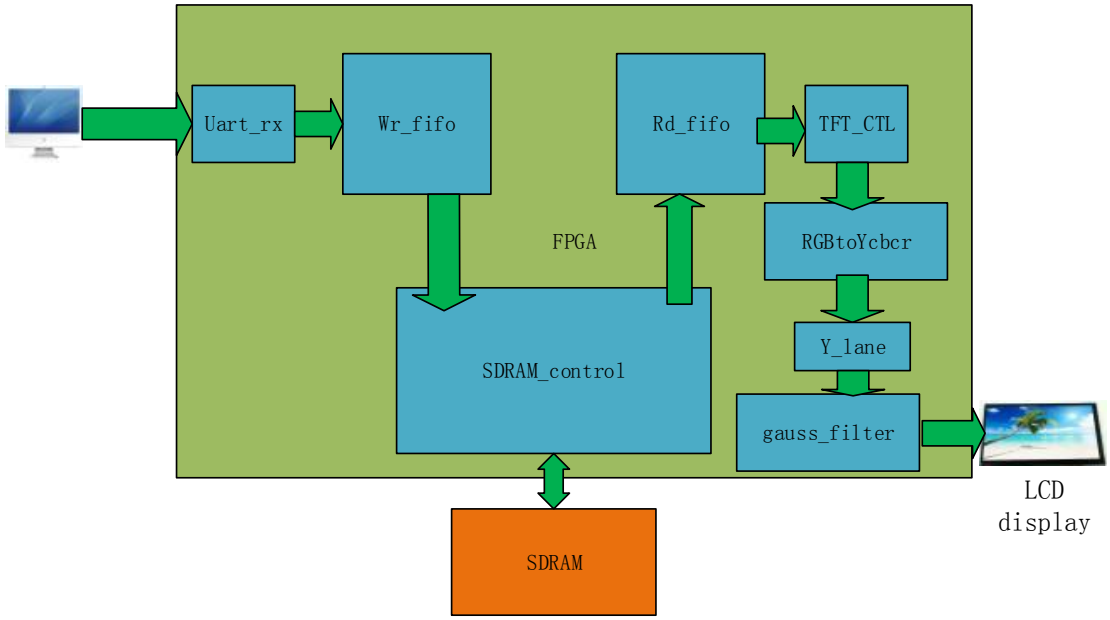


图 3　Y lane 形成灰度图像进行高斯滤波

如图 3 所示，使用 RGB 图像转换成 Ycbcr 的图像的 Y 通道形成灰度图像进入高斯滤波模块。

模块接口设计：

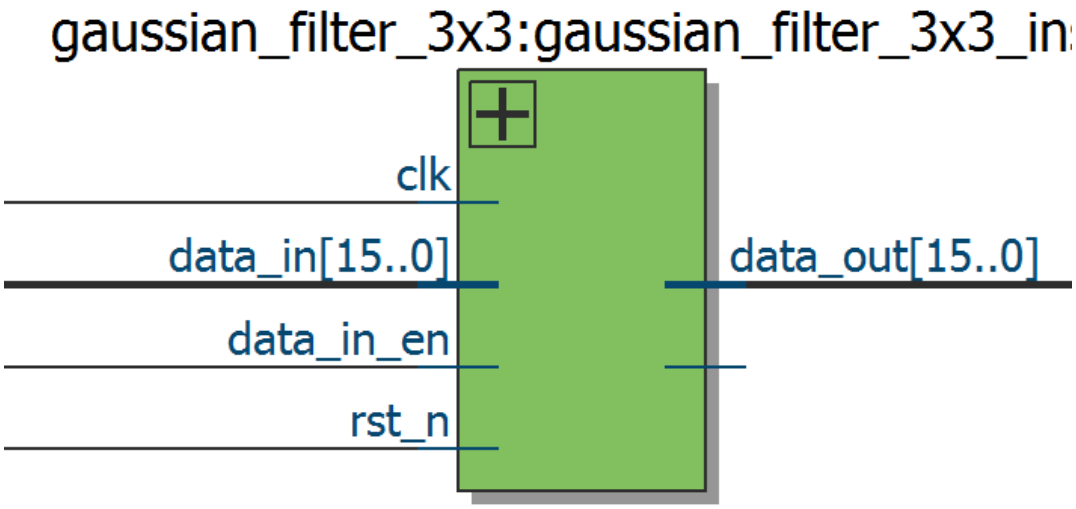


图 4　高斯滤波模块

如图 4 所示，高斯滤波模块的端口如表 1 解释：

表 1　高斯滤波端口描述

端口名称	I/O	端口描述
clk	I	clk 为像素时钟（4.3 寸 lcd 为 9Mhz,5 寸为 33MHZ）
rst_n	I	rst_n 为系统复位信号
data_in	I [15:0]	data_in 为 16 位灰度像素输入通道

data_in_en	I	data_in_en 为 lcd 平显示有效区域使能信号
data_out	O[15:0]	data_out 为 16 位灰度像素输出通道

源码：

```
/*
Module name:    gaussian_filter_3x3.v
Description:    Using the 3x3 Gauss filter operator to complete t
he Gauss filtering of gray image
Date:           2017/12/22
Engineer:       lipu
e-mail:         137194782@qq.com
*/
`timescale 1ns/1ps

module gaussian_filter_3x3(
            input               clk,    //pixel clk
            input               rst_n,

            input [15:0]        data_in, //16 bit RGB565 pixel
            input               data_in_en,

            output  reg [15:0]  data_out,
            output              data_out_en
            );
//----------------------------------
// 形成三行像素缓存
//----------------------------------
wire [15:0] line0;
wire [15:0] line1;
wire [15:0] line2;
//---------------------------------------
//形成 3x3 像素矩阵
//---------------------------------------
reg [15:0] line0_data0;
reg [15:0] line0_data1;
reg [15:0] line0_data2;

reg [15:0] line1_data0;
reg [15:0] line1_data1;
reg [15:0] line1_data2;
```

```
reg [15:0] line2_data0;
reg [15:0] line2_data1;
reg [15:0] line2_data2;
//-------------------------------------------
// shift_ram IP 形成三行像素缓存
//-------------------------------------------
line3x3 line3x3_inst(
            .clken(data_in_en),
             .clock(clk),
             .shiftin(data_in),
             .shiftout(),
             .taps0x(line0),
             .taps1x(line1),
             .taps2x(line2)
              );
//-------------------------------------------------------------------
// Form an image matrix of three multiplied by three
//-------------------------------------------------------------------

always @(posedge clk or negedge rst_n) begin
   if(!rst_n) begin
        line0_data0 <= 16'b0;
        line0_data1 <= 16'b0;
        line0_data2 <= 16'b0;

        line1_data0 <= 16'b0;
        line1_data1 <= 16'b0;
        line1_data2 <= 16'b0;

        line2_data0 <= 16'b0;
        line2_data1 <= 16'b0;
        line2_data2 <= 16'b0;
   end
   else if(data_in_en) begin    //像素有效信号
      line0_data0 <= line0;
        line0_data1 <= line0_data0;
        line0_data2 <= line0_data1;

        line1_data0 <= line1;
        line1_data1 <= line1_data0;
        line1_data2 <= line1_data1;

        line2_data0 <= line2;
```

```
        line2_data1 <= line2_data0;
        line2_data2 <= line2_data1;
    end
end

//----------------------------------------------------------------------------------------------------
// 计算最终结果
//----------------------------------------------------------------------------------------------------

always @(posedge clk or negedge rst_n) begin
    if(!rst_n)
        data_out <= 16'b0;
    else if(data_in_en)
        data_out <= (line0_data0 + line0_data1*2 + line0_data2 + line1_data0*2 +
line1_data1*4 + line1_data2*2 + line2_data0 + line2_data1*2 + line2_data2)>>4;
        //data_out <= line2_data1;
    else ;
end

endmodule
```

仿真代码：

```
/*
Module name:    gaussian_filter_3x3_tb.v
Description:    Generate 0-479 incrementing data to simulate 480 pixels per line.

Date:           2018/1/23
Engineer:       lipu
e-mail:         137194782@qq.com
*/
`timescale 1ns/1ps
`define CLK_PERIOD 20
module gaussian_filter_3x3_tb();

reg clk;
reg rst_n;
reg [15:0] data_in;
reg data_in_en;

wire [15:0] data_out;
wire data_out_en;
```

```
gaussian_filter_3x3   gaussian_filter_3x3_inst(
                        .clk(clk),
                        .rst_n(rst_n),

                        .data_in(data_in),
                        .data_in_en(data_in_en),

                        .data_out(data_out),
                        .data_out_en(data_out_en)
                        );

initial begin
   clk = 0;
   rst_n = 0;
   data_in_en = 0;
   #(`CLK_PERIOD*10);
   rst_n = 1;
   #(`CLK_PERIOD*10);
   data_in_en = 1;
   #(`CLK_PERIOD*10000);
   $stop;
end

always #(`CLK_PERIOD/2)        clk = ~clk;

always @(posedge clk or negedge rst_n) begin
   if(!rst_n)
      data_in <= 16'd0;
   else if(data_in_en)
      if(data_in == 16'd479)
         data_in <= 16'd0;
      else
         data_in <= data_in + 16'd1;
   else
      ;
end
endmodule
```

仿真结果：

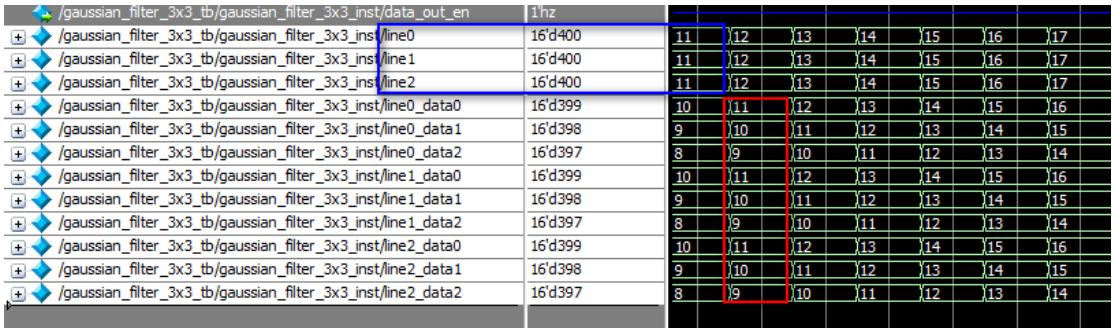


图 5 形成 3x3 的图像矩阵

由图 5 蓝色框数据可知，IP 缓存了三行一模一样的数据，形成了三行数据的缓存；

由红色框可知，一个时钟形成了（x,y）为中心的 3x3 像素矩阵。

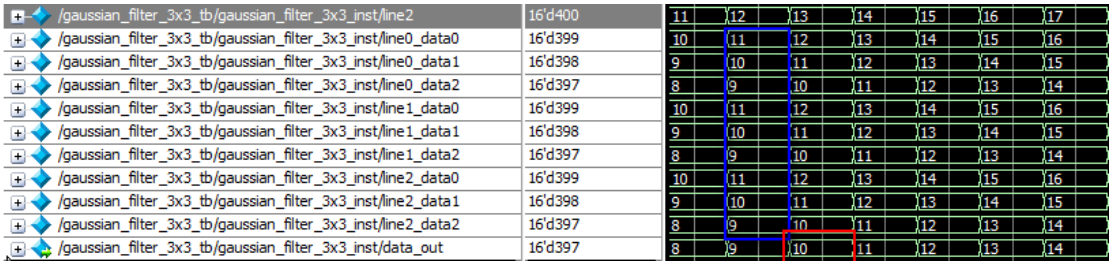


图 6 高斯滤波的计算结果

由图 6 的蓝色框我们可以知道九个像素点为[11,10,9;11,10,9;11,10,9]有公式（1）计算可得 10，与红色框里边的结果一致。

实验结果：

图 7 实验使用原图

图 8 灰度图像

图 9 灰度图像经过高斯滤波后的图像

总结：

至此，基于 FPGA 的三大图像滤波（均值滤波、中值滤波、高斯滤波）处理已经讲解完毕，其中的图像处理效果需要大家自己去实验，去对比。手机拍摄出来的毕竟有差距。

FPGA 在前端捕获到数据后首先要对视频图像做一个预处理，然后根据噪声的来源，针对椒盐噪声进行中值滤波，针对高斯噪声进行高斯滤波处理，均值滤波在图像处理中也很常见。

基于 FPGA 图像处理之 sobel 算子边缘检测算法的实现

背景知识

边缘检测是图像处理和计算机视觉中的基本问题，边缘检测的目的是标识数字图像中亮度变化明显的点。图像属性中的显著变化通常反映了属性的重要事件和变化。 这些包括（i）深度上的不连续、（ii）表面方向不连续、（iii）物质属性变化和（iv）场景照明变化。 边缘检测是图像处理和计算机视觉中，尤其是特征提取中的一个研究领域。

边缘检测算子

一阶：Roberts Cross 算子，Prewitt 算子，Sobel 算子， Kirsch 算子，罗盘算子；

二阶： Marr-Hildreth，在梯度方向的二阶导数过零点，Canny 算子，Laplacian 算子。今天我们要讲的是基于 Sobel 算子的边缘检测的 FPGA 算法的实现。

Sobel 算子实现

Sobel 算法是像素图像边缘检测中最重要的算子之一，在机器学习、数字媒体、计算机视觉等信息科技领域起着举足轻重的作用。在技术上，它是一个离散的一阶差分算子，用来计算图像亮度函数的一阶梯度之近似值。在图像的任何一点使用此算子，将会产生该点对应的梯度矢量或是其法矢量

Sobel 边缘检测算法比较简，实际应用中效率比 canny 边缘检测效率要高，但是边缘不如 Canny 检测的准确，但是很多实际应用的场合，sobel 边缘却是首选，尤其是对效率要求较高，而对细纹理不太关心的时候。

Sobel 边缘检测通常带有方向性，可以只检测竖直边缘或垂直边缘或都检测。

Sobel 算子 x 方向

-1	0	+1
-2	0	+2
-1	0	+1

+1	+2	+1

y 方向

0	0	0
-1	-2	-1

原始图像 f

（i-1，j-1）	(i,j-1)	(i+1,j-1)
(i-1,j)	(i,j)	(i+1,j)
(i-1,j+1)	(i,j+1)	(i+1,j+1)

实现步骤：

1. Gx = P ★ Sobelx　-- 原始图像与 Sobel 算子 X 方向卷积；
2. Gy = P★ Sobely　-- 原始图像与 Sobel 算子 Y 方向卷积；
3. $G=\sqrt{G_x^2+G_y^2}$；
4. 阈值比较形成边缘查找后的二值图像。

FPGA 实现

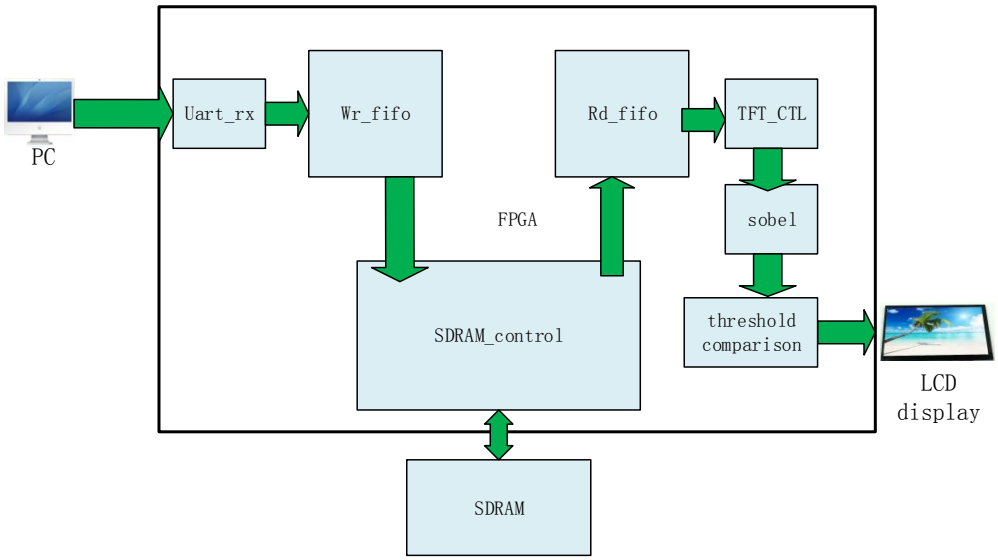


图 1 FPGA 基于串口传图实现 Sobel 算子边缘检测架构

由图 1 可知，在基于串口传图的基础上我们增加 sobel 模块以及阈值比较模块来实现边缘检测算法。

硬软件需求：

表 1 硬软件需求表

1	笔记本电脑/台式机	

2	串口线	
3	FPGA 开发板	小梅哥新航线 AC620
4	4.3/5 寸 TFT 显示屏	
5	Quartus II 13.0	
6	串口传图工具	
7	Picture2Hex	

Sobel 模块接口：

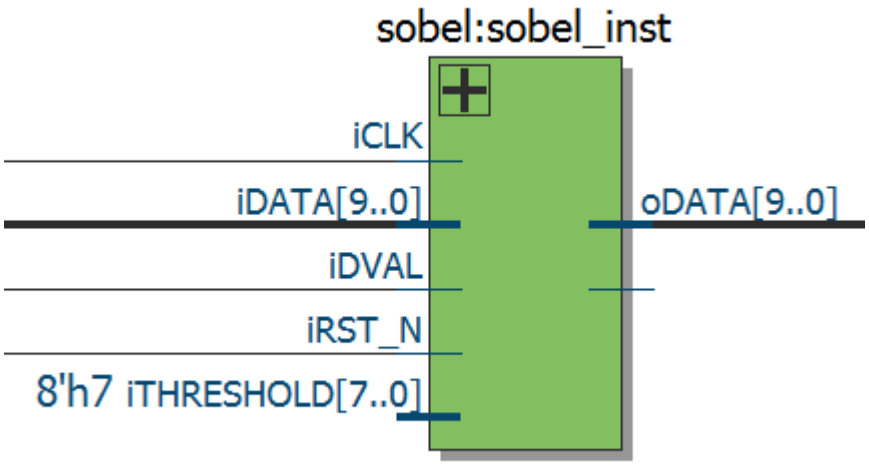


图 2 sobel 模块接口图

表 2 sobel 模块接口

序号	接口名称	接口描述
1	iclk	显示屏控制时钟
2	idata	单通道 G
3	idval	数据有效
4	Irst_n	系统复位
5	ithreshold	阈值
6	odata	像素输出

```
/*
Filename      : Sobel.v
Compiler      : Quartus II 13.0
Description : implement Sobel Edge Detector
*/

module sobel (
  input                 iCLK,
```

```
    input                iRST_N,
    input        [7:0] iTHRESHOLD,
    input                iDVAL,
    input        [9:0] iDATA,
    output reg           oDVAL,
    output reg [9:0] oDATA
);
//-----------------------------------------------------
// 将 Sobel 算子换算成有符号数（signed）
//-----------------------------------------------------
// mask x
parameter X1 = 8'hff, X2 = 8'h00, X3 = 8'h01;
parameter X4 = 8'hfe, X5 = 8'h00, X6 = 8'h02;
parameter X7 = 8'hff, X8 = 8'h00, X9 = 8'h01;

// mask y
parameter Y1 = 8'h01, Y2 = 8'h02, Y3 = 8'h01;
parameter Y4 = 8'h00, Y5 = 8'h00, Y6 = 8'h00;
parameter Y7 = 8'hff, Y8 = 8'hfe, Y9 = 8'hff;

wire   [7:0] Line0;
wire   [7:0] Line1;
wire   [7:0] Line2;

wire   [17:0]   Mac_x0;
wire   [17:0]   Mac_x1;
wire   [17:0]   Mac_x2;

wire   [17:0]   Mac_y0;
wire   [17:0]   Mac_y1;
wire   [17:0]   Mac_y2;

wire   [19:0]   Pa_x;
wire   [19:0]   Pa_y;

wire   [15:0]   Abs_mag;
//-----------------------------------------
// 实现 3x3 矩阵原始图像 P
//-----------------------------------------
LineBuffer LineBuffer_inst (
    .clken(iDVAL),
    .clock(iCLK),
    .shiftin(iDATA[9:2]),
    .taps0x(Line0),
```

```
    .taps1x(Line1),
    .taps2x(Line2)
);
//-----------------------------------------
// Gx = P ★ Sobelx
// x 方向卷积运算实现
//-----------------------------------------
MAC_3 x0 (
    .aclr3(!iRST_N),
    .clock0(iCLK),
    .dataa_0(Line0),
    .datab_0(X9),
    .datab_1(X8),
    .datab_2(X7),
    .result(Mac_x0)
);

MAC_3 x1 (
    .aclr3(!iRST_N),
    .clock0(iCLK),
    .dataa_0(Line1),
    .datab_0(X6),
    .datab_1(X5),
    .datab_2(X4),
    .result(Mac_x1)
);

MAC_3 x2 (
    .aclr3(!iRST_N),
    .clock0(iCLK),
    .dataa_0(Line2),
    .datab_0(X3),
    .datab_1(X2),
    .datab_2(X1),
    .result(Mac_x2)
);
PA_3 pa0 (
    .clock(iCLK),
    .data0x(Mac_x0),
    .data1x(Mac_x1),
    .data2x(Mac_x2),
    .result(Pa_x)
);
```

```
//-------------------------------------------------
// Gy = P★ Sobely
// y 方向卷积运算的实现
//-------------------------------------------------
// Y
MAC_3 y0 (
   .aclr3(!iRST_N),
   .clock0(iCLK),
   .dataa_0(Line0),
   .datab_0(Y9),
   .datab_1(Y8),
   .datab_2(Y7),
   .result(Mac_y0)
);

MAC_3 y1 (
   .aclr3(!iRST_N),
   .clock0(iCLK),
   .dataa_0(Line1),
   .datab_0(Y6),
   .datab_1(Y5),
   .datab_2(Y4),
   .result(Mac_y1)
);

MAC_3 y2 (
   .aclr3(!iRST_N),
   .clock0(iCLK),
   .dataa_0(Line2),
   .datab_0(Y3),
   .datab_1(Y2),
   .datab_2(Y1),
   .result(Mac_y2)
);
PA_3 pa1 (
   .clock(iCLK),
   .data0x(Mac_y0),
   .data1x(Mac_y1),
   .data2x(Mac_y2),
   .result(Pa_y)
);
//-------------------------------------------
// 得到 G
//-------------------------------------------
```

```
SQRT sqrt0 (
    .clk(iCLK),
    .radical(Pa_x * Pa_x + Pa_y * Pa_y),
    .q(Abs_mag)
);
//-----------------------------------------------
// 阈值比较
//-----------------------------------------------
always@(posedge iCLK, negedge iRST_N) begin
    if (!iRST_N)
        oDVAL <= 0;
    else begin
        oDVAL <= iDVAL;

        if (iDVAL)
            oDATA <= (Abs_mag > iTHRESHOLD) ? 0 : 1023;
        else
            oDATA <= 0;
    end
end

endmodule
```

IP 设置：

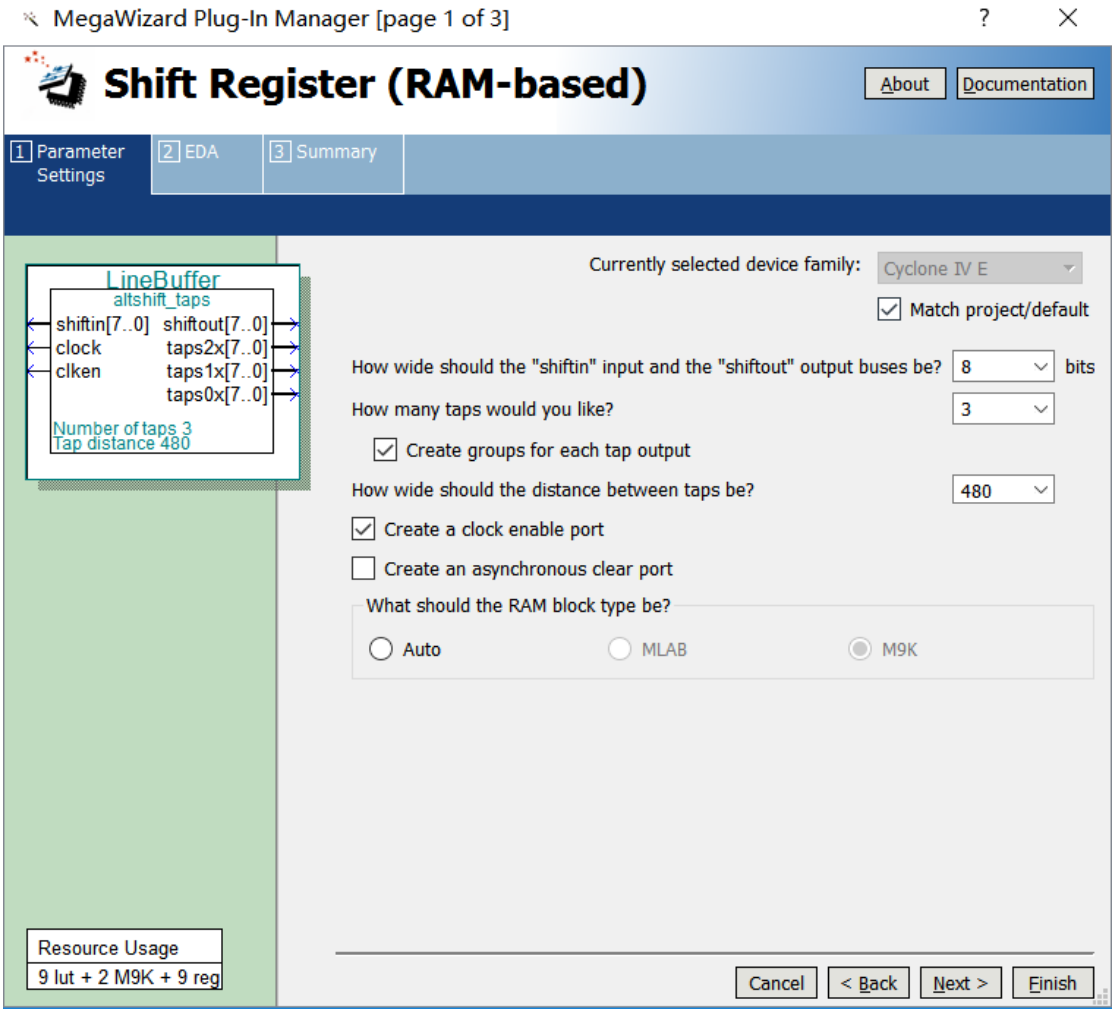


图 3 LineBuffer IP 的设置

因为我们使用的是 3*3 与 3*3 的矩阵相乘，（480*272）每行像素 480 个，形成三行流水，所以设置如图 3。

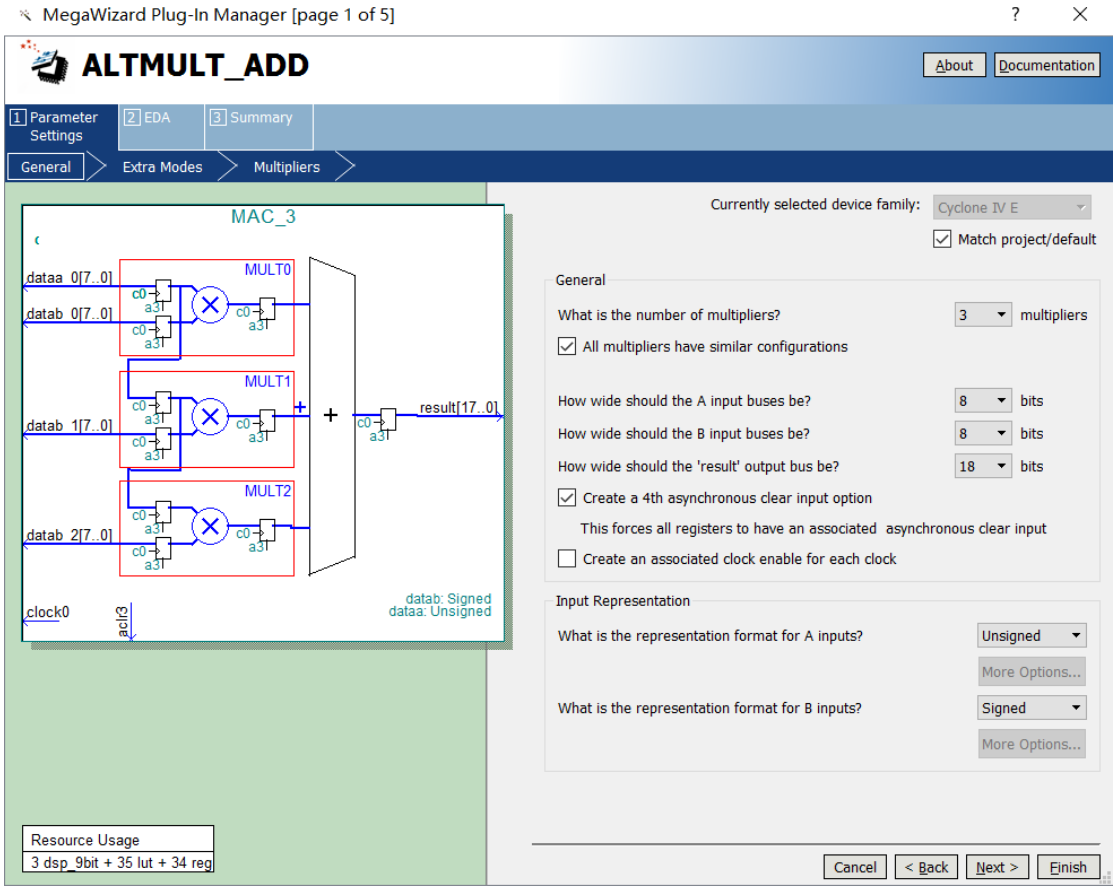


图 4 乘加 IP 设置

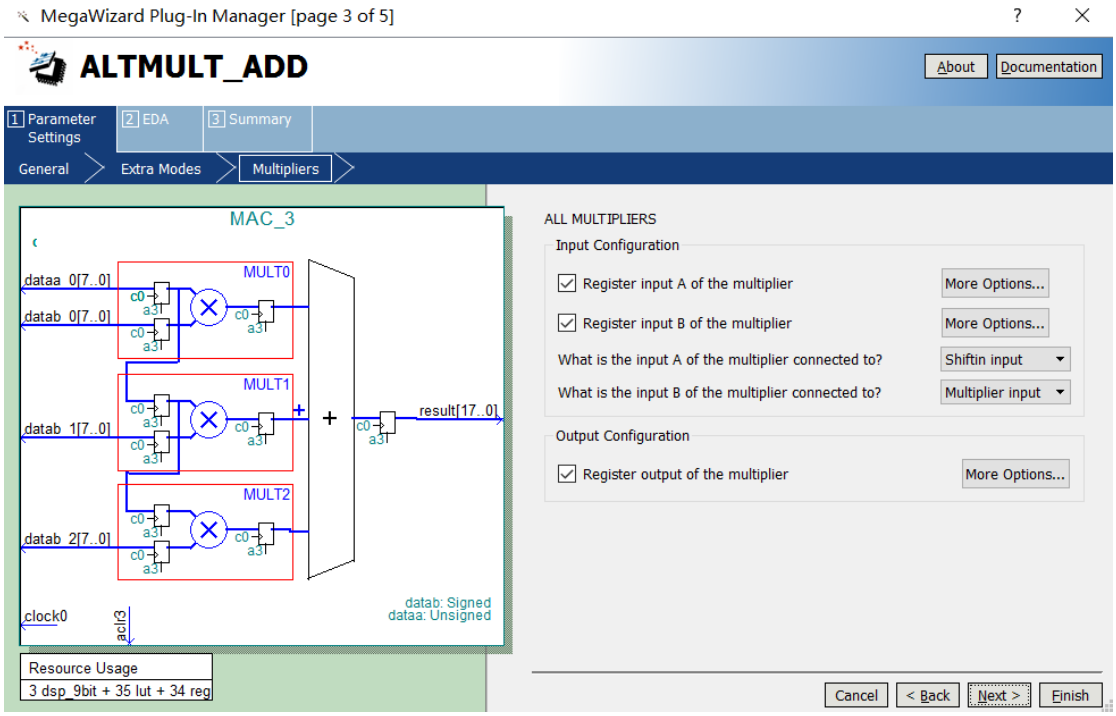


图 5 乘加 IP 设置

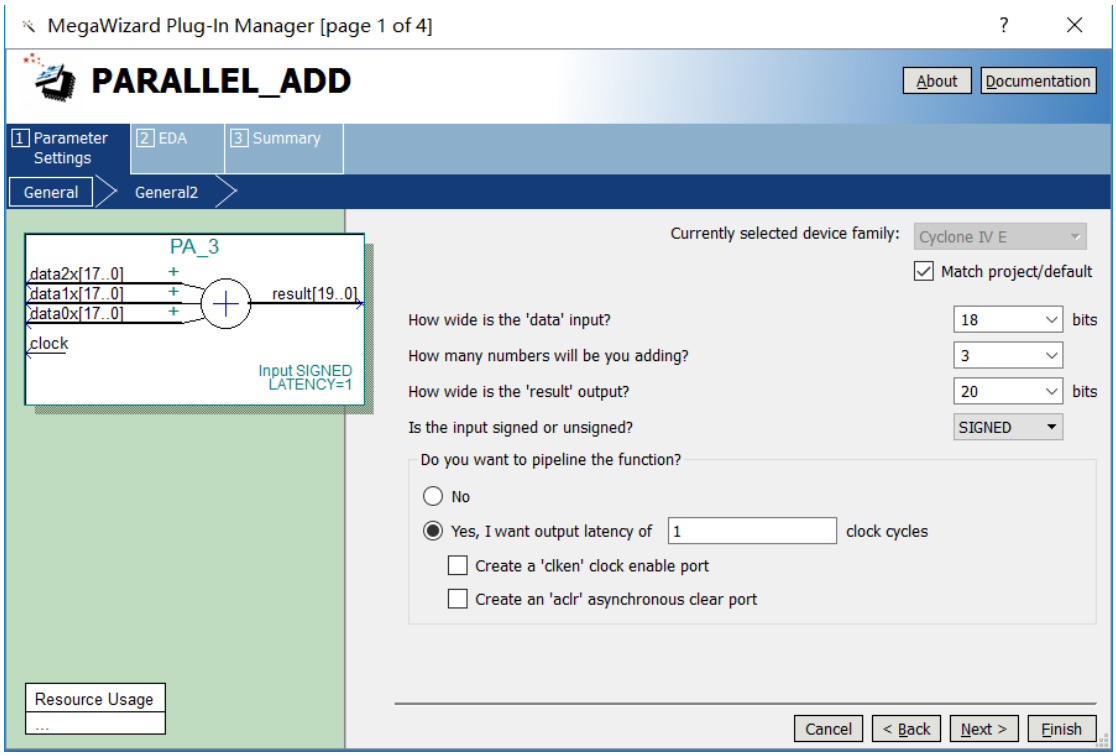


图 6 PA_3 IP 的设置

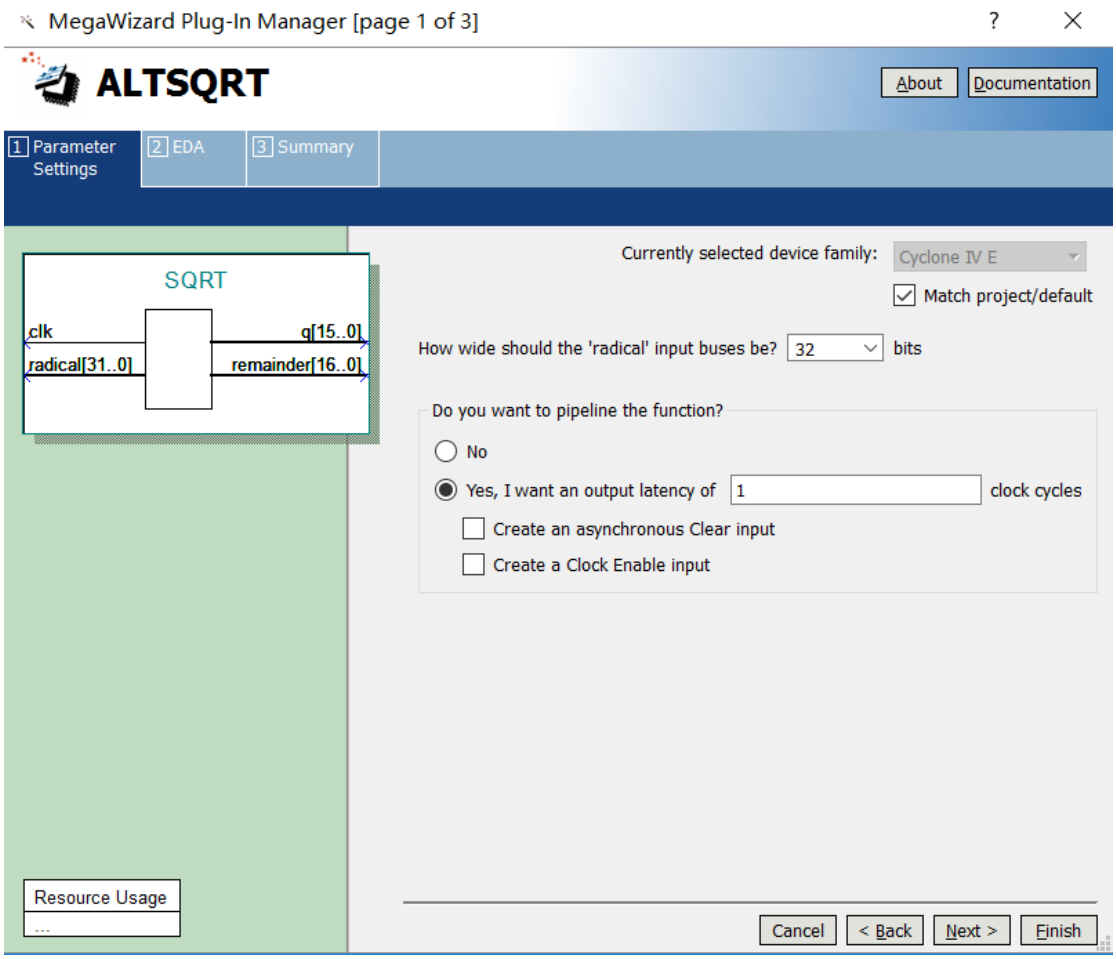


图 7 SQRT IP 的设置

实现结果：

图 8 lena 原图

图 9 阈值 3

图 10 阈值 5

图 11 阈值 7

由图9,10,11可知随着阈值的增大边缘检测的细节在不断减少，大家可以自己改变阈值进行实验多对比。

基于 FPGA 的 5 寸 LCD 显示屏的显示控制

作者：lee 神

图像处理基础知识

数字图像处理是指将图像信号转换成数字信号并利用计算机对其进行处理的过程。图像处理最早出现于 20 世纪 50 年代，当时的电子计算机已经发展到一定水平，人们开始利用计算机来处理图形和图像信息。数字图像处理作为一门学科大约形成于 20 世纪 60 年代初期。早期的图像处理的目的是改善图像的质量，它以人为对象，以改善人的视觉效果为目的。图像处理中，输入的是质量低的图像，输出的是改善质量后的图像，常用的图像处理方法有图像增强、复原、编码、压缩等。

数字图像处理常用方法：

1 ）图像变换：由于图像阵列很大，直接在空间域中进行处理，涉及计算量很大。因此，往往采用各种图像变换的方法，如傅立叶变换、沃尔什变换、离散余弦变换等间接处理技术，将空间域的处理转换为变换域处理，不仅可减少计算量，而且可获得更有效的处理（如傅立叶变换可在频域中进行数字滤波处理）。目前新兴研究的小波变换在时域和频域中都具有良好的局部化特性，它在图像处理中也有着广泛而有效的应用。

2 ）图像编码压缩：图像编码压缩技术可减少描述图像的数据量（即比特数），以便节省图像传输、处理时间和减少所占用的存储器容量。压缩可以在不失真的前提下获得，也可以在允许的失真条件下进行。编码是压缩技术中最重要的方法，它在图像处理技术中是发展最早且比较成熟的技术。

3 ）图像增强和复原：图像增强和复原的目的是为了提高图像的质量，如去除噪声，提高图像的清晰度等。图像增强不考虑图像降质的原因，突出图像中所感兴趣的部分。如强化图像高频分量，可使图像中物体轮廓清晰，细节明显；如强化低频分量可减少图像中噪声影响。图像复原要求对图像降质的原因有一定的了解，一般讲应根据降质过程建立“降质模型”，再采用某种滤波方法，恢复或重建原来的图像。

4 ）图像分割：图像分割是数字图像处理中的关键技术之一。图像分割是将图像中有意义的特征部分提取出来，其有意义的特征有图像中的边缘、区域等，这是进一步进行图像识别、分析和理解的基础。虽然目前已研究出不少边缘提取、区域分割的方法，但还没有一种普遍适用于各种图像的有效方法。因此，对图像分割的研究还在不断深入之中，是目前图像处理中研究的热点之一。

5 ）图像描述：图像描述是图像识别和理解的必要前提。作为最简单的二值图像可采用其几何特性描述物体的特性，一般图像的描述方法采用二维形状描述，它有边界描述和区域描述两类方法。对于特殊的纹理图像可采用二维纹理特征描述。随着图像处理研究的深入发展，已经开始进行三维物体描述的研究，提出了体积描述、表面描述、广义圆柱体描述等方法。

6 ）图像分类（识别）：图像分类（识别）属于模式识别的范畴，其主要内容是图像经过某些预处理（增强、复原、压缩）后，进行图像分割和特征提取，从而进行判决分类。图像分类常采用经典的模式识别方法，有统计模式分类和句法（结构）模式分类，近年来新发展起来的模糊模式识别和人工神经网络模式分类在图像识别中也越来越受到重视。

随着计算机技术的发展，图像处理技术已经深入到我们生活中的方方面面，其中，在娱乐休闲上的应用已经深入人心。图像处理技术在娱乐中的应用主要包括：电影特效制作、电脑电子游戏、数码相机、视频播放、数字电视等。

电影特效制作：自从 20 世纪 60 年代以来，随着电影中逐渐运用了计算机技术，一个全新的电影世界展现在人们面前，这也是一次电影的革命。越来越多的计算机制作的图像被运用到了电影作品的制作中。其视觉效果的魅力有时已经大大超过了电影故事的本身。如今，我们已经很难发现在一部电影中没有任何的计算机数码元素。

电脑电子游戏：电脑电子游戏的画面，是近年来电子游戏发展最快的部分之一。从 1996 年到现在，游戏画面的进步简直可以用突飞猛进来形容，随着图像处理技术的发展，众多在几年前无法想象的画面在今天已经成为了平平常常的东西。

数码相机：所谓数码相机，是一种能够进行拍摄，并通过内部处理把拍摄到的景物转换成以数字格式存放图像的特殊照相机。与普通相机不同，数码相机并不使用胶片，而是使用固定的或者是可拆卸的半导体存储器来保存获取的图像。数码相机可以直接连接到计算机、电视机或者打印机上。在一定条件下，数码相机还可以直接接到移动式电话机或者手持 PC 机上。由于图像是内部处理的，所以使用者可以马上检查图像是否正确，而且可以立刻打印出来或是通过电子邮件传送出去。

视频播放与数字电视：家庭影院中的 VCD ， DVD 播放器和数字电视中，大量使用了视频编码解码等图像处理技术，而视频编码解码等图像处理技术的发展，也推动了视频播放与数字电视象高清晰，高画质发展。

LCD 显示的基本原理

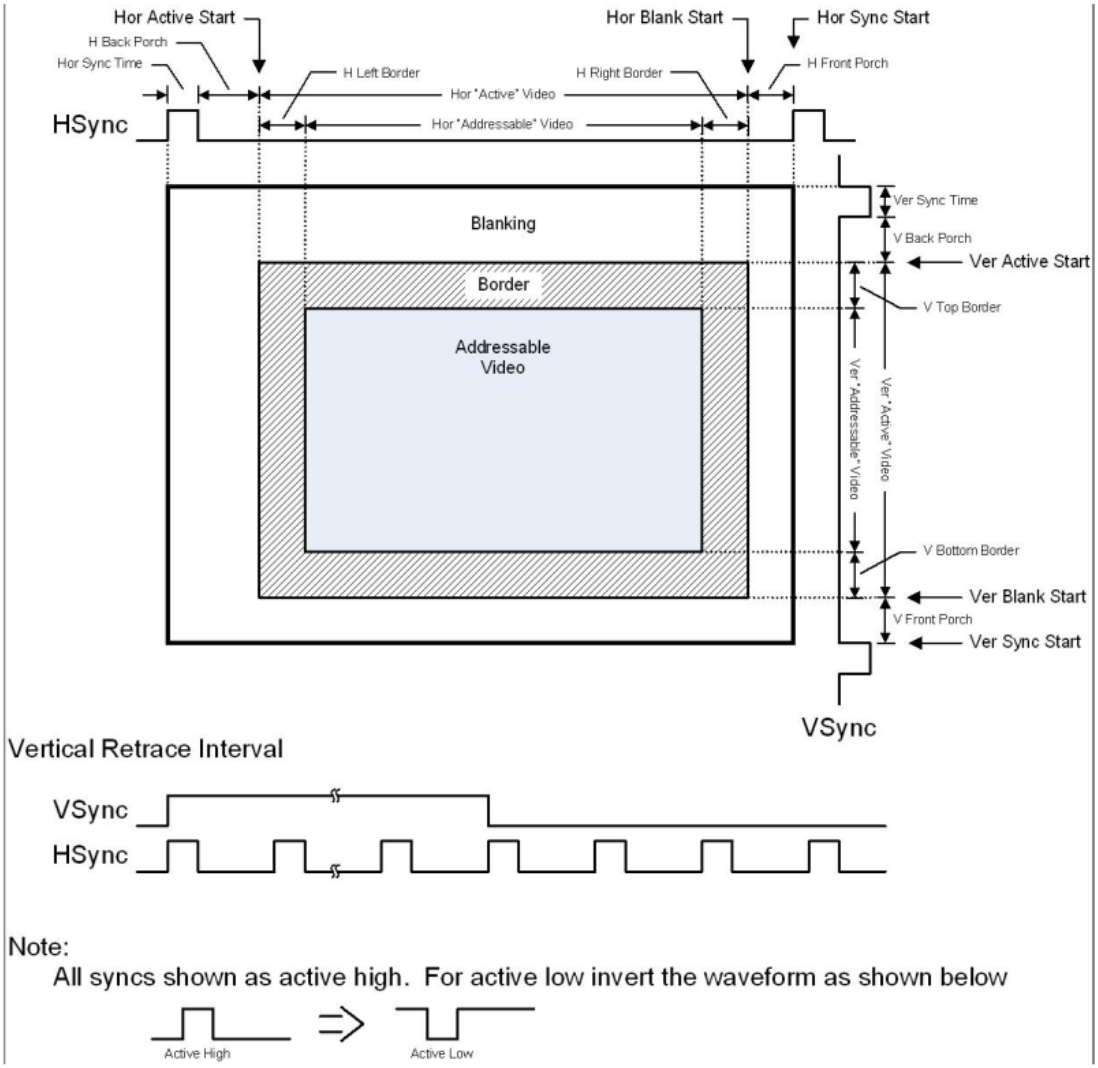


图 1 VGA 的显示时序

如图 1 所示，LCD 的显示和 VGA 的显示时序基本一致，都是从屏幕的左上角开始（从左往右，从上往下）经过 Hor_sync_time 和 H_back_porch 时间，屏幕开始显示，到 H_front_porch 时间后结束一行的显示，然后回到下一行左侧，循环到屏幕的最后一行扫描。在竖直方向上，经过 ver_sync_time 和 v_back_porch 时间竖直方向屏幕开始显示，到 ver_front_porch 竖直方向显示结束。一帧显示完成。

当屏幕的刷新频率快于人眼的视觉感知的频率我们将看不出屏幕的闪烁效果。

FPGA 实现

本实验目的：

本节目的是让大家了解 LCD 屏的显示原理，以及为后期我们的 FPGA 的数字图像处理打下基础。

模块划分：

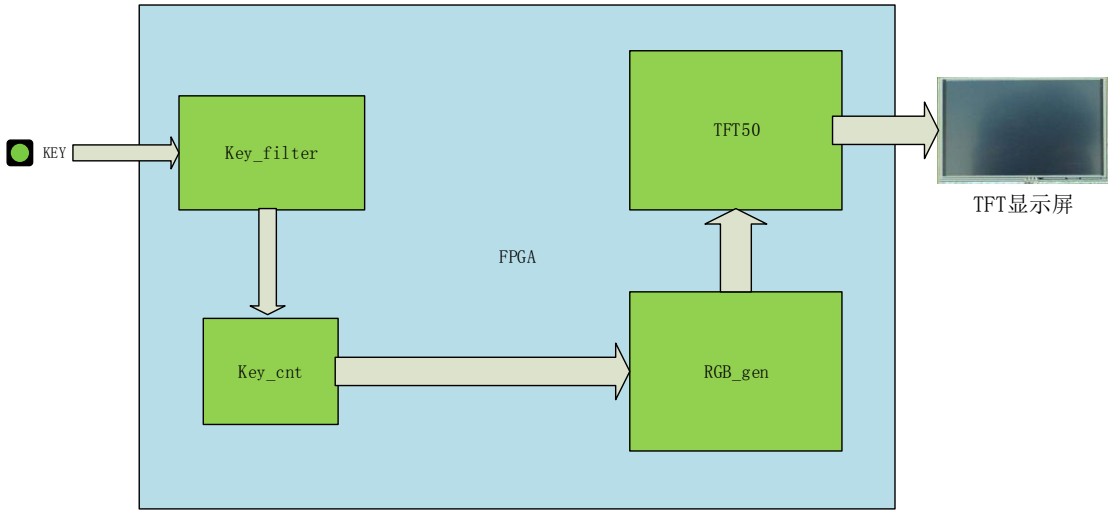


图 2 TFT5 寸显示屏显示 FPGA 模块结构

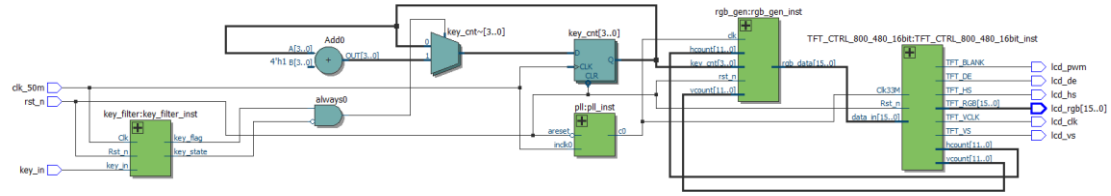


图 3 综合后 FPGA 的内部模块以及接口

从图 2 和图 3 可知，LCD 屏显示控制有 Key_filter、rgb_gen 以及 TFT_CTRL_800_480_16bit 三大模块组成。Key_filter 完成按键的消抖，rgb_gen 完成屏幕显示的控制，TFT_CTRL_800_480_16bit 模块完成 TFT5 寸屏幕的驱动。

本实验通过按键来完成对屏幕颜色输出的控制。

RGB565 可以完成 65536 种颜色的输出。

硬件平台：

TFT5 寸屏幕/或 VGA 显示屏

FPGA 开发板

FPGA 源码：

1）TFT_5 模块代码：

```
/*
Module name:    tft_5.v
Description:    TFT5 screen display control.
```

```
Engineer:          lipu
*/
`timescale 1ns/1ps
module TFT_5(
          input              clk_50m,
            input              rst_n,
            input              key_in,
            output [15:0] lcd_rgb,
            output             lcd_hs,
            output             lcd_vs,
            output             lcd_pwm,
            output             lcd_clk,
            output             lcd_de
          );

wire clk_33m;
wire key_flag;
wire key_state;
wire    [15:0] rgb;
wire    [11:0] hcount;
wire    [11:0] vcount;

reg [3:0] key_cnt;

always @(posedge clk_50m or negedge rst_n) begin
   if(!rst_n)
      key_cnt <= 4'd0;
   else if(key_flag && (!key_state))
      key_cnt <= key_cnt + 4'd1;
   else
      key_cnt <= key_cnt;
end

pll pll_inst(
       .areset(!rst_n),
       .inclk0(clk_50m),
       .c0(clk_33m)
       );

rgb_gen rgb_gen_inst(
            .clk(clk_33m),//33mhz
              .rst_n(rst_n),
              .key_cnt(key_cnt),
              .rgb_data(rgb),
```

```
            .hcount(hcount),
          .vcount(vcount)
            );


key_filter key_filter_inst(
                .Clk(clk_50m),          //50M 时钟输入
                .Rst_n(rst_n),       //模块复位
                .key_in(key_in),     //按键输入
                .key_flag(key_flag), //按键标志信号
                .key_state(key_state) //按键状态信号
          );

TFT_CTRL_800_480_16bit TFT_CTRL_800_480_16bit_inst(
                .Clk33M(clk_33m),     //系统输入时钟 25MHZ
                .Rst_n(rst_n),     //复位输入，低电平复位
                .data_in(rgb),     //待显示数据
                .hcount(hcount),       //TFT 行扫描计数器
                .vcount(vcount),       //TFT 场扫描计数器
                .TFT_RGB(lcd_rgb),     //TFT 数据输出
                .TFT_HS(lcd_hs),       //TFT 行同步信号
                .TFT_VS(lcd_vs),       //TFT 场同步信号
                .TFT_BLANK(lcd_pwm),
                .TFT_VCLK(lcd_clk),
                .TFT_DE(lcd_de)
);


endmodule
```

2）Rgb_gen 模块源码：

```
  /*
Module name:     rgb_gen.v
Description:   rgb 颜色产生
*/
`timescale 1ns/1ps
module rgb_gen(
          input clk,
            input rst_n,
            input [3:0] key_cnt,

            output reg [15:0] rgb_data,
            input    [11:0] hcount,
          input    [11:0] vcount
```

```
        );
    parameter TFT_HS_end=10'd1,
                hdat_begin=10'd46,
                hdat_end=10'd846,
                hpixel_end=12'd1056,
                TFT_VS_end=10'd1,
                vdat_begin=10'd24,
                vdat_end=10'd504,
                vline_end=10'd524;
    parameter h_8 = 10'd100;
    parameter v_8 = 10'd60;
always @(posedge clk or negedge rst_n) begin
  if(!rst_n)
    rgb_data <= 16'h0000;
  else
    case(key_cnt)
      4'd0:rgb_data <= 16'hf800;       //red
        4'd1:rgb_data <= 16'h07e0;    //green
        4'd2:rgb_data <= 16'h001f;    //blue
        4'd3:rgb_data <= 16'hf81f;    //purple
        4'd4:rgb_data <= 16'hffe0;    //yellow
        4'd5:rgb_data <= 16'h07ff;    //cyan
        4'd6:rgb_data <= 16'hfc00;    //orange
        4'd7:rgb_data <= 16'hffff;    //white
        4'd8:rgb_data <= 16'h0000;    //black
        4'd9:begin
          if((hcount[3] == 1'b1) ^ (vcount[3] == 1'b1))
            rgb_data <= 16'hffff;
          else
            rgb_data <= 16'h0000;                    //Lattices image 1
        end
        4'd10:begin
          if((hcount[6] == 1'b1) ^ (vcount[6] == 1'b1))
            rgb_data <= 16'hffff;
          else
            rgb_data <= 16'h0000;                    //Lattices image 2
        end
        4'd11:begin
          if(hcount == hdat_begin)
            rgb_data <= 16'hf800;
          else if(hcount == hdat_begin + h_8)
            rgb_data <= 16'h07e0;
          else if(hcount == hdat_begin + h_8*2)
            rgb_data <= 16'h001f;
```

```
            else if(hcount == hdat_begin + h_8*3)
                rgb_data <= 16'hf81f;
            else if(hcount == hdat_begin + h_8*4)
                rgb_data <= 16'hffe0;
            else if(hcount == hdat_begin + h_8*5)
                rgb_data <= 16'h07ff;
            else if(hcount == hdat_begin + h_8*6)    //color bar h
                rgb_data <= 16'hfc00;
            else if(hcount == hdat_begin + h_8*7)
                rgb_data <= 16'hffff;
            else
                rgb_data <= rgb_data;
        end
        4'd12:begin
            if(vcount == vdat_begin)
                rgb_data <= 16'hf800;
            else if(vcount == vdat_begin + v_8)
                rgb_data <= 16'h07e0;
            else if(vcount == vdat_begin + v_8*2)
                rgb_data <= 16'h001f;
            else if(vcount == vdat_begin + v_8*3)
                rgb_data <= 16'hf81f;
            else if(vcount == vdat_begin + v_8*4) //color bar v
                rgb_data <= 16'hffe0;
            else if(vcount == vdat_begin + v_8*5)
                rgb_data <= 16'h07ff;
            else if(vcount == vdat_begin + v_8*6)
                rgb_data <= 16'hfc00;
            else if(vcount == vdat_begin + v_8*7)
                rgb_data <= 16'hffff;
            else
                rgb_data <= rgb_data;
        end
        4'd13:begin
            rgb_data <= {5'b0,vcount[8:3],5'b0};    //Horizontal green gradual change
        end
        4'd14:begin
            rgb_data <= {vcount[8:4],vcount[8:3],vcount[8:4]}; //vertical gray Gradient
        end
        4'd15:begin
            rgb_data <= {hcount[8:4],11'b0};    //Horizontal red gradual change
        end
    endcase
end
```

```
endmodule
```

3）（小梅哥源码）TFT_CTRL_800_480_16bit 模块源码：

```
module TFT_CTRL_800_480_16bit(
    Clk33M,  //系统输入时钟 33MHZ
    Rst_n,   //复位输入，低电平复位
    data_in,  //待显示数据
    hcount,      //TFT 行扫描计数器
    vcount,      //TFT 场扫描计数器
    TFT_RGB,//TFT 数据输出
    TFT_HS,      //TFT 行同步信号
    TFT_VS,      //TFT 场同步信号
    TFT_BLANK,
    TFT_VCLK,
    TFT_DE
);

    //----------------模块输入端口----------------
    input   Clk33M;             //系统输入时钟 33MHZ
    input   Rst_n;
    input   [15:0]data_in;      //待显示数据

    //----------------模块输出端口----------------
    output [11:0]hcount;
    output [11:0]vcount;
    output [15:0]TFT_RGB;   //TFT 数据输出
    output TFT_HS;              //TFT 行同步信号
    output TFT_VS;              //TFT 场同步信号
    output TFT_BLANK;
    output TFT_DE;
    output TFT_VCLK;

    //----------------内部寄存器定义----------------
    reg [11:0] hcount_r;        //TFT 行扫描计数器
    reg [11:0] vcount_r;        //TFT 场扫描计数器
    //----------------内部连线定义----------------
    wire hcount_ov;
    wire vcount_ov;
    wire TFT_DE;//有效显示区标定

    //TFT 行、场扫描时序参数表
    parameter TFT_HS_end=10'd1,
                  hdat_begin=10'd46,
```

```
                hdat_end=10'd846,
                hpixel_end=12'd1056,
                TFT_VS_end=10'd1,
                vdat_begin=10'd24,
                vdat_end=10'd504,
                vline_end=10'd524;
    assign hcount=hcount_r;
    assign vcount=vcount_r;

    assign TFT_BLANK = Rst_n;
    assign TFT_VCLK = Clk33M;

    //**********************TFT 驱动部分**********************
    //行扫描
    always@(posedge Clk33M or negedge Rst_n)
    if(!Rst_n)
        hcount_r<=12'd0;
    else if(hcount_ov)
        hcount_r<=12'd0;
    else
        hcount_r<=hcount_r+12'd1;

    assign hcount_ov=(hcount_r==hpixel_end);

    //场扫描
    always@(posedge Clk33M or negedge Rst_n)
    if(!Rst_n)
        vcount_r<=12'd0;
    else if(hcount_ov) begin
        if(vcount_ov)
            vcount_r<=12'd0;
        else
            vcount_r<=vcount_r+12'd1;
    end
    else
        vcount_r<=vcount_r;

    assign    vcount_ov=(vcount_r==vline_end);

    //数据、同步信号输出
    assign TFT_DE=((hcount_r>=hdat_begin)&&(hcount_r<hdat_end))
                        &&((vcount_r>=vdat_begin)&&(vcount_r<vdat_end));

    assign TFT_HS=(hcount_r>TFT_HS_end);
```

```
    assign TFT_VS=(vcount_r>TFT_VS_end);
    assign TFT_RGB=(TFT_DE)?data_in:16'h000000;

endmodule
```

实验结果：

图 3 Red

图 4 Green

图 5 Blue

图 6 Color_bar_h

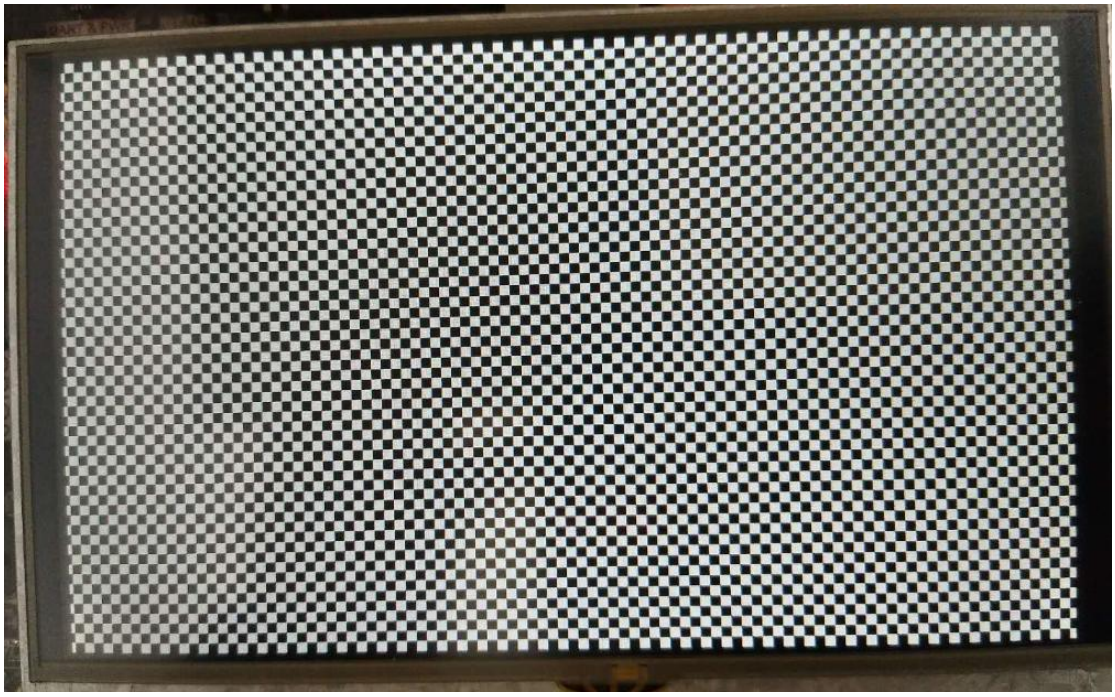


图 6 Lattices image 1

通过按键进行不同的显示切换。

本节旨在让大家了解 TFT 显示屏的显示原理以及控制原理，为后边的图像处理打下坚实的基础。