

1 灰度图像高斯滤波设计实现(HDMI 和 TFT 显示)

工程源码	----02_设计实例 ----- acx720_uart_ddr3_tft_hdmi_gaussian_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，更换图像处理模块，更换为高斯滤波处理模块，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，同时进行高斯滤波的处理，然后将高斯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 显示器上同时显示高斯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。

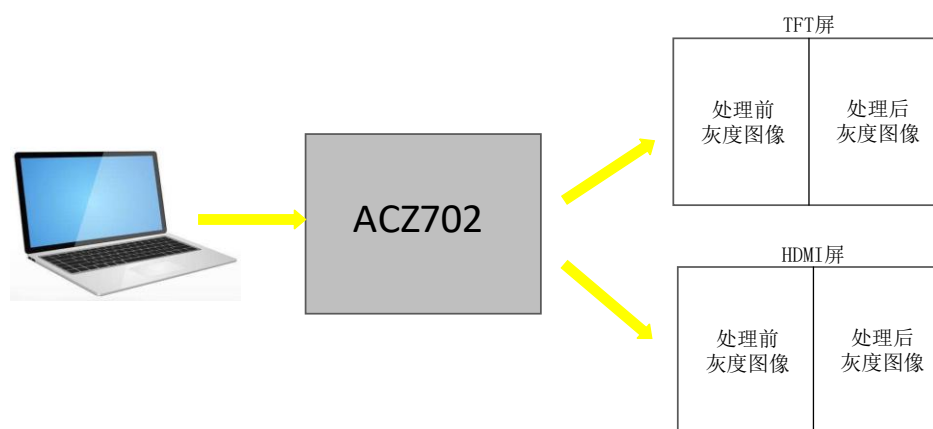


图 1-1 高斯滤波实验目标

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高斯滤波处理模块。

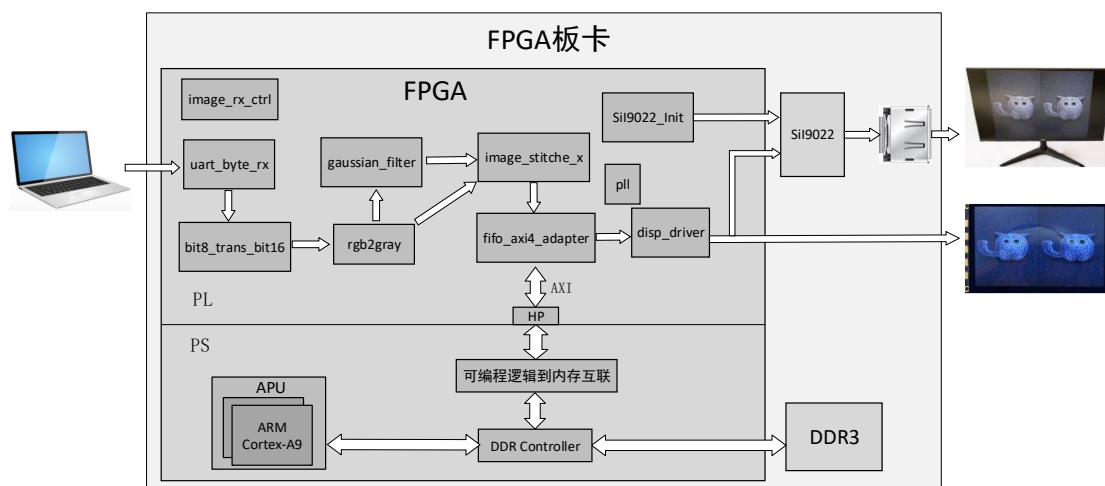


图 1-2 系统设计框图

除了高斯滤波模块，其他各模块介绍与前面一样，这里不再重复介绍。

1.2 灰度图像高斯滤波处理模块的设计

1.2.1 基本原理

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。通俗地讲，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。

一维高斯分布：

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

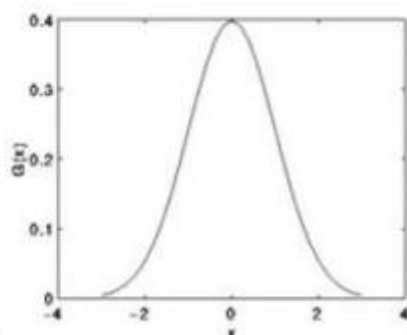


图 1-3 一维高斯分布

二维高斯分布：

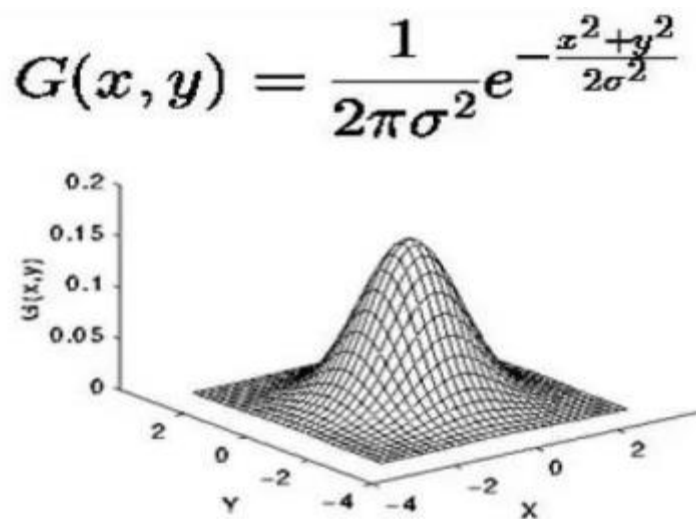


图 1-4 二维高斯分布

高斯滤波后图像被平滑的程度取决于标准差。它的输出是邻域像素的加权平均，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。

高斯滤波被用作平滑滤波器的本质原因是，它是一个低通滤波器，而且大部分基于卷积平滑滤波器都是低通滤波器。

由于高斯滤波算法克服了边界效应，因而滤波后的图像较好。

1.2.2 实现方法

均值滤波算法中所有参与运算像素点权重一致，而高斯滤波算法则不同。高斯滤波以中心像素点为圆心，按距离中心像素点的距离，给邻域点赋予权重值。距离越近则权重值越高，距离越远则权重值越低。根据权重公式，5x5 高斯滤波算子权重表格如下：

表 1-1 高斯滤波 5x5 算子

(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

而 3x3 高斯滤波算子权重表格如下：

表 1-2 高斯滤波 3x3 算子

	1	2	1
(1/16)*	2	4	2
	1	2	1

表 1-3 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)	$(x+1,y)$
$(x-1,y+1)$	$(x,y+1)$	$(x+1,y+1)$

如果用 $f(x,y)$ 表示待处理像素 (x,y) 点的像素值；而用 $g(x,y)$ 表示 (x,y) 点经过高斯滤波处理后的值；

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计算公式可以如下表示：

$$g(x,y) = (1/16) * (f(x-1,y-1) + 2f(x,y-1) + f(x+1,y-1) + 2f(x-1,y) + 4f(x,y) + 2f(x+1,y) + f(x-1,y+1) + 2f(x,y+1) + f(x+1,y+1))$$

如果用模板扫描图像中的每一个像素，就可以完成整幅图像的高斯滤波。

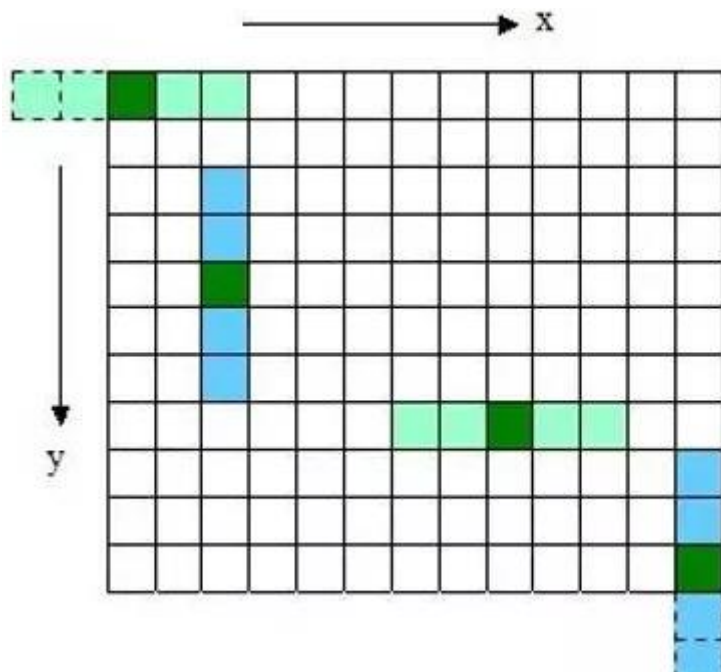


图 1-5 模板扫描像素点示意图

如图 1-5 所示，如果使用高斯滤波模板依次（从左到右，从上到下）划过每一个屏幕图像待输出点的对应存储空间，则就最终可以完成一帧图像的高斯滤波。因为原始图像，是按逐行扫描，每扫完一行换下一行的规律进行的图像输出，所以高斯滤波模板的滑动，也可以按这一输出规则，同步对这些数据进行处理而实现。

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：根据公式计算乘加和

第三步：将第二步结果右移 4 位（即除以 16）得到结果。

1.2.3 模块接口设计

和前面的章节讲解内容一致，高斯滤波数据输入接口为 16 位灰度值，输出接口为滤波后得到的 data_out 值。

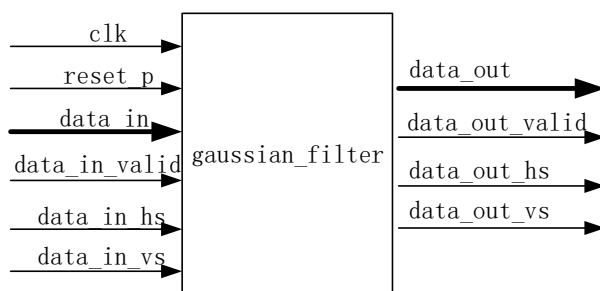


图 1-6 高斯滤波输入输出接口

表 1-4 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	图像处理后数据输出，数据位宽与 data_in 相同
data_out_valid	O	图像处理后像素数据有效标识，为 1 表示当前 data_out 有效
data_out_hs	O	图像处理后行信号
data_out_vs	O	图像处理后场信号

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，只需要计算乘加和，然后除以 16 即可。具体代码实现如下。

```
module gaussian_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 reg[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;
//
reg  data_in_valid_dly1;
reg  data_in_hs_dly1;
reg  data_in_vs_dly1;
reg [DATA_WIDTH+3:0] sum_data;

//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;

//-----
```

```
// 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
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

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)
        sum_data <= 'd0;
    else if(data_in_valid_dly1)
        sum_data <= (row0_col0 + row0_col1*2 + row0_col2 +
                    row1_col0*2 + row1_col1*4 + row1_col2*2 +
                    row2_col0 + row2_col1*2 + row2_col2);
end

assign data_out = sum_data>>4;

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

高斯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 gaussian_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

gaussian_filter
#(
    .DATA_WIDTH ( 8 )
)gaussian_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。

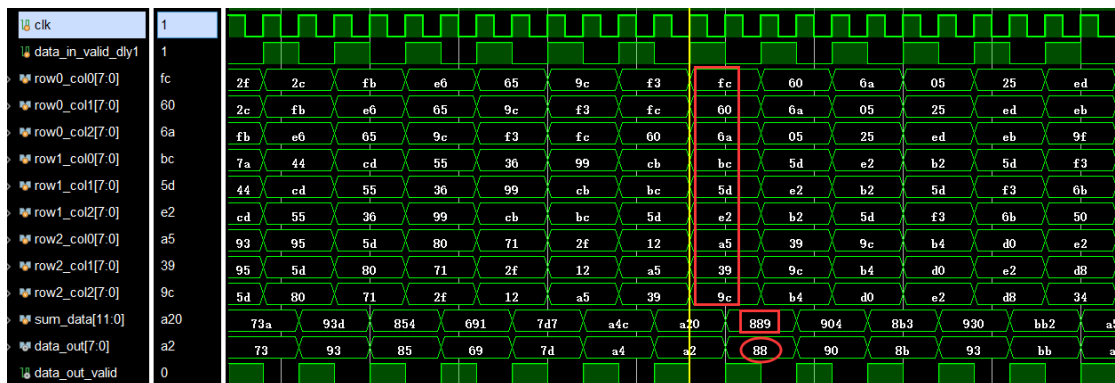


图 1-7 仿真波形数据

从上面波形数据可以看出，当前 3*3 的模板图像数据如下

0xfc	0x60	0x6a
0xbc	0x5d	0xe2
0xa5	0x39	0x9c

图 1-8 3*3 模板

根据 FPGA 计算公式可得，高斯滤波处理后的数据计算结果为
 $(1*0xfc+2*0x60+1*0x6a+2*0xbc+4*0x5d+2*0xe2+1*0xa5+2*0x39+1*0x9c) = 0x889$, 然后右移 4 位得到 0x88。仿真中 sum_data 和 data_out 与该计算结果一致，说明模块设计成功。

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 1-5 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

灰度图像高斯滤波的板级验证，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，右侧为高斯滤波后图像。
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错误!未找到引用源。和错误!未找到引用源。所示：

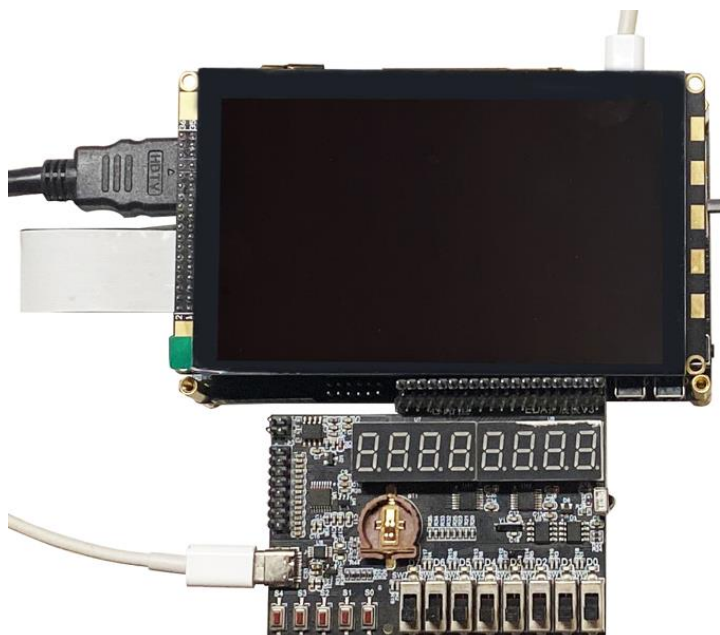


图 1-9 硬件连接

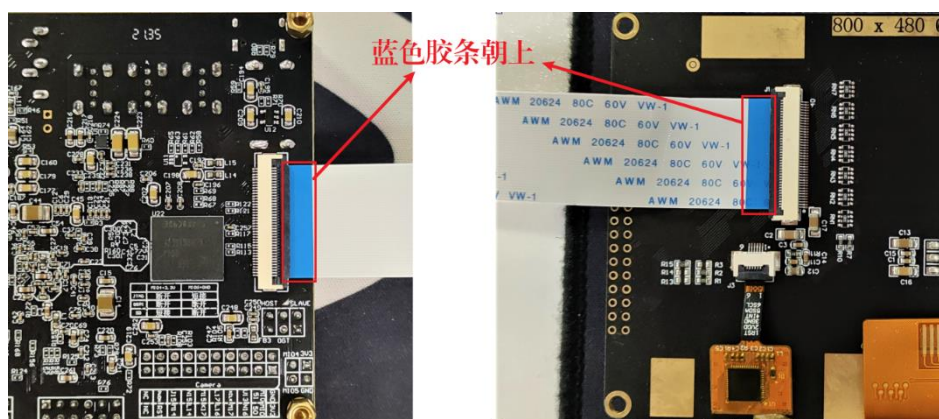


图 1-10 硬件连接

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。如错误!未找到引用源。所示：



图 1-11 连接小梅哥串口传图工具

接下来选择待传输的图片，这里我们选择素材里分辨率为 400*480 的带高斯噪声的灰色斑点小猫，如错误!未找到引用源。所示：

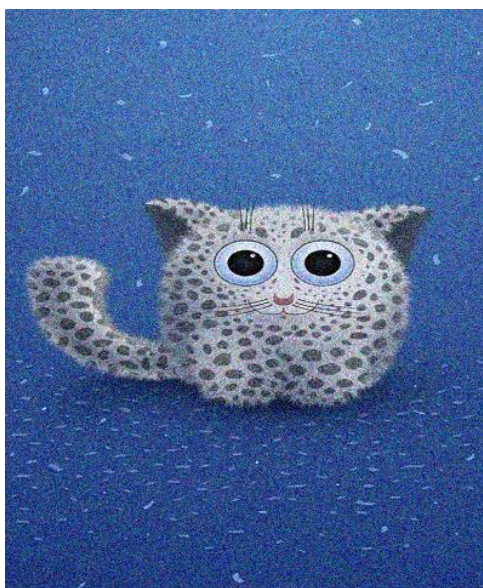


图 1-12 待传输图片（高斯噪声）

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错误!未找到引用源。所示：

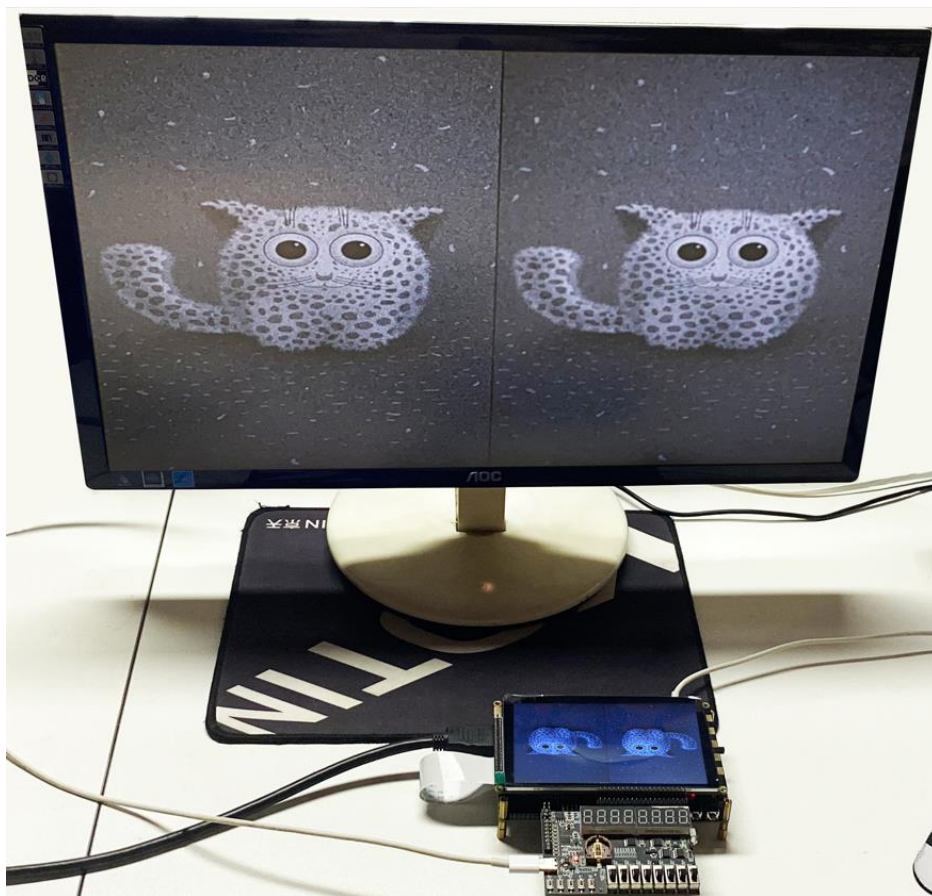


图 1-13 高斯滤波处理前（左）后（右）效果对比图（TFT）

通过对比可以看出，显示屏上左右两边分别显示的是彩色图像灰度化图像数据和经过高斯滤波处理之后的灰度图像。采用高斯滤波处理后成功滤除掉了部分噪音，但是图像会由于平滑处理而变得较为模糊。

至此，灰度图像高斯滤波在 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高斯滤波模板的处理，得到高斯滤波后的图像。从设计来看，高斯滤波相对于均值滤波（mean filter）它的平滑效果更柔和，而且边缘保留的也更好。GAUSS 滤波算法克服了边界效应，因而滤波后的图像较好。