

41 灰度图像中值滤波设计实现(HDMI 和 TFT 显示)

工程源码	---02_设计实例 ---ch41_acz7015_uart_ddr3_tft_hdmi_median_filter
相关视频课程	
	如果您手头的硬件不支持本实验，您可以学习本实验的理论内容，也可以跳过本节内容，继续后续内容的学习。

章节导读

本章节将基于上一章节的内容，讲述灰度图像的中值滤波算法实现。将灰度图像进行中值滤波处理的功能是实现图像的增强和复原。中值滤波可以有效将图像中孤立噪声点、椒盐噪声点杂色滤除，使图像的内容得到修复和完善。中值滤波算法的核心在于实现像素矩阵数据的大小排序。如何快速而准确地实现像素数值大小排序是评判该中值滤波算法模块优劣的重要标准。

本章节主要是在上一节的基础上，加入中值滤波图像处理模块，实现在 PC 端通过上位机下发尺寸为 400*480 大小的彩色图像数据到 FPGA 的串口，FPGA 通过串口接收的彩色图像数据并进行实时彩色图像灰度化处理，同时进行中值滤波的处理，然后将中值滤波处理前和处理后的图像拼接在一起并缓存在 DDR3 中，最终在 TFT 屏和 HDMI 显示器上同时显示中值滤波处理前的灰度图像和处理后的灰度图像。

41.1 系统整体设计

本节内容为基于灰度图像的中值滤波，我们希望得到如下实验效果。最终在 TFT 屏和 HDMI 显示器显示的要求如下。

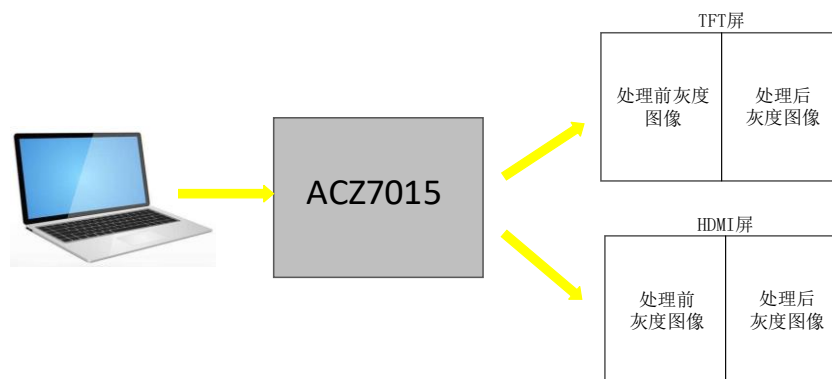


图 41-1 灰度图像中值滤波实验目标

系统整体设计框图如下。大体结构与“彩色图像灰度化的设计实现”基本一致，增加了中值滤波处理模块。

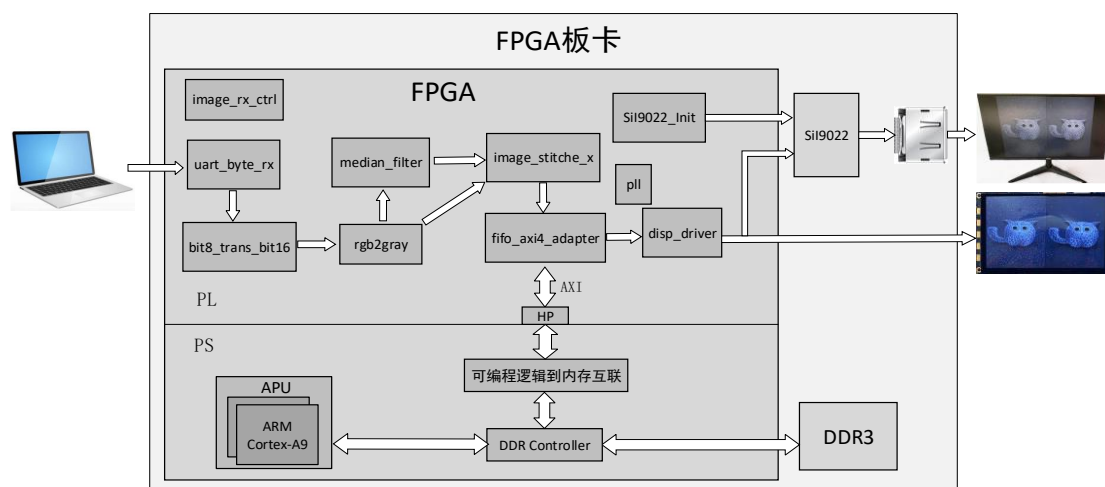


图 41-2 系统设计框图

其中，

- (1) **uart_byte_rx** 模块：负责串口图像数据的接收，该模块的设计细节参考前面对应章节。
- (2) **bit8_trans_bit16** 模块：将串口接收的每两个 8bit 数据转换成一个 16bit 数据（图像数据是 16bit 的 RGB565 的数据，电脑是通过串口将一个像素点数据分两次发送到 FPGA，FPGA 需将串口接收数据重组成 16bit 的图像数据），实现过程相对比较简单。
- (3) **rgb2gray** 模块：彩色图像灰度化处理，对串口接收的彩色图像数据实时进行灰度化处理。
- (4) **median_filter** 模块：对彩色图像进行灰度化处理后的数据进行中值滤波处理。
- (5) **image_stitch_x** 模块：将串口接收的尺寸为 400*480 大小的彩色图像与灰度化处理后的 400*480 大小的图像数据以左右形式合并成一张 800*480 的图像。
- (6) **fifo_axi4_adapter** 模块组：包含 wr_ddr3_fifo、rd_ddr3_fifo、fifo2axi4 以模块，完成采集的图像数据缓存。
- (7) **pll** 模块：上述各个模块所需时钟的产生，使用 PLL IP。
- (8) **image_rx_ctrl** 模块。用于显示接收进度，给出接收完成标志。
- (9) **dvi_encoder** 模块：用于将生成信号转换成 HDMI 输出信号作 HDMI 显

示。

本系统就是在上一系统基础上添加图像处理模块搭建系统。除去使用 IP 和前面章节讲过的模块外，只需要设计中值滤波模块。

41.2 灰度图像中值滤波处理模块的设计

41.2.1 基本原理

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）的二维数据序列。二维中值滤波输出为 $g(x,y) = \text{mid}\{f(x-k,y-l), (k,l \in W)\}$ ，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，在光学测量条纹图像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。

41.2.2 FPGA 中值滤波实现算法简介

下面介绍的即是通过硬件平台实现 FPGA 中值滤波排序算法：

为便于说明实现步骤，滤波的像素矩阵模板取 3×3 大小的九宫格，矩阵第二行第二列的值 $L(2,2)$ 即为目标处理像素点，其邻域的 8 个点，即为其周围的像素值，用以辅助 $L(2,2)$ 实现中值滤波运算。

表 41-1 FPGA 中值滤波算法实现

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;
- 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;
- 由 L1mid, L2mid, L3mid 得到 Lmid_max, Lmid_mid, Lmid_min;
- 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 median）。

FPGA 的算法实现步骤基本如此。

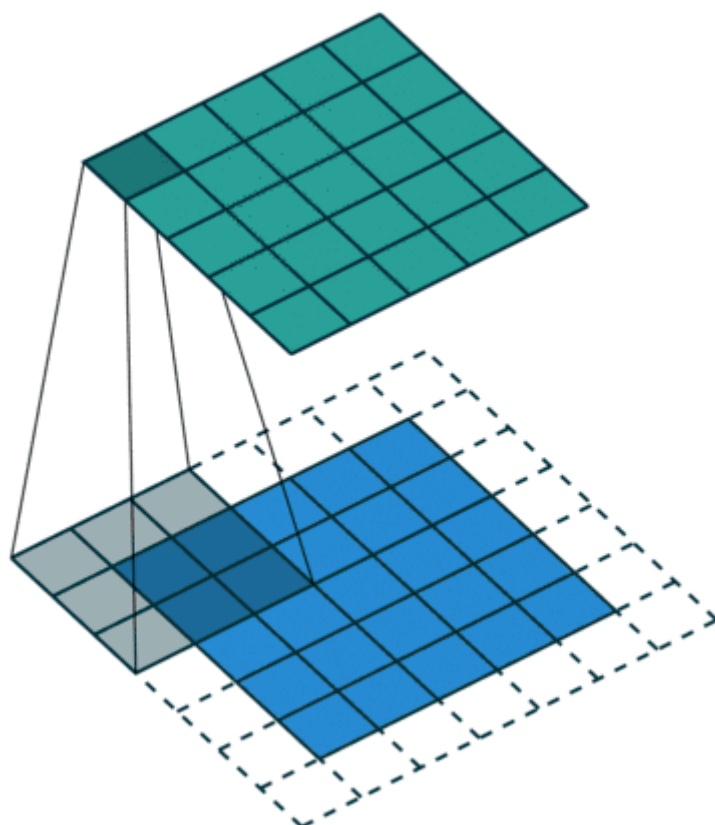


图 41-3 需计算的像素点及提取的滤波像素矩阵

为进一步了解中值滤波算法的实现步骤，我们以数值进行举例演示。假设数值的大小即为像素点亮度大小。

41.2.3FPGA 中值滤波实现算法演示

举例来说，某图像数据的局部，存在如下 3x3 的亮度数据矩阵如下：

表 41-2 3x3 图像数据矩阵举例

2	4	80
3	5	4
6	3	2

从上面 3x3 的数据矩阵中，不难发现，以上像素矩阵以九宫格中心数字 5 为需要处理的像素点。选取的像素矩阵中有数据 80 和周边的点有明显的颜色显示数值差异。反映到图像中，应该是一个比周边值更亮的噪声孤点。

如果采用中值滤波算法，对以上原始数据矩阵进行处理：

第一步：首先将 9 个图像数据按行分成 3 组， 分别对三组数据进行排序。

(1) 对 L11, L12, L13 进行排序得到：

L1max (即 80), L1mid (即 4), L1min (即 2);

(2) 对 L21, L22, L23 进行排序得到：

L2max (即 5), L2mid (即 4), L2min (即 3);

(3) 对 L31, L32, L33 进行排序得到：

L3max (即 6), L3mid (即 3), L3min (即 2)。

处理后，得到一个临时过渡的矩阵，内容为：

表 41-3 图像矩阵举例第一步运算结果

80	4	2
5	4	3
6	3	2

第二步：分别对三组像素数据中的 3 个最大， 3 个中间和 3 个最小分别进行排序。

(1) 对 L1max, L2max, L3max 进行排序得到：

Lmaxmax (即 80), Lmaxmid (即 6), Lmaxmin (即 5);

(2) 对 L1mid, L2mid, L3mid 进行排序得到：

Lmidmax (即 4), Lmidmid (即 4), Lmidmin (即 3);

(3) 对 L1min, L2min, L3min 进行排序得到：

Lminmax (即 3), Lminmid (即 2), Lminmin (即 2);

处理后，得到这样第二个临时的过渡矩阵，内容为：

表 41-4 图像矩阵举例第二步运算结果

80	6	5
4	4	3
3	2	2

第三步：对最大的最小(Lmaxmin)，中间的中间(Lmidmid)以及最小的最大(Lminmax)进行排序，排序得到的中间数据为最终的中值。

表 41-5 最终中值筛选

5（第一行末，最大的最小）	4（第二行中，中间的中间）	3（第三行首，最小的最大）
---------------	---------------	---------------

最终，得到该图像矩阵中值为 4。

由于 FPGA 的图像显示数据采用的是逐个像素点的行场扫描输出方案，因此，当有噪声信号点（如图中数据 80）参与中值滤波点阵处理后，该点是不会对像素矩阵最终输出值产生影响的，即实现了图片噪声信号滤波。

这个中值，最终作为我们滤波像素矩阵计算像素输出点安置在新的像素矩阵中，该值就是原像素矩阵中待处理的中心点数字 5 的对应行场位置输出结果。如果对整幅图像都作同样的处理，则将会得到一幅新的图像。这幅新的图像每个像素，都和原图像像素形成中值滤波运算后一一对应的关系。从存储空间的角度分析，完成中值滤波，必须开辟至少两幅图像的存储空间，第一块存储空间作为原图像的存储空间，第二块存储空间作为图像处理后缓存数据的存储空间。否则，如果将处理后的数据安置回原始的存储空间，则下一个运算点进行中值滤波计算时，取用的像素矩阵将不是原始图像的像素矩阵数值，而是带有一个已经滤波完成的像素点参与运算的像素矩阵，这样将导致所有的运算，都受到其他已运算点位的影响，从而使最终输出结果不符合设计预期。

41.2.4模块接口设计

经过分析，中值滤波算法只需要将待处理的数据输入中值滤波模块，即可得到滤波后对应的输出像素值。

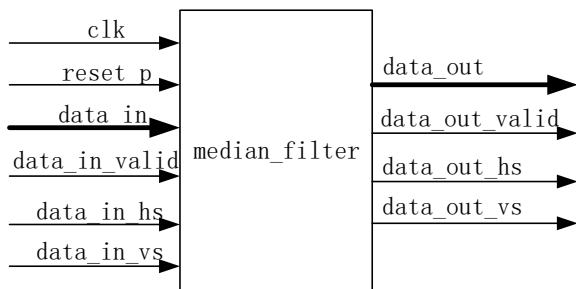


图 41-4 中值处理模块输入输出接口

表 41-6 模块端口描述如下表

端口名称	I/O	端口说明
clk	I	图像像素时钟
reset_p	I	模块复位信号，高电平有效
data_in	I	图像像素数据，数据位宽可自定义，对于灰度图像，位宽为 8
data_in_valid	I	图像像素数据有效标识，为 1 表示当前 data_in 有效

data_in_hs	I	图像行信号
data_in_vs	I	图像场信号
data_out	O	图像处理数据输出，数据位宽与 data_in 相同
data_out_valid	O	图像处理像素数据有效标识，为 1 表示当前 data_out 有效
data_out_hs	O	图像处理行信号
data_out_vs	O	图像处理场信号

41.2.5 实现过程

通过上面的学习，我们知道图像数据是按照时钟节拍一个像素一个像素传入到 FPGA，要实现中值滤波，首先是要产生上述的 3x3 的模板，然后再按照算法步骤进行排序计算。

模块设计之初，必须对原始图像每一个像素点数据的读取和存储进行分析。要想稳定地实现滤波模板的滑动，必须保证滤波模板的输入和输出数据平衡，即：实现模块的端口为 1 个像素点的滤波前灰度值输入，1 个像素点的滤波后灰度值输出。而图像的显示扫描过程是按完整的逐行扫描行来执行，这样就导致必须至少缓存有完整的两行数据，另外加上第三行的至少 3 个原始像素值，才能运算获得第一个像素点的值输出。

换句话说：由于图像数据是一个一个输入进来的，貌似要实现 3x3 的模板，就首先必须要保证同时能对 3 行图像数据进行获取，这样就必须要对图像数据进行行缓存。乍一看，3x3 模板需要缓存 3 行，其实不然，缓存 2 行后，接下来输入进来的数据就是第 3 行的数据了，这样就实现了 3 行数据同时存在的情况了，对行缓存区的要求是左端进入一个数据，右端出来一个数据，这个要求与移位寄存器有些类似。

另外，对于数据缓存，还有这样一个要求：除了这些计算第一个像素点的 9 个数值必须保留并参与运算以外，第一第二行的其他像素点的数值也不能抛弃。这是因为，在像素矩阵滑动取值时，每个点只有一次被扫描读取的机会，如果需要利用第一行后续点位像素值计算第二行相同位置的像素值时，那么这些值就再也读不回来了。

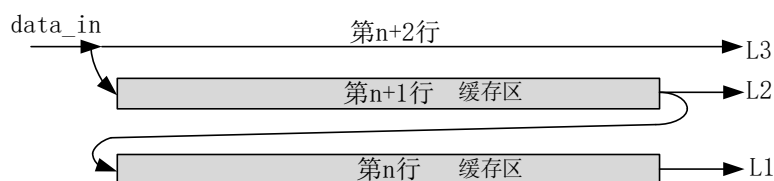


图 41-5 逐行中值滤波矩阵处理

接下来，看看如何生成移位寄存器 IP。

在 Xilinx 里有个 IP 正好可以满足我们的需求，IP 名叫 RAM-base Shift Register，可在 Vivado IPCatalog 进行搜索。IP 设置界面如下。这里是对灰度图像数据进行处理，数据位宽设置为 8。由于待处理的图像数据尺寸为 400*480，所以深度设置为 400。

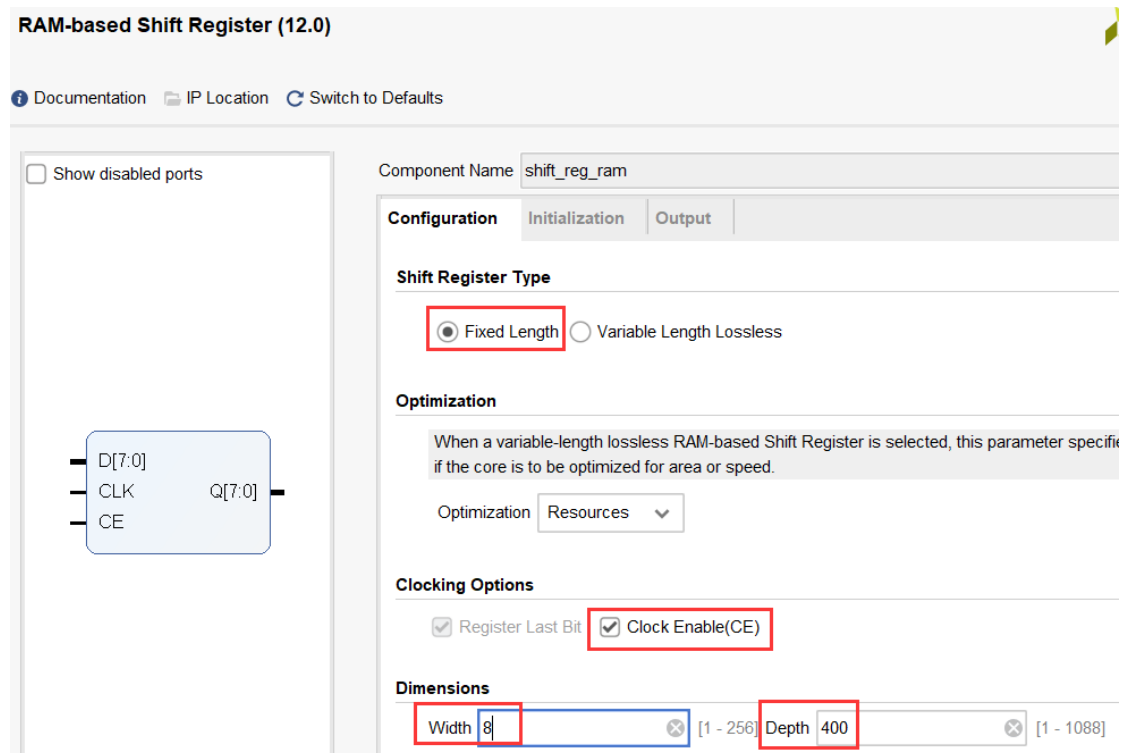


图 41-6 中值滤波处理 IP

该 IP 的基本原理如下图所示。D[7:0]为输入数据，Q[7:0]输出数据，在时钟使能 CE 为高的情况下，输入端 D，在每个时钟周期输入一个数据，输出端 Q 输出一个数据，缓存在 RAM 内的数据向右移动一个位置。

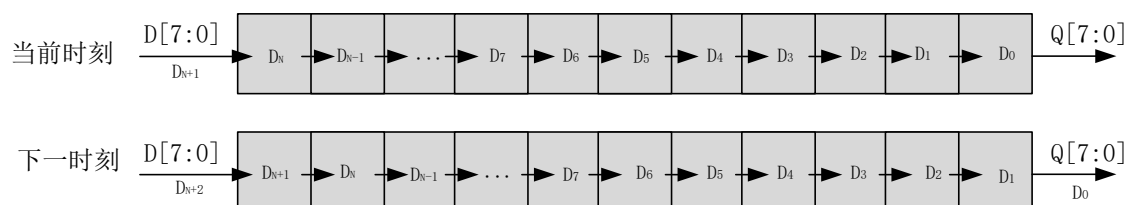


图 41-7 图像数据处理流程

要实现 2 行数据的缓存，只需要例化两个移位寄存器 IP，然后将其中一个 IP 的输出接到另一个 IP 的输入。具体代码如下

```
module shift_register_2taps
```

```

#(
  parameter DATA_WIDTH = 8
)
(
  input          clk,
  input [DATA_WIDTH-1:0] shiftin,
  input          shiftin_valid,

  output [DATA_WIDTH-1:0] shiftout,
  output [DATA_WIDTH-1:0] taps1x,
  output [DATA_WIDTH-1:0] taps0x
);

assign shiftout = taps0x;

shift_reg_ram shift_reg_ram_inst1 (
  .D (shiftin          ),// input wire [7 : 0] D
  .CLK (clk            ),// input wire CLK
  .CE (shiftin_valid   ),// input wire CE
  .Q  (taps1x          ) // output wire [7 : 0] Q
);

shift_reg_ram shift_reg_ram_inst2 (
  .D (taps1x           ),// input wire [7 : 0] D
  .CLK (clk            ),// input wire CLK
  .CE (shiftin_valid   ),// input wire CE
  .Q  (taps0x          ) // output wire [7 : 0] Q
);

endmodule

```

代码中 taps0x 对应下图中 L1 输出，taps1x 对应下图中 L2 输出，shiftin 对应图中 L3 输出。

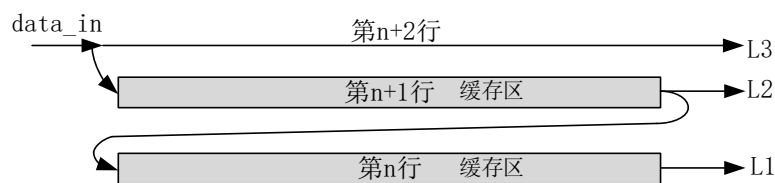


图 41-8

编写模块仿真测试代码如下。仿真中产生了 3 组相同的数据，每组数据从 1 开始，每个周期数据加 1。

```

`timescale 1ns/1ns
`define CLK_PERIOD 40

```

```
module shift_register_2taps_tb();
    reg        clk;
    reg [7:0] shiftin;
    reg        shiftin_valid;

    wire [7:0] shiftout;
    wire [7:0] taps1x;
    wire [7:0] taps0x;

    initial clk = 1'b1;
    always #(`CLK_PERIOD/2) clk = ~clk;

    initial begin
        shiftin_valid = 1'b0;
        shiftin = 'd0;
        #201;
        repeat(3) begin
            shiftin = 'd0;
            repeat(400) begin
                shiftin_valid = 1'b1;
                shiftin = shiftin + 1'b1;
                #(`CLK_PERIOD);
            end
        end
        #200;
        $stop;
    end

    shift_register_2taps
    shift_register_2taps_inst
    (
        .clk          (clk          ),
        .shiftin       (shiftin      ),
        .shiftin_valid (shiftin_valid),

        .shiftout      (shiftout     ),
        .taps1x        (taps1x       ),
        .taps0x        (taps0x       )
    );
endmodule
```

仿真波形如下。

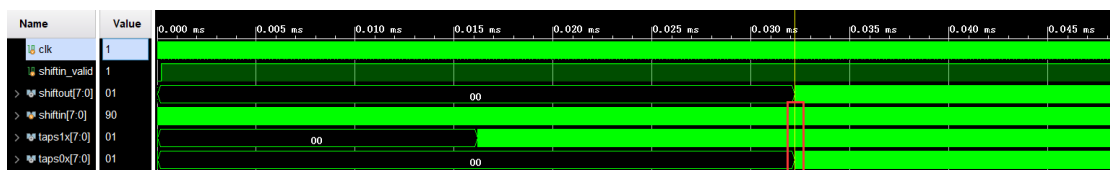


图 41-9 移位寄存模块仿真波形

对上图中红色圈注的地方进行波形放大查看如下图。

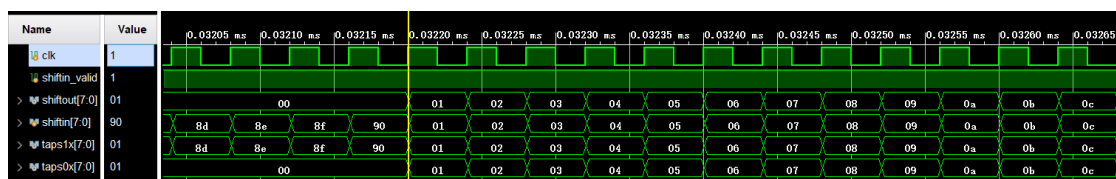


图 41-10 移位寄存模块放大信息图

从仿真波形可以看出从这个时候开始，taps0x、taps1x 和 shiftin 输出数据都是从 1 开始递增，并且数据一样，这与预期一致，因为仿真输入的是 3 行完全相同的数据，该模块实现的功能是缓存 2 行数据，这样在开始输入第 3 行数据的时候，就可以在输出同时出现 3 行相同的数据了。

上面仅仅实现了 3 行数据的同时存在，即 3*1 的模板，要实现 3*3 的模板，还需要进一步处理。输入的数据是一个接着一个的，只需要 3 组采用寄存器将数据存储就可以实现 3 列数据同时出现，如下图所示是 3*3 模板数据与寄存器之间对应关系。

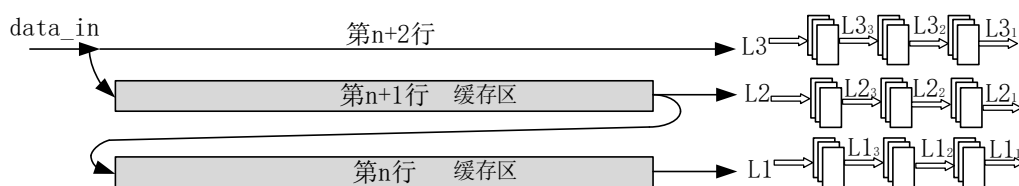


图 41-11 3*3 模板数据与寄存器之间对应关系

具体实现代码如下。

第 1 部分：定义滤波像素矩阵模板的数据缓存空间

代码中 row0_col0、row0_col1、row0_col2、row1_col0、row1_col1、row1_col2、row2_col0、row2_col1、row2_col2 分别对应 3*3 模板中的 9 个数据缓存空间。

```
//-----
// matrix 3x3 data
// row0_col0  row0_col1  row0_col2
// row1_col0  row1_col1  row1_col2
// row2_col0  row2_col1  row2_col2
//-----
always @(posedge clk or posedge reset_p) begin
    if(reset_p) begin
        row0_col0 <= 'd0;
        row0_col1 <= 'd0;
```

```
row0_col2 <= 'd0;

row1_col0 <= 'd0;
row1_col1 <= 'd0;
row1_col2 <= 'd0;

row2_col0 <= 'd0;
row2_col1 <= 'd0;
row2_col2 <= 'd0;
end
else if(data_in_hs && data_in_vs)
  if(data_in_valid) begin
    row0_col2 <= line0_data;
    row0_col1 <= row0_col2;
    row0_col0 <= row0_col1;

    row1_col2 <= line1_data;
    row1_col1 <= row1_col2;
    row1_col0 <= row1_col1;

    row2_col2 <= line2_data;
    row2_col1 <= row2_col2;
    row2_col0 <= row2_col1;
  end
  else begin
    row0_col2 <= row0_col2;
    row0_col1 <= row0_col1;
    row0_col0 <= row0_col0;

    row1_col2 <= row1_col2;
    row1_col1 <= row1_col1;
    row1_col0 <= row1_col0;

    row2_col2 <= row2_col2;
    row2_col1 <= row2_col1;
    row2_col0 <= row2_col0;
  end
  else begin
    row0_col0 <= 'd0;
    row0_col1 <= 'd0;
    row0_col2 <= 'd0;

    row1_col0 <= 'd0;
    row1_col1 <= 'd0;
    row1_col2 <= 'd0;

    row2_col0 <= 'd0;
```

```
row2_col1 <= 'd0;  
row2_col2 <= 'd0;  
end  
end
```

第 2 部分：实现 3 个数据的排序处理。

3*3 模板中的 9 个数据准备好之后，就是按照中值滤波算法的处理步骤进行，从中值滤波的步骤中可以总结出一个通用的模块，就是对 3 个数据的排序处理。这个处理可以单独作为一个子模块来实现。实现方法比较简单，就是 3 个数据两两进行比较即可，具体实现代码如下。代码中数据位宽采用参数指定，便于移植。

```
module sort  
#(  
  parameter DATA_WIDTH = 8  
)  
(  
  input          clk,          //pixel clk  
  input          reset_p,  
  input          data_in_valid,  
  input [DATA_WIDTH-1:0] data0_in,  
  input [DATA_WIDTH-1:0] data1_in,  
  input [DATA_WIDTH-1:0] data2_in,  
  
  output reg [DATA_WIDTH-1:0] data_max_out,  
  output reg [DATA_WIDTH-1:0] data_mid_out,  
  output reg [DATA_WIDTH-1:0] data_min_out,  
  output reg          data_out_valid  
);  
  
always @(posedge clk or posedge reset_p) begin  
  if(reset_p) begin  
    data_max_out <= 'd0;  
    data_mid_out <= 'd0;  
    data_min_out <= 'd0;  
  end  
  else if(data_in_valid) begin  
    if((data0_in >= data1_in) && (data0_in >= data2_in)) begin  
      data_max_out <= data0_in;  
  
      if(data1_in >= data2_in) begin  
        data_mid_out <= data1_in;  
        data_min_out <= data2_in;  
      end  
    else begin  
      data_mid_out <= data2_in;
```

```
        data_min_out <= data1_in;
    end
end
else if((data1_in > data0_in) && (data1_in >= data2_in)) begin
    data_max_out <= data1_in;

    if(data0_in >= data2_in) begin
        data_mid_out <= data0_in;
        data_min_out <= data2_in;
    end
    else begin
        data_mid_out <= data2_in;
        data_min_out <= data0_in;
    end
end
else if((data2_in > data0_in) && (data2_in > data1_in)) begin
    data_max_out <= data2_in;

    if(data0_in >= data1_in) begin
        data_mid_out <= data0_in;
        data_min_out <= data1_in;
    end
    else begin
        data_mid_out <= data1_in;
        data_min_out <= data0_in;
    end
end
end
end

always @(posedge clk or posedge reset_p)
    if(reset_p)
        data_out_valid <= 1'b0;
    else if(data_in_valid)
        data_out_valid <= 1'b1;
    else
        data_out_valid <= 1'b0;

endmodule
```

第 3 部分：3x3 像素矩阵排序的实现

排序模块就不再单独进行仿真，后面直接放在中值滤波模块里一起进行仿真。目前 3*3 模板数据已经有了，3 个数据的排序也有了，整个中值滤波的实现就比较简单了，按照算法的 3 个步骤进行实现就可以比较轻松的完成，具体代码如下，实现过程中注意 data_in_valid 与 data_in 的对应关系

```
always @(posedge clk)
```

```
begin
    data_in_valid_dly1 <= data_in_valid;
    data_in_valid_dly2 <= data_in_valid_dly1;
    data_in_valid_dly3 <= data_in_valid_dly2;

    data_in_hs_dly1    <= data_in_hs;
    data_in_hs_dly2    <= data_in_hs_dly1;
    data_in_hs_dly3    <= data_in_hs_dly2;

    data_in_vs_dly1    <= data_in_vs;
    data_in_vs_dly2    <= data_in_vs_dly1;
    data_in_vs_dly3    <= data_in_vs_dly2;
end

//-----
// line0 of (max mid min)
//-----
sort
#(
    .DATA_WIDTH (DATA_WIDTH)
)sort_line0
(
    .clk          (clk          ),    //pixel clk
    .reset_p      (reset_p      ),
    .data_in_valid (data_in_valid_dly1 ),
    .data0_in      (row0_col0    ),
    .data1_in      (row0_col1    ),
    .data2_in      (row0_col2    ),

    .data_max_out  (line0_max    ),
    .data_mid_out  (line0_mid    ),
    .data_min_out  (line0_min    ),
    .data_out_valid(             )
);

//-----
// line1 of (max mid min)
//-----
sort
#(
    .DATA_WIDTH (DATA_WIDTH)
)sort_line1
(
    .clk          (clk          ),    //pixel clk
    .reset_p      (reset_p      ),
    .data_in_valid (data_in_valid_dly1 ),
    .data0_in      (row1_col0    ),
```

```
.data1_in      (row1_col1      ),
.data2_in      (row1_col2      ),

.data_max_out   (line1_max      ),
.data_mid_out   (line1_mid      ),
.data_min_out   (line1_min      ),
.data_out_valid(                )
);

//-----
// line2 of (max mid min)
//-----
sort
#(
    .DATA_WIDTH (DATA_WIDTH)
)sort_line2
(
    .clk          (clk          ),    //pixel clk
    .reset_p      (reset_p      ),
    .data_in_valid (data_in_valid_dly1 ),
    .data0_in      (row2_col0     ),
    .data1_in      (row2_col1     ),
    .data2_in      (row2_col2     ),

    .data_max_out  (line2_max     ),
    .data_mid_out  (line2_mid     ),
    .data_min_out  (line2_min     ),
    .data_out_valid(                )
);

//-----
// max of (max mid min)
//-----
sort
#(
    .DATA_WIDTH (DATA_WIDTH)
)sort_max
(
    .clk          (clk          ),    //pixel clk
    .reset_p      (reset_p      ),
    .data_in_valid (data_in_valid_dly2 ),
    .data0_in      (line0_max     ),
    .data1_in      (line1_max     ),
    .data2_in      (line2_max     ),

    .data_max_out  (max_max       ),
    .data_mid_out  (max_mid       ),
```

```
.data_min_out (max_min      ),
.data_out_valid(             )
);

//-----
// mid of (max mid min)
//-----
sort
#(
    .DATA_WIDTH (DATA_WIDTH)
)sort_mid
(
    .clk          (clk          ),    //pixel clk
    .reset_p      (reset_p      ),
    .data_in_valid (data_in_valid_dly2 ),
    .data0_in      (line0_mid    ),
    .data1_in      (line1_mid    ),
    .data2_in      (line2_mid    ),

    .data_max_out  (mid_max      ),
    .data_mid_out  (mid_mid      ),
    .data_min_out  (mid_min      ),
    .data_out_valid(             )
);

//-----
// min of (max mid min)
//-----
sort
#(
    .DATA_WIDTH (DATA_WIDTH)
)sort_min
(
    .clk          (clk          ),    //pixel clk
    .reset_p      (reset_p      ),
    .data_in_valid (data_in_valid_dly2 ),
    .data0_in      (line0_min    ),
    .data1_in      (line1_min    ),
    .data2_in      (line2_min    ),

    .data_max_out  (min_max      ),
    .data_mid_out  (min_mid      ),
    .data_min_out  (min_min      ),
    .data_out_valid(             )
);

//-----
```

```
// matrix 3x3 of mid
//-----
sort
#(
    .DATA_WIDTH (DATA_WIDTH)
)sort_matrix_mid
(
    .clk          (clk          ),    //pixel clk
    .reset_p      (reset_p      ),
    .data_in_valid (data_in_valid_dly3 ),
    .data0_in      (max_min      ),
    .data1_in      (mid_mid      ),
    .data2_in      (min_max      ),

    .data_max_out  (              ),
    .data_mid_out  (matrix_mid   ),
    .data_min_out  (              ),
    .data_out_valid(              )
);

//-----
//result
//-----
assign data_out = matrix_mid;
```

41.2.6 仿真验证

中值滤波处理模块设计完成后，编写仿真验证 testbench 文件，代码如下。

```
`timescale 1ns/1ns
`define CLK_PERIOD 20

module median_filter_tb();

    reg      clk;    //pixel clk
    reg      reset_p;
    reg [7:0] data_in;
    reg      data_in_valid;
    reg      data_in_hs;
    reg      data_in_vs;
    wire [7:0] data_out;
    wire      data_out_valid;
    wire      data_out_hs;
    wire      data_out_vs;

    initial clk = 1'b1;
    always #(`CLK_PERIOD/2) clk = ~clk;
```

```
initial begin
    reset_p = 1'b1;
    data_in = 8'd0;
    data_in_valid = 1'b0;
    data_in_hs = 1'b0;
    data_in_vs = 1'b0;
    #201;
    reset_p = 1'b0;
    #200;

    data_in_vs = 1'b1;
    repeat(480) begin
        #500;
        data_in_hs = 1'b1;
        #500;
        repeat(400*2) begin
            data_in_valid = ~data_in_valid;
            data_in = $random % 256;
            #(`CLK_PERIOD);
        end
        #500;
        data_in_hs = 1'b0;
    end
    data_in_vs = 1'b0;
    #(`CLK_PERIOD);
    data_in_vs = 1'b1;

    #2000;
    $stop;
end

median_filter
#(
    .DATA_WIDTH ( 8 )
)median_filter
(
    .clk          (clk          ),    //pixel clk
    .reset_p      (reset_p      ),
    .data_in       (data_in      ),
    .data_in_valid (data_in_valid),
    .data_in_hs    (data_in_hs   ),
    .data_in_vs    (data_in_vs   ),

    .data_out      (data_out      ),
    .data_out_valid (data_out_valid),
    .data_out_hs    (data_out_hs   ),
    .data_out_vs    (data_out_vs   )
```

```
);  
  
endmodule
```

在仿真波形中任意找几个数据查看算法计算的对错，下图波形分别是对每行 3 个数据进行排序计算最大值，中间值，最小值。可以看出算法计算符合预期。同理可以查看波形中计算的 max_max、max_mid、max_min、mid_max、mid_mid、mid_min、min_max、min_mid、min_min 计算结果是否符合预期。最后看计算出来的中间值是否符合预期。

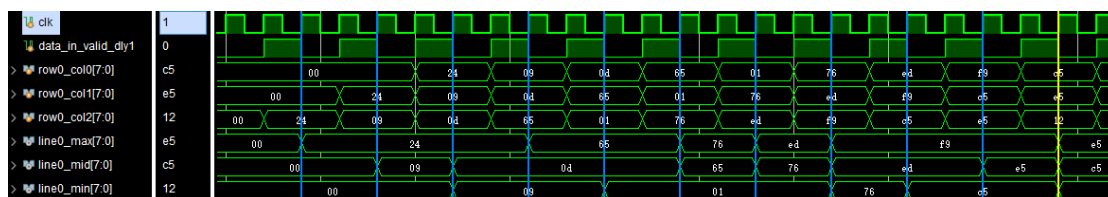


图 41-12 line0 最大值、中间值、最小值

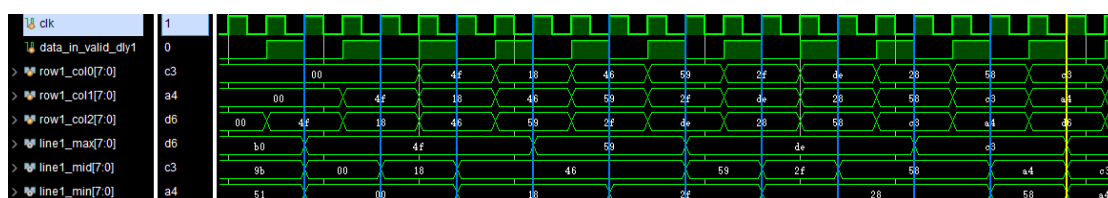


图 41-13 line1 最大值、中间值、最小值

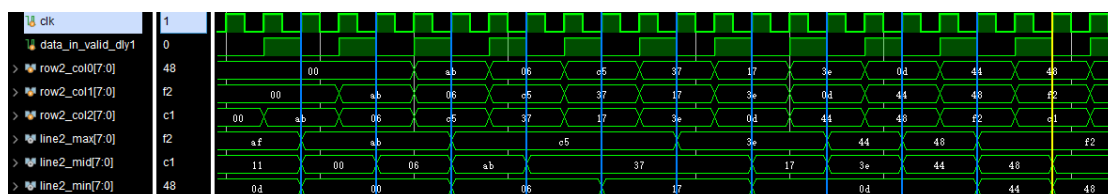


图 41-14 line2 最大值、中间值、最小值

下面是一组 3*3 模板整个中值滤波计算过程的数据波形如下。

(1) 任意选取的一处位置的 3*3 模板数据

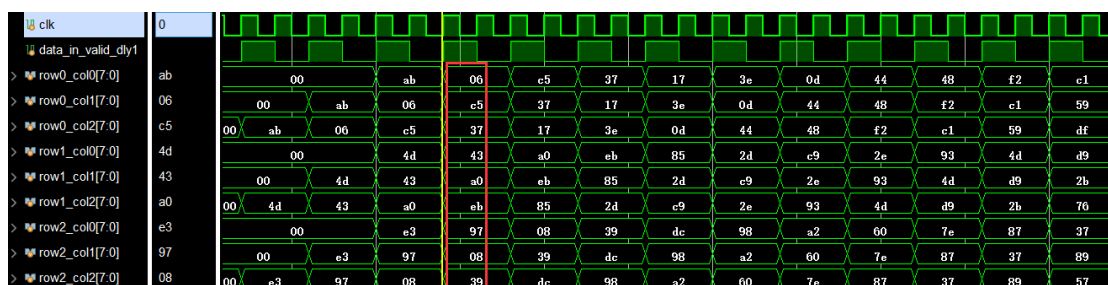


图 41-15 3*3 模板输入数据

(2) 计算的每行 3 个数据的排序后的结果

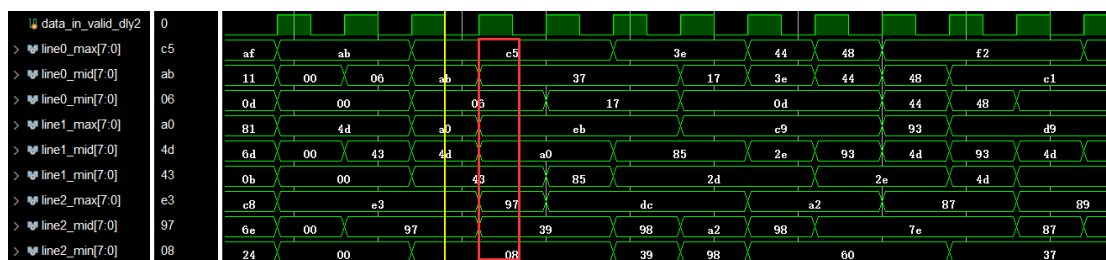


图 41-16 对每行进行最大值、中间值、最小值排序

(3) 分别对 3 行数据中最大值、中间值、最小值的排序后的结果

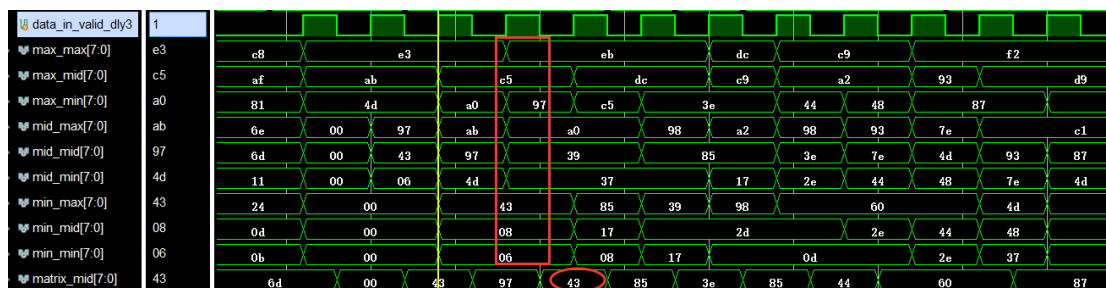


图 41-17 对 3*3 模板中最大值、中间值、最小值排序

(4) 从上面波形数据可以看出，max_min 为 0x97，mid_mid 为 0x39，min_max 为 0x43，根据算法的最后异步得到最后的中值为 0x43，与上面波形最下面的 matrix_mid 的数据是匹配的，验证了设计的正确性。

41.2.7 顶层模块设计

设计并通过仿真验证了中值滤波模块功能的正确性之后，接下来我们只需参考设计总体结构框图将中值滤波模块例化到顶层中即可。

在创建好顶层并综合确认无误后，为设计分配引脚并约束电平。本次设计的引脚分配及约束与彩色图像灰度化设计中一致，如下：

表 41-7 引脚分配表

Pin Name	Signal Name	Pin NO.	Pin Name	Signal Name	Pin NO.
FPGA_UART_RX	uart_rx	W18	TFT_rgb[12]	Display_R1	K3
FPGA_KEY0	reset_n	R4	TFT_rgb[11]	Display_R0	K4
AUD_I2C_SCL	SiI9022_sclk	J3	TFT_rgb[10]	Display_G5	H8
AUD_I2C_SDA	SiI9022_sdat	K2	TFT_rgb[9]	Display_G4	L4
FPGA_LED0	led	P7	TFT_rgb[8]	Display_G3	R8
TFT_clk	Display_PCLK	M1	TFT_rgb[7]	Display_G2	J6
TFT_de	Display_DE	P6	TFT_rgb[6]	Display_G1	J7
TFT_pwm	Display_BL	P5	TFT_rgb[5]	Display_G0	M3
TFT_hs	Display_HSYNC	N1	TFT_rgb[4]	Display_B4	M2
TFT_vs	Display_VSYNC	P1	TFT_rgb[3]	Display_B3	L1
TFT_rgb[15]	Display_R4	M4	TFT_rgb[2]	Display_B2	L2
TFT_rgb[14]	Display_R3	K5	TFT_rgb[1]	Display_B1	J1

TFT_rgb[13]	Display_R2	J5		TFT_rgb[0]	Display_B0	J2
-------------	------------	----	--	------------	------------	----

随后约束引脚电平为 LVC MOS33，在完成以上步骤后便可以生成 bit 文件，将 bit 文件包含在内的硬件资源描述文件导出到 SDK，便可以开始本次设计的板级验证了。

41.3 板级验证

本节将进行基于 ACZ7015 开发板的灰度图像中值滤波设计的板级验证，设计需要使用到 PL 端的串口，因此需要通过 40pin 拓展接口外接 EDA 拓展板。

41.3.1 系统所需硬件

本次设计所需硬件如下，相关模块资料可以点击超链接查看：

1. [ACZ7015 开发板](#) x1
2. [EDA 扩展板](#) x1
3. dc 电源线 x1
4. Type-c 线 x2
5. HDMI 线缆 x1
6. 支持 HDMI 接口的液晶显示器 x1

41.3.2 板级验证需求

灰度图像中值滤波的板级验证，主要验证以下三个方面：

1. 能够正确地全屏点亮 TFT 屏和 HDMI 显示器，显示稳定。
2. 能否正确地显示两个画面，即左侧为带噪点的灰度图像，右侧为中值滤波后图像。
3. 能否正确地定位坐标，即实现在指定的位置显示对应的数据。

41.3.3 硬件连接

本次设计硬件连接如图 41-18 和图 41-19 所示：

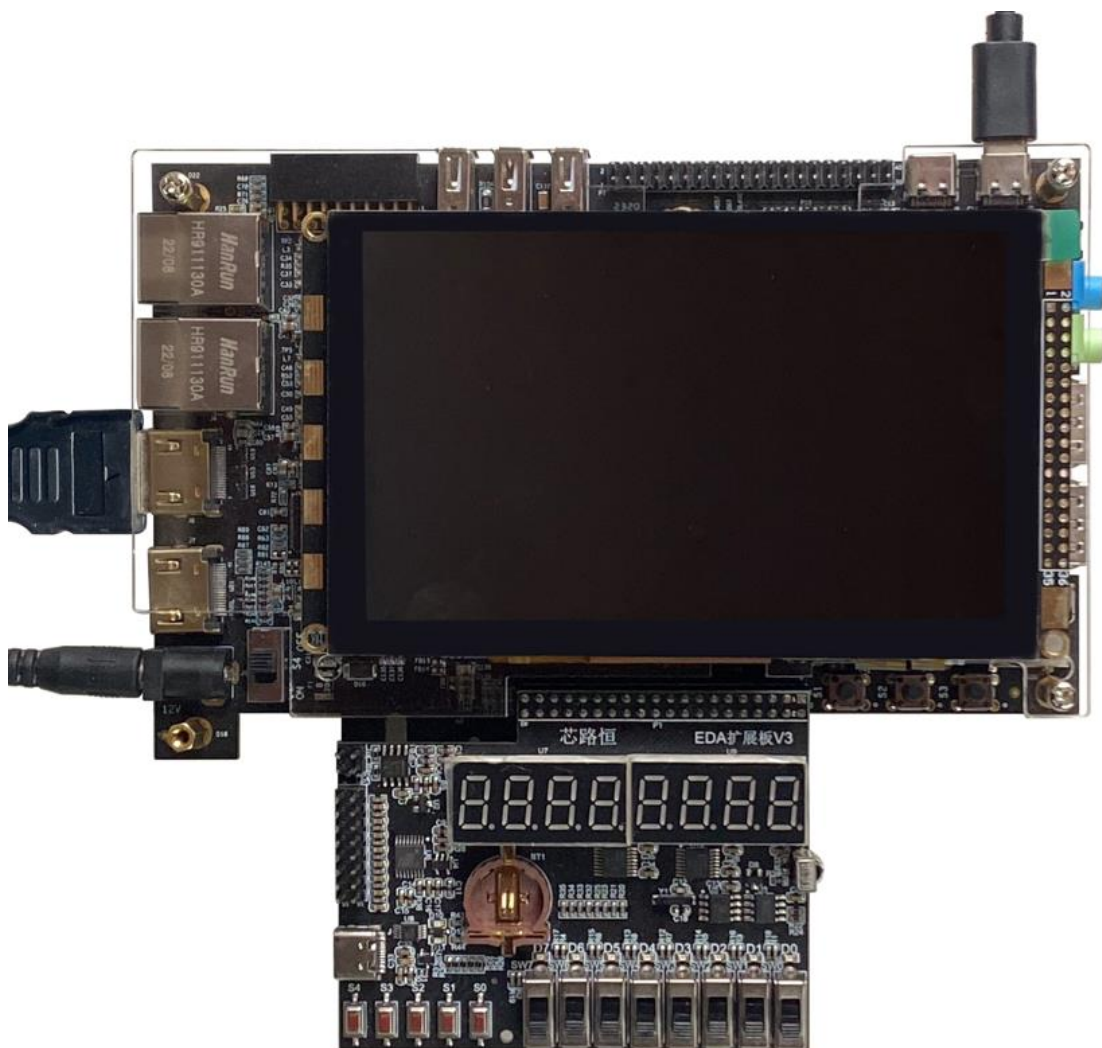


图 41-18 硬件连接

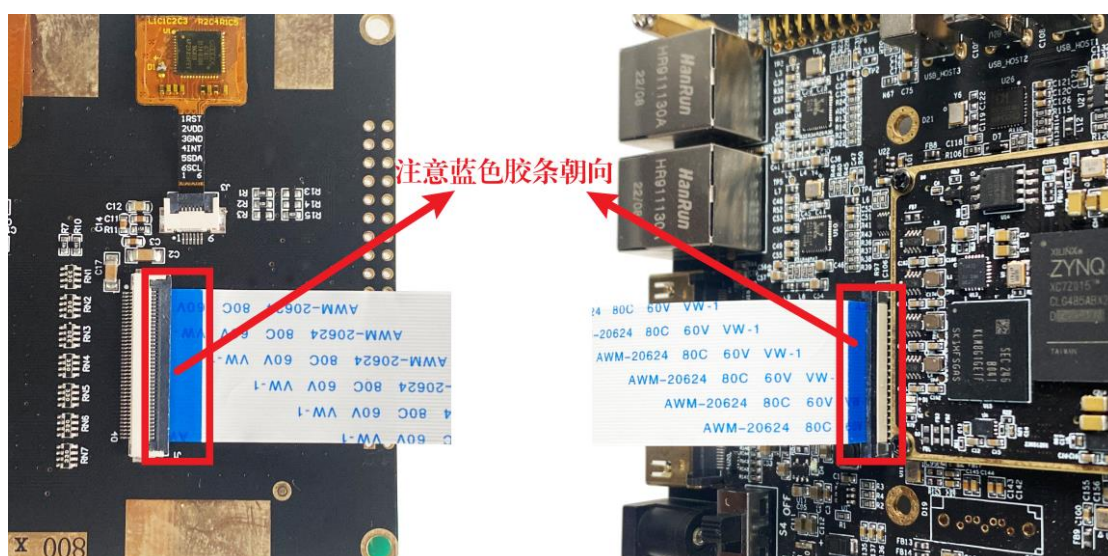


图 41-19 硬件连接

使用 HDMI 线缆连接开发板与 HDMI 显示器，注意线缆一端连接开发板 HDMI_1 接口，一端连接显示器 HDMI 接口；使用软排线连接开发板与 TFT 屏，连接时注意软排线的朝向；将 EDA 拓展板插接在开发板 40pin 接口上，正确连接时，接口应该一一接合；两个 type-c 线一根连接开发板 PS 侧调试接口和主机，一根连接 EDA 拓展板上的串口。

在连接完硬件后将电源拨码开关拨动到对应启动侧，接下来便可以准备烧录了。

41.3.4 显示效果

在 SDK 中创建配置任务，将设计烧录到开发板中。随后，在设备管理器中查询串口端口号，打开小梅哥串口传图工具，设置图像分辨率为 400*480，波特率为 1562500，连接串口。如图 41-20 所示：

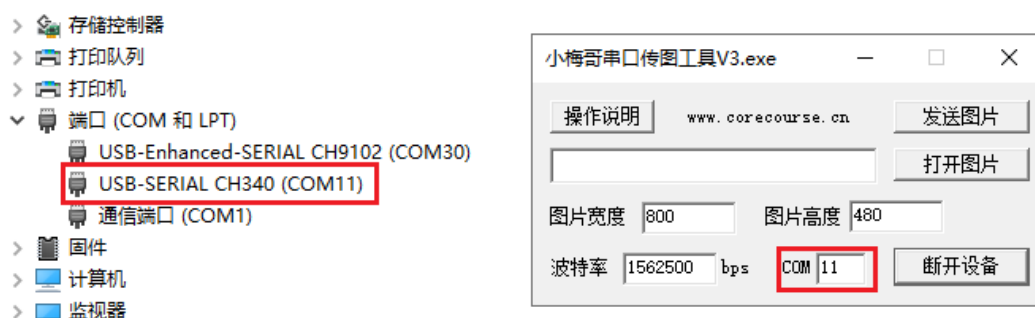


图 41-20 连接小梅哥串口传图工具

接下来选择待传输的图片，这里我们选择的是素材里的分辨率为 400*480 的带椒盐噪声的灰色斑点小猫，如图 41-21 所示：

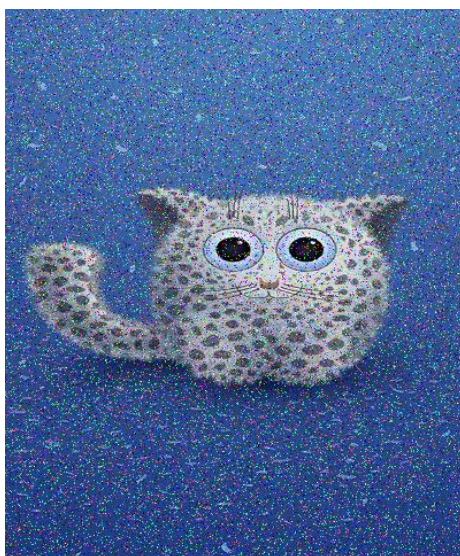


图 41-21 需进行中值滤波处理的图片（椒盐噪声）

将图片数据通过串口传输到开发板后，可以看到 TFT 屏和 HDMI 显示器上开始从上至下扫描成像，最终完整的成像结果如图 41-22 所示：

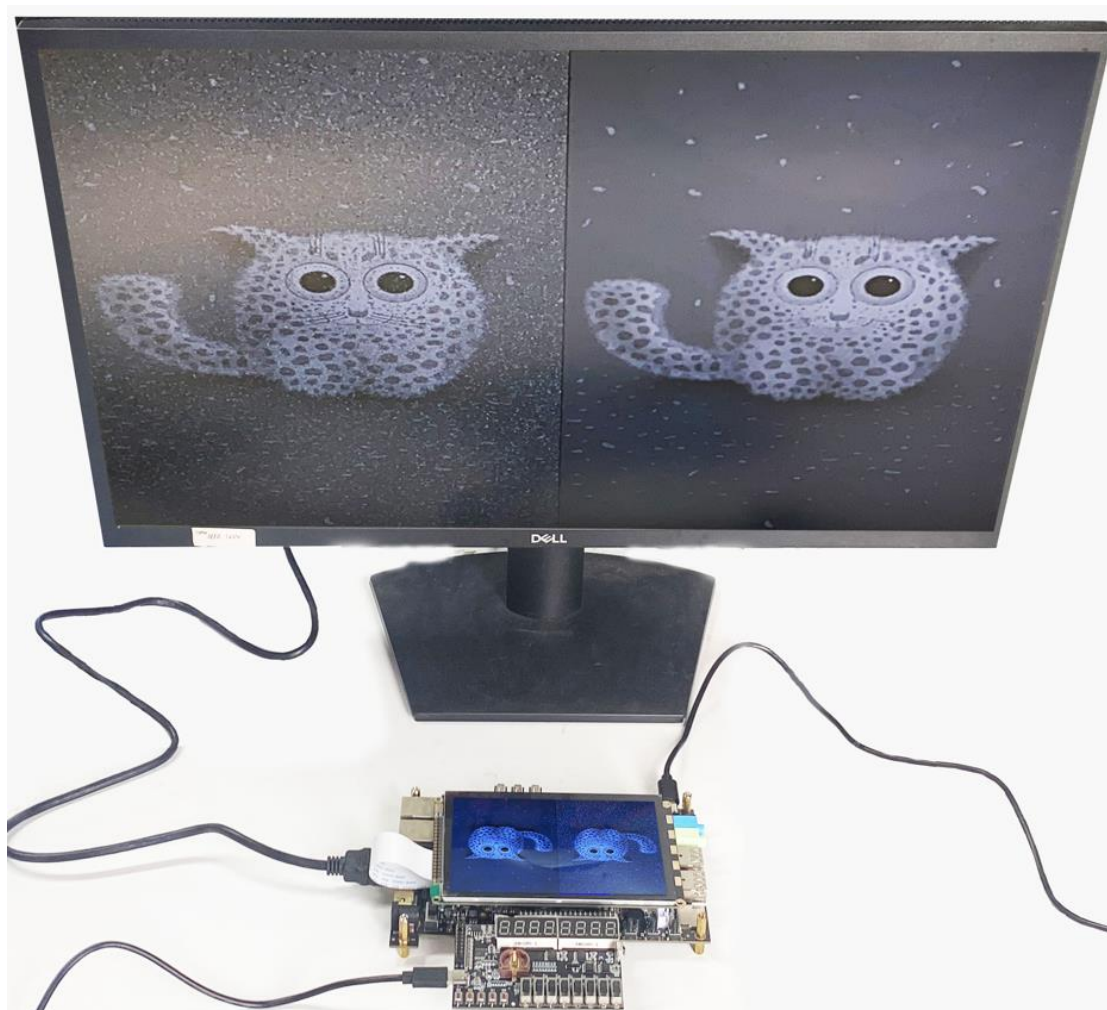


图 41-22 中值滤波处理前（左）后（右）效果对比图（TFT）

可以看到，图像在经过中值滤波后，能够很好地消除掉图像中的噪声。至此，灰度图像中值滤波在 ACZ7015 开发板上的设计及验证就成功完成了。

41.4 总结

将彩色模块的数据流经过灰度处理，得到灰度图像，将灰度图像的数据流经过中值滤波模板的处理，得到中值滤波后的图像。从设计来看，灰度图像的中值滤波可以有效祛除图像上的孤立噪声点。