

1 灰度图像均值滤波设计实现(HDMI 和 TFT 显示)

工程源码	----02_设计实例 ----- acz702_uart_ddr3_tft_hdmi_mean_filter.zip
相关视频课程	----盘 C -----
	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

本节主要是在上一节的基础上，将中值滤波图像处理模块更换成均值滤波处理，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，同时进行均值滤波的处理，然后将均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 显示器上同时显示均值滤波处理前的灰度图像和处理后的灰度图像。

1.1 系统整体设计

本节内容为基于灰度图像的均值滤波，我们希望得到如下实验效果。最终 TFT 和 HDMI 的显示要求如下。

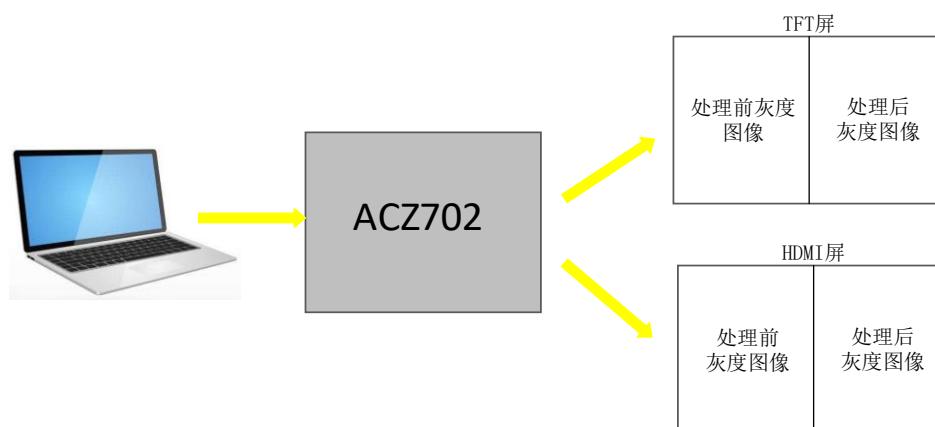


图 1-1 均值滤波实验目标

系统整体设计框图如下。大体结构与“灰度图像中值滤波的设计实现”基本一致，将中值滤波模块更换成了均值滤波处理模块。

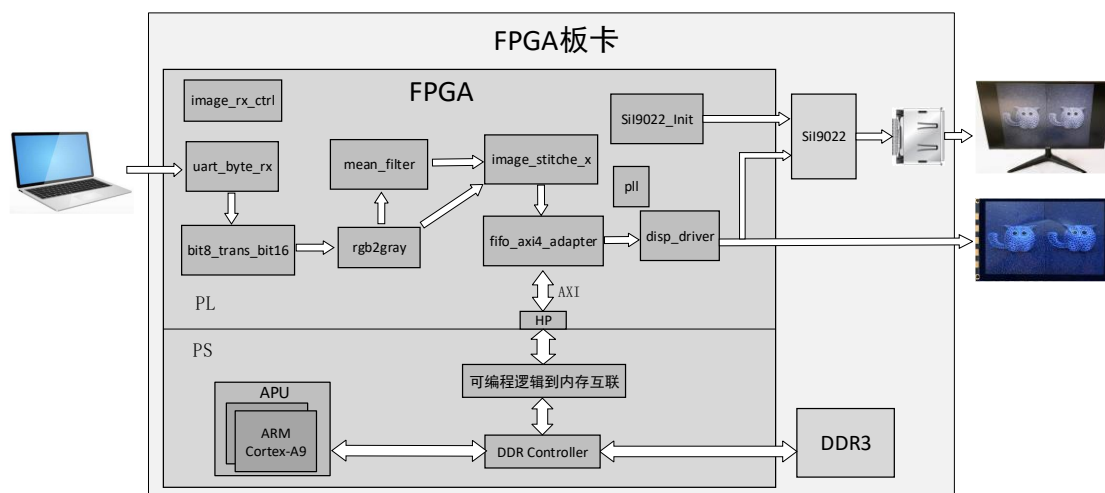


图 1-2 系统设计框图

除了均值滤波模块，其他各模块介绍与前面一样，这里不再重复介绍。

1.2 灰度图像均值滤波处理模块的设计

1.2.1 基本原理

均值滤波是典型的线性滤波算法。和中值滤波相似的是，它同样是通过让待处理目标像素，被一个邻域模板施加影响，达到图像处理的效果。如 3×3 的模板就是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。实际使用时，该算法通过求取模板中的全体像素的平均值来代替原来像素值构成一幅新的图像以实现滤波。而这种算法，在专业领域被命名为邻域平均法。

例如，对于希望处理的当前像素点 (x, y) ，选择一个模板，该模板由其邻近的若干像素组成，先求出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 $g(x, y) = 1/m \sum f(x, y)$ m 为该模板中包含当前像素在内的像素总个数。

虽然均值滤波可以在一定程度上去除图像上的噪点，但专家的理论分析结合前人的实践经验还告诉我们，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。

1.2.2 实现方法

如下所示，为一个 3×3 的图像模板。

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

图 1-3 均值滤波图像模板

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:

$$g(x,y) = 1/9 * (f(x-1,y-1) + f(x,y-1) + f(x+1,y-1) + f(x-1,y) + f(x,y) + f(x+1,y) + f(x-1,y+1) + f(x,y+1) + f(x+1,y+1))$$

由上式我们看出 (x,y) 点的 3×3 像素点的均值等于 3×3 模板的 9 个点的像素值之和除以 9。由于 FPGA 不擅长算乘除法, 为了简便运算只将目标像素周围八个点求和然后除以 8 (即右移三位), 取代中间像素点。

这样, 公式可以化简如下:

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

FPGA 实现步骤该算法步骤如下:

第一步: 形成 3×3 矩阵像素, 这个在中值滤波设计中已经有讲过, 这节可以直接使用。

第二步: 求周围邻域八个点的像素值之和

第三步: 将第二步结果右移三位 (相当于除以 8) 得到结果。

1.2.3 模块接口设计

了解了均值滤波算法后, 我们同样可以明确均值滤波模块接口如下:

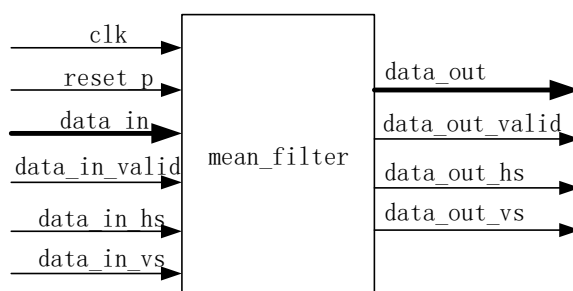


图 1-4 均值滤波模块输入输出接口图

表 1-1 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

1.2.4 实现过程

3*3 模板数据的产生在上节中值滤波处理已经讲解，根据算法的步骤，只需要计算邻域八个点的像素值求均值即可。具体代码实现如下。

第 1 部分：端口定义

```
module mean_filter
#(
    parameter DATA_WIDTH = 8
)
(
    input          clk,          //pixel clk
    input          reset_p,
    input [DATA_WIDTH-1:0] data_in,
    input          data_in_valid,
    input          data_in_hs,
    input          data_in_vs,

    output [DATA_WIDTH-1:0] data_out,
    output reg      data_out_valid,
    output reg      data_out_hs,
    output reg      data_out_vs
);
//line data
wire [DATA_WIDTH-1:0] line0_data;
wire [DATA_WIDTH-1:0] line1_data;
wire [DATA_WIDTH-1:0] line2_data;
//matrix 3x3 data
reg [DATA_WIDTH-1:0] row0_col0;
reg [DATA_WIDTH-1:0] row0_col1;
reg [DATA_WIDTH-1:0] row0_col2;

reg [DATA_WIDTH-1:0] row1_col0;
reg [DATA_WIDTH-1:0] row1_col1;
```

```
reg [DATA_WIDTH-1:0] row1_col2;  
  
reg [DATA_WIDTH-1:0] row2_col0;  
reg [DATA_WIDTH-1:0] row2_col1;  
reg [DATA_WIDTH-1:0] row2_col2;
```

第 2 部分：移位寄存器例化

```
//  
reg[DATA_WIDTH+2:0] data_out_tmp;  
reg data_in_valid_dly1;  
reg data_in_hs_dly1;  
reg data_in_vs_dly1;  
  
//3xline data  
shift_register_2taps  
#(  
    .DATA_WIDTH ( DATA_WIDTH )  
)shift_register_2taps(  
    .clk          (clk          ),  
    .shiftin      (data_in      ),  
    .shiftin_valid (data_in_valid ),  
  
    .shiftout      (          ),  
    .taps0x        (line0_data  ),  
    .taps1x        (line1_data  )  
);  
assign line2_data = data_in;
```

第 3 部分：建立滤波像素矩阵

```
//-----  
// 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
end
```

第 4 部分：均值滤波运算

```
always @(posedge clk)
begin
  data_in_valid_dly1 <= data_in_valid;
```

```
data_in_hs_dly1    <= data_in_hs;
data_in_vs_dly1    <= data_in_vs;
end

//-----
//result
//-----
always @(posedge clk or posedge reset_p) begin
    if(reset_p)
        data_out_tmp <= 'd0;
    else if(data_in_valid_dly1)
        data_out_tmp <= (row0_col0 + row0_col1 + row0_col2 +
                        row1_col0 +          row1_col2 +
                        row2_col0 + row2_col1 + row2_col2 );
end

assign data_out = data_out_tmp>>3;

always @(posedge clk)
begin
    data_out_valid <= data_in_valid_dly1;
    data_out_hs    <= data_in_hs_dly1;
    data_out_vs    <= data_in_vs_dly1;
end

endmodule
```

1.2.5 仿真验证

均值滤波处理模块设计完成后，编写仿真验证 testbench 文件，仿真验证代码与中值滤波基本是一样的，将例化的中值滤波模块替换成均值滤波模块即可，具体代码如下。

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

module mean_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

mean_filter
#(
    .DATA_WIDTH ( 8 )
)mean_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

```

运行仿真，任意找到一个仿真地方进行放大观察，仿真波形如下。

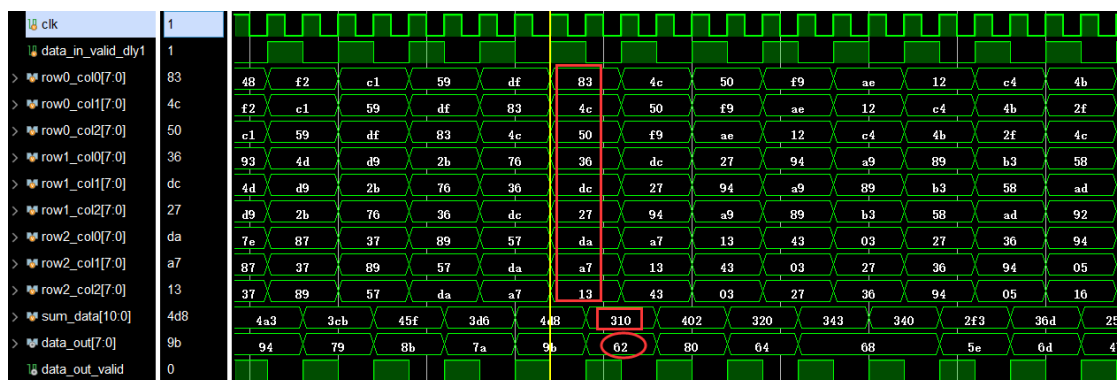


图 1-5 仿真波形任意处核验设计思路

从上面波形数据可以看出，当前 3*3 的模板数据如下

0x83	0x4c	0x50
0x36	0xdc	0x27
0xda	0xa7	0x13

图 1-6 被观察的像素点及其邻域

根据 FPGA 计算均值的公式可得，周围邻域八个点的像素值之和为 $(0x83+0x4c+0x50+0x36+0x27+0xda+0xa7+0x13)=0x310$ ，然后右移 3 位得到 0x62。仿真中 sum_data 和 data_out 与该计算结果一致，验证了设计的正确性。

1.2.6 顶层设计与引脚约束

完成子模块的设计后，顶层的设计就相对容易些，根据整体设计框图对子模块端口信号进行连接即可，这里就不做详细讲解。

连接完硬件后，为设计分配引脚并约束电平，本次设计引脚分配表如下：

表 1-2 引脚分配表

Pin Name	Signal Name	Pin NO.	Pin Name	Signal Name	Pin NO.
----------	-------------	---------	----------	-------------	---------

FPGA_UART_RX	uart_rx	K16		TFT_rgb[12]	Display_R1	V16
FPGA_KEY0	reset_n	F20		TFT_rgb[11]	Display_R0	T15
AUD_I2C_SCL	SiI9022_sclk	T10		TFT_rgb[10]	Display_G5	V20
AUD_I2C_SDA	SiI9022_sdat	R14		TFT_rgb[9]	Display_G4	U17
FPGA_LED0	led	T14		TFT_rgb[8]	Display_G3	V18
TFT_clk	Display_PCLK	U15		TFT_rgb[7]	Display_G2	T16
TFT_de	Display_DE	W15		TFT_rgb[6]	Display_G1	R16
TFT_pwm	Display_BL	R17		TFT_rgb[5]	Display_G0	U19
TFT_hs	Display_HSYNC	U14		TFT_rgb[4]	Display_B4	Y19
TFT_vs	Display_VSYNC	W14		TFT_rgb[3]	Display_B3	W18
TFT_rgb[15]	Display_R4	W20		TFT_rgb[2]	Display_B2	Y18
TFT_rgb[14]	Display_R3	W19		TFT_rgb[1]	Display_B1	W16
TFT_rgb[13]	Display_R2	V17		TFT_rgb[0]	Display_B0	Y17

分配并约束完引脚后，生成 bit 流，将包含 bit 的硬件资源描述文件一起导出到 SDK 中，便可以连接硬件准备进行板级验证了。

1.3 板级验证

本节将进行基于 ACZ702 开发板的灰度图像均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。

1.3.1 系统所需硬件

- 1.ACZ702 开发板
- 2.电源线一根（可选）
- 3.Type-c 线两根
- 4.EDA 拓展板一个
- 5.HDMI 线缆一根
- 6.支持 HDMI 接口的液晶显示器一台

1.3.2 板级验证需求

灰度图像均值滤波的板级验证，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，右侧为均值滤波后图像。
- 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。

1.3.3 硬件连接

本次设计硬件连接如图 1-7 和图 1-8 所示：

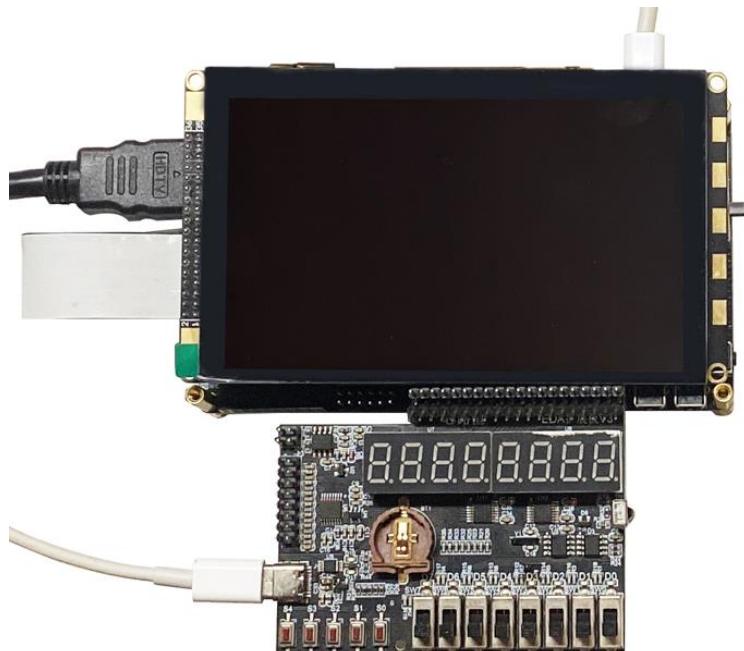


图 1-7 硬件连接

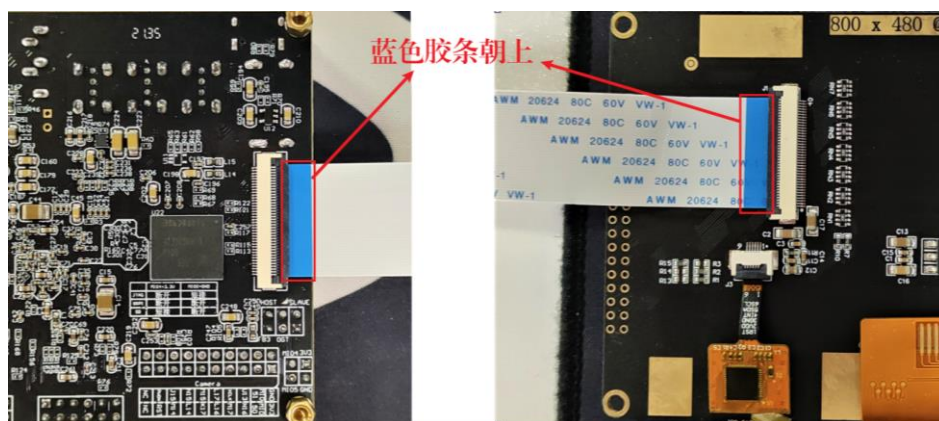


图 1-8 硬件连接

由于本次设计对供电要求不高，因此可以仅使用 type-c 线供电，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。

1.3.4 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 1-9 所示：



图 1-9 连接小梅哥串口传图工具

接下来选择待传输的图片，这里我们还是选择素材里分辨率为 400*480 的带椒盐噪声的灰色斑点小猫，如图 1-10 所示：

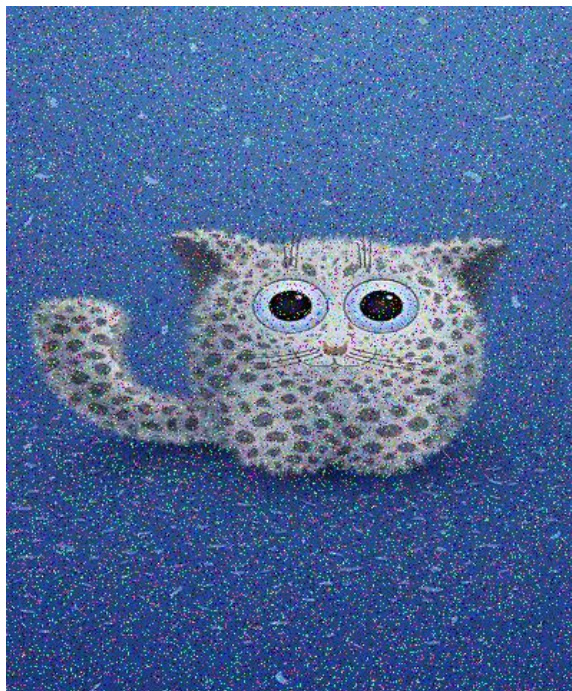


图 1-10 待传输图片（椒盐噪声）

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 1-11 所示：

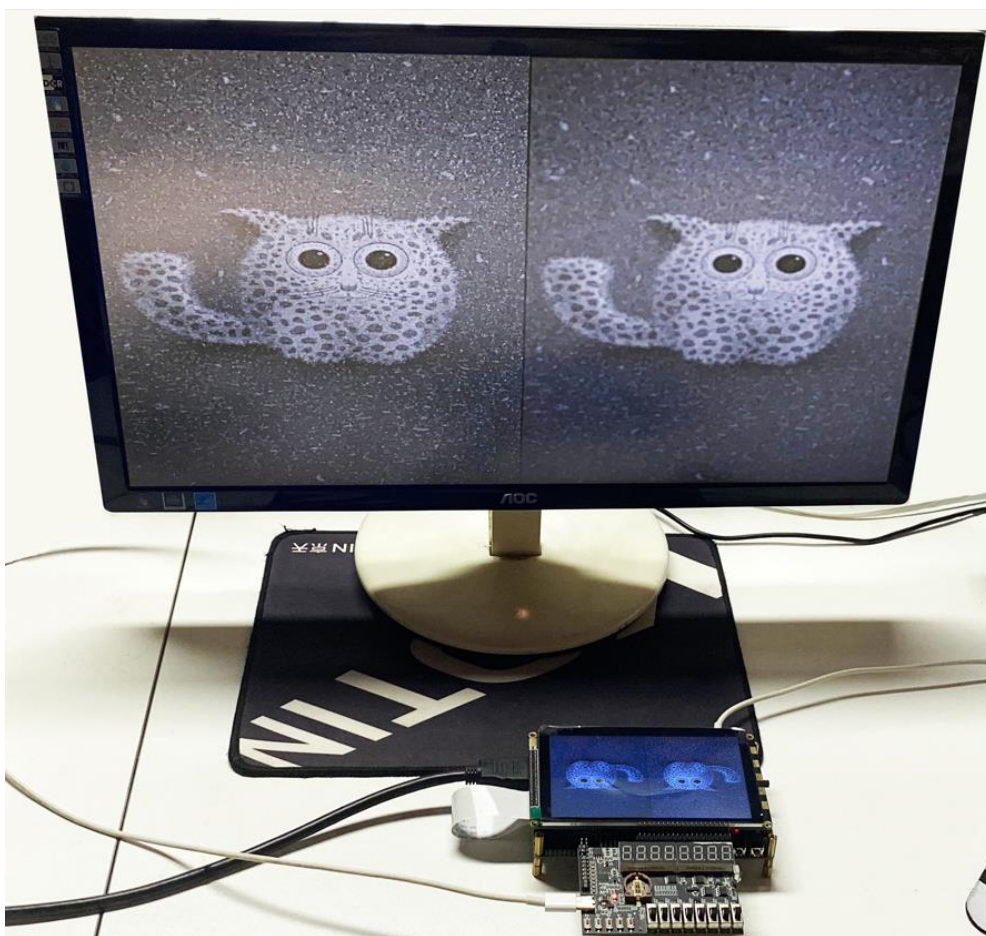


图 1-11 均值滤波处理前（左）后（右）效果对比图（TFT）

通过对比可以看出，显示屏左右两边分别显示的是彩色图像灰度化图像数据和经过均值滤波处理之后的灰度图像。均值滤波略去了部分噪声，但不如上章的中值滤波效果好。均值滤波后，图片变得模糊，其实是做了平滑处理，像素值高的会被拉低，像素值低的会被拉高，趋向于一个平均值。至此，灰度图像中值滤波在 ACX720 开发板上的设计及验证就成功完成了。

1.4 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均值滤波模板的处理，得到均值滤波后的图像。从设计来看，灰度图像的均值滤波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。