第 5 章

FPGA 图像前处理

5.1 色彩滤波矩阵 IP 核的仿真

5.1.1 色彩滤波矩阵介绍

1. 科普 CFA

CFA(Color Filter Array,色彩滤波阵列)也就是我们常说的 CMOS 色彩滤镜,应该说是一个比较重要、厂商在宣传的时候也会偶尔提及的东西。但是对于这个东西如何起作用,不同的排列又有什么优缺点,可能很多人就不太清楚了。拜耳阵列是当前应用最成熟最广泛的 CFA,下面就以拜耳阵列的工作原理为例进行介绍。

对于彩色图像,需要采集多种最基本的颜色,如红(R)、绿(G)和蓝(B)三种颜色。最简单的方法就是采用滤镜的方法,红色的滤镜透过红色的波长,绿色的滤镜透过绿色的波长,蓝色的滤镜透过蓝色的波长。如果要采集 R、G 和 B 三个基本色,则需要三块滤镜,这样价格昂贵,且不好制造,因为三块滤镜都必须保证每一个像素点都对齐。柯达公司的科学家 Bryce Bayer(拜耳)发明了以其名字命名的拜耳阵列被广泛运用于数字图像传感器中,拜耳阵列并不需要每个像素都放置三块滤镜,它有效地利用了人眼对绿色比较敏感、但对红色和蓝色相对不敏感的特性,在每个像素点只放置一个滤镜的情况下,让绿色的滤镜间隔出现,而红色和蓝色滤镜只需要在每 2 像素×2 像素矩阵放置一块。虽然拜耳阵列采集到的每个像素都只有一个色彩的图像数据,但是通过后端的数字算法(即我们这个实例要实现的 CFA 算法)可以实现每个像素点的其他色彩的重建。

拜耳阵列俗称为“马赛克传感器”,如图 5.1 所示,这种排列方式看起来的确有点像花花绿绿的马赛克晶格。

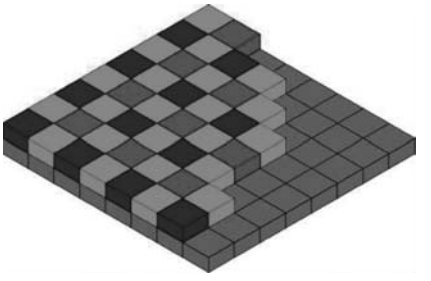


图 5.1 马赛克晶格(见彩插)

很明显可以看到拜耳阵列是由一行 RGRGRG……和一行 BGBGBG……交错排列而成,每一个像素点只能读取单独的颜色信息。其中绿色像素的采样频率是输出像素的 1/2,红、蓝色像素的采样频率是输出像素的 1/4,故有拜耳阵列传感器的分辨率是由绿

色像素决定的这一说法。

拜耳阵列传感器采样生成的图像要输出我们常见的全色彩图像必须经过反马赛克运算——但这跟我们平时俗称的“猜色”的字面意义不同，拜耳阵列的颜色并不是猜的，而是每个 2×2 方块经过 9 次矩阵运算计算出来的，也就是说不存在猜这回事，每个像素的颜色其实是一个确定值(矩阵运算是线性运算，这并不是一个混沌系统)。但没有争议的是拜耳阵列确实存在欠采样问题，这也使得它会出现摩尔纹和伪色(摩尔纹出现的原因就是输入信号的最高频率成分超过了传感器的奈奎斯特采样定理中的极限值，也就是说传感器的高频采样能力存在一些不足)，100％查看时的画质也不是特别理想。

但是，看一个结构或者说架构是否强势，重点要看其成熟度与可扩增性，有时候一些“暴力美学”解决起问题来反而十分优雅。

拜耳阵列就是这么一个典型，简单的结构与成熟的工艺让它“堆”起像素来十分容易，只要光电管足够，3000 万、4000 万、5000 万甚至上亿像素也没问题。虽然结构本身存在欠采样问题，但理论上来说，当拜耳阵列传感器的实际像素数超越无 CFA 或 X3 这种传感器的输出像素 2.8 倍的时候，在输出同样大小的图片时便可以获取超越全色传感器的分辨率和 100％查看画质。

2. CFA 插值运算

CFA 虽然解决了像素问题，却带来了新的色彩问题。从图像传感器采集过来的 Bayer Raw 数据，每个像素只提供一个色彩的颜色数据(Red、Green 或 Blue)。但是，最终显示或还原每个像素的色彩信息，却是每个像素都需要具备 R、G、B 三色数据。怎么解决这个问题？没错，色彩插值！在 CFA 发明之前，前人也的确是通过每个像素分别摆放 R、G、B 三个滤光片来获得每个像素的 R、G、B 数据。但是，聪明的 Kodak 科学家 Bryce Bayer 发现，通过 CFA 方式进行后期插值可以几乎不失真地还原每个像素的 R、G、B 信息。用最节能环保的方式实现性能相当的产品，这才是工程实现的最高境界。

一种常见的 Bayer Raw 图像的色彩排布如图 5.2 所示，插值的基本原理很简单，每个像素没有采集到的色彩，可以通过周边的对应色彩值进行一定的运算获得。

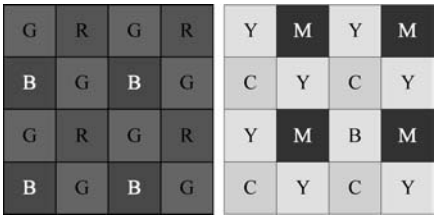


图 5.2　Bayer Raw 色彩阵列(见彩插)

在 Xilinx 的 Vivado 中，提供了 Sensor Demosaic IP 用于实现 CFA 的插值运算，内部功能框图如图 5.3 所示。多行图像的缓存，对不同像素值位置的判断，相应邻近色彩数据的运算处理，边界上还需要特殊的判断和处理，整个控制上还是略有些复杂。作者早年由于项目需要，做过一个 CFA 插值的实现，各种分支判断处理，极费脑力。好在今天 Xilinx 提供的这个 Sensor Demosaic IP 省去了设计者大量的时间和精力。

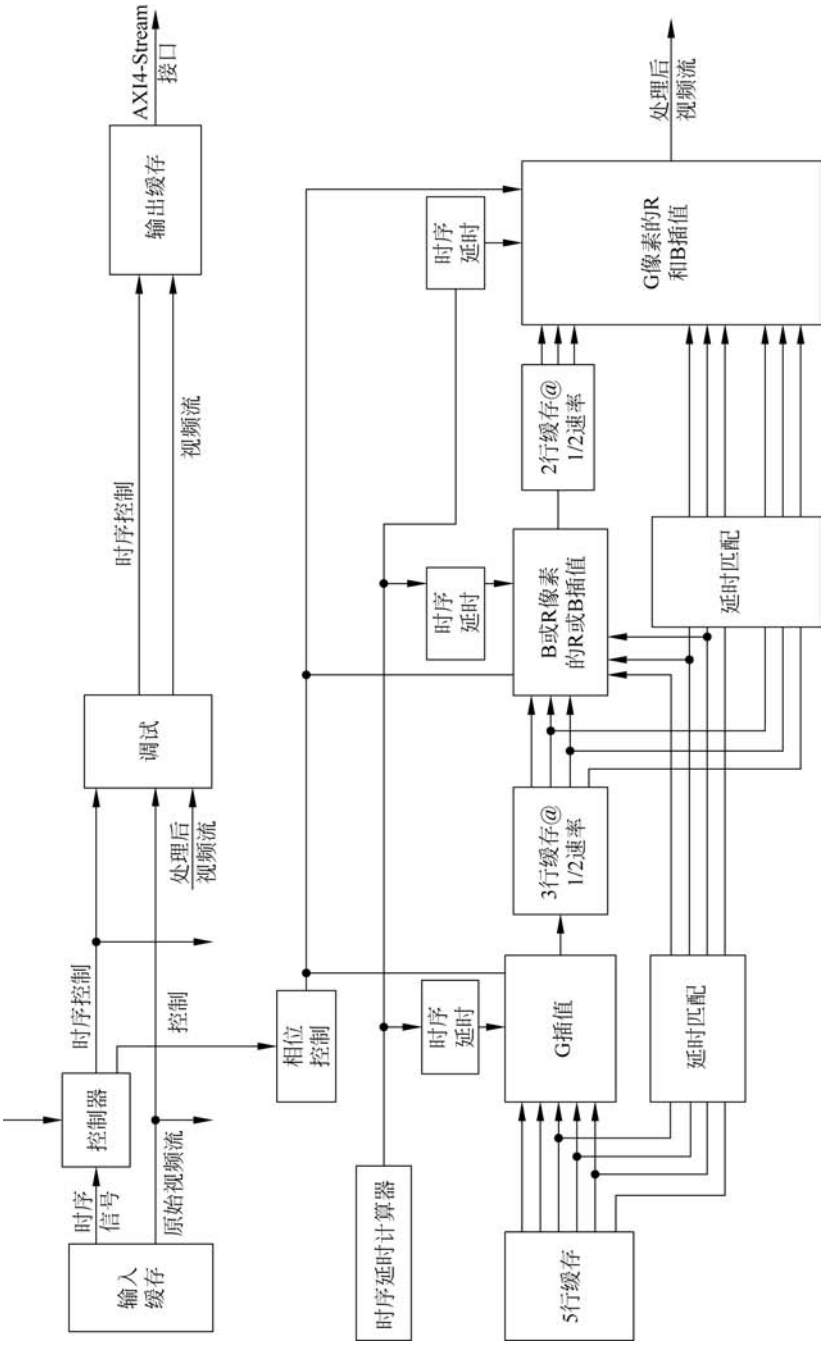


图 5.3　Sensor Demosaic IP 内部功能框图

5.1.2 基于 MATLAB 的 CFA 处理

在 MATLAB 中，调用函数 demosaic 可以实现 Bayer RAW 转 RGB 图像的功能。运行工程文件夹 at7_img_ex03\matlab 下的 MATLAB 脚本 beyer2RGB_matlab.m，可以实现 Bayer Raw 格式的图像 mandi_bayer_raw.tif 转换为 RGB 图像 mandi_rgb.tif。MATLAB 源码如下：

```
clc;clear all;close all;

% load origin bayer raw image
I = imread('mandi_bayer_raw.tif');

% convert a bayer raw image to RGB image
J = demosaic(I,'bggr');

% write image as .tif
imwrite(J,'mandi_rgb.tif');

% show origin bayer raw and RGB image
figure(1)
subplot(1,2,1);imshow(I);
title('Origin Bayer Raw Image')

subplot(1,2,2);imshow(J);
title('RGB Image');
```

Bayer Raw 图像和 RGB 色彩插值后的图像比对如图 5.4 所示。

图 5.4 MATLAB 实现的 Bayer Raw 原始图像和转换后的 RGB 图像(见彩插)

5.1.3 Demosaic IP 配置与接口说明

官方文档 pg286-v-demosaic.pdf 对 Sensor Demosaic IP 的使用做了很详细的说明介绍，这里只做简单说明。

1. Demosaic IP 配置

打开 Vivado，如图 5.5 所示，在 IP Catalog 窗口中的 Search 栏中输入 sensor，可以在 Vivado Repository→Video & Image Processing 下看到名为 Sensor Demosaic 的 IP 核，双击它。

图 5.5　IP 分类中的 Sensor Demosaic IP

弹出配置页面如图 5.6 所示，单击 OK 完成配置。

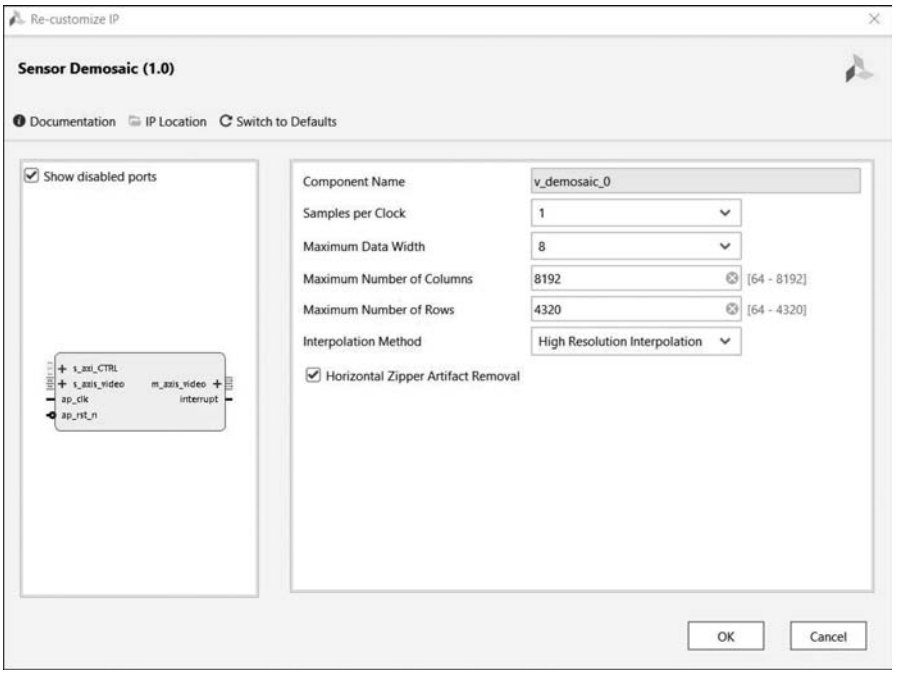


图 5.6　Sensor Demosaic IP 核配置页面

- 设定采样时钟数(Samples per Clock)为 1，数据位宽(Maximum Data Width)为 8(位)。
- 最大列分辨率(Maximum Number of Columns)为 8192(pixel)。
- 最大行分辨率(Maximum Number of Rows)为 4320pixel。
- 插值方式选择高分辨率插值法(High Resolution Interpolation)。

如图 5.7 所示，单击 Generate 按钮生成 IP 文件，可能需要较长时间。

接着还需要使用 AXI4-Lite 总线接口对 IP 核做初始化配置，才能让 IP 核正常工作起来。Sensor Demosaic IP 寄存器功能描述如表 5.1 所示。寄存器 0x04、0x08、0x0c 目前都是保留不用的，无须设置。我们只需要对寄存器 0x00、0x10、0x18、0x28 进行设置即可。

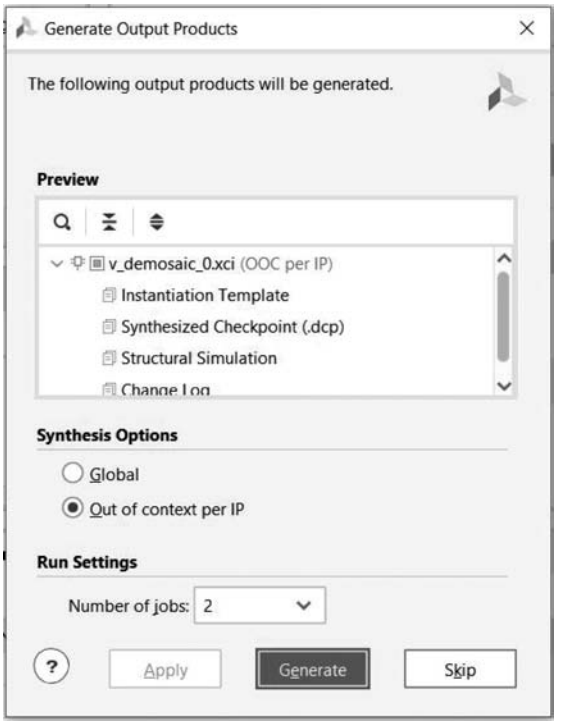


图 5.7　Sensor Demosaic IP 生成页面

0x00 地址是控制(Control)寄存器,若希望 IP 核能够持续工作,位 0(ap_start)和位 7(auto_restart)都需要拉高。0x10 地址是图像的有效宽度(Active Width)寄存器,0x18 地址是图像的有效高度(Active Height)寄存器。例如,我们的图像分辨率为 640×480 像素,则分别设定 0x10 地址为 640,0x18 地址为 480。0x28 地址是拜耳格式(Bayer Phase)寄存器,该寄存器取值范围是 0～3,分别表示输入的 Bayer Raw 图像的格式为:0-RG/GB,1-GR/BG,2-GB/RG,3-BG/GR。在初始化配置时,通常需要先配置好 0x10、0x18、0x28 寄存器的值,最后配置 0x00 寄存器值。

表 5.1　Sensor Demosaic IP 寄存器功能描述

地　址	寄存器名称	读/写	功 能 描 述
0x00	控制寄存器	读/写	位 0:ap_start(读/写) 位 1:ap_done(只读) 位 2:ap_idle(只读) 位 3:ap_ready(只读) 位 6～4:保留不用 位 7:auto_restart(读/写) 位 31～8:保留不用

续表

地　址	寄存器名称	读/写	功 能 描 述
0x04	全局中断使能寄存器	读/写	保留不用
0x08	中断使能寄存器	读/写	保留不用
0x0c	中断状态寄存器	读	保留不用
0x10	有效像素宽度寄存器	读/写	每行图像的有效像素宽度
0x18	有效像素高度寄存器	读/写	每帧图像的有效像素行数
0x28	拜耳格式寄存器	读/写	Bayer Raw 图像的格式

2. Demosaic IP 接口说明

添加好 IP 后，可以在 IP Sources 中找到新产生的 v_demosaic_0 的 IP。如图 5.8 所示，展开 Instantiation Template 后，可以找到 Verilog 的例化模板 v_demosaic_0.veo，双击打开。

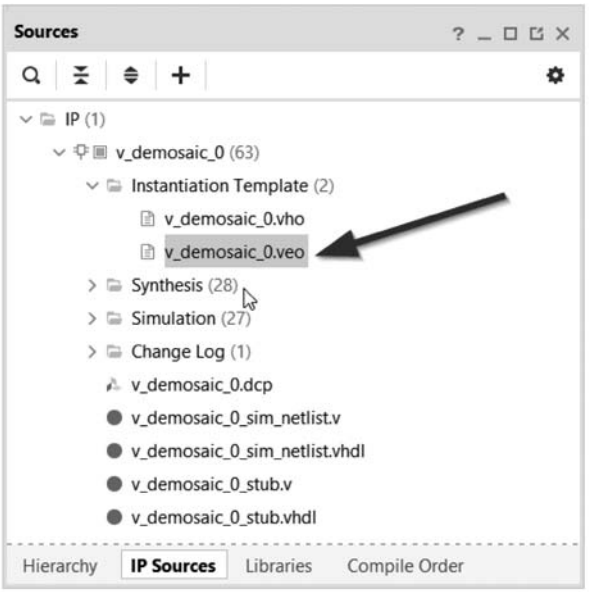


图 5.8　v_demosaic_0 的 IP 文件夹

v_demosaic_0.veo 文件中可以复制下面这段例化模板到设计源码中进行必要的修改。

```
v_demosaic_0 your_instance_name (
.s_axi_CTRL_AWADDR(s_axi_CTRL_AWADDR),          // input wire [5 : 0] s_axi_CTRL_AWADDR
.s_axi_CTRL_AWVALID(s_axi_CTRL_AWVALID),        // input wire s_axi_CTRL_AWVALID
.s_axi_CTRL_AWREADY(s_axi_CTRL_AWREADY),        // output wire s_axi_CTRL_AWREADY
.s_axi_CTRL_WDATA(s_axi_CTRL_WDATA),            // input wire [31 : 0] s_axi_CTRL_WDATA
.s_axi_CTRL_WSTRB(s_axi_CTRL_WSTRB),            // input wire [3 : 0] s_axi_CTRL_WSTRB
.s_axi_CTRL_WVALID(s_axi_CTRL_WVALID),          // input wire s_axi_CTRL_WVALID
```

```
    .s_axi_CTRL_WREADY(s_axi_CTRL_WREADY),          // output wire s_axi_CTRL_WREADY
    .s_axi_CTRL_BRESP(s_axi_CTRL_BRESP),            // output wire [1 : 0] s_axi_CTRL_BRESP
    .s_axi_CTRL_BVALID(s_axi_CTRL_BVALID),          // output wire s_axi_CTRL_BVALID
    .s_axi_CTRL_BREADY(s_axi_CTRL_BREADY),          // input wire s_axi_CTRL_BREADY
    .s_axi_CTRL_ARADDR(s_axi_CTRL_ARADDR),          // input wire [5 : 0] s_axi_CTRL_ARADDR
    .s_axi_CTRL_ARVALID(s_axi_CTRL_ARVALID),        // input wire s_axi_CTRL_ARVALID
    .s_axi_CTRL_ARREADY(s_axi_CTRL_ARREADY),        // output wire s_axi_CTRL_ARREADY
    .s_axi_CTRL_RDATA(s_axi_CTRL_RDATA),            // output wire [31 : 0] s_axi_CTRL_RDATA
    .s_axi_CTRL_RRESP(s_axi_CTRL_RRESP),            // output wire [1 : 0] s_axi_CTRL_RRESP
    .s_axi_CTRL_RVALID(s_axi_CTRL_RVALID),          // output wire s_axi_CTRL_RVALID
    .s_axi_CTRL_RREADY(s_axi_CTRL_RREADY),          // input wire s_axi_CTRL_RREADY
    .ap_clk(ap_clk),                                // input wire ap_clk
    .ap_rst_n(ap_rst_n),                            // input wire ap_rst_n
    .interrupt(interrupt),                          // output wire interrupt
    .s_axis_video_TVALID(s_axis_video_TVALID),      // input wire s_axis_video_TVALID
    .s_axis_video_TREADY(s_axis_video_TREADY),      // output wire s_axis_video_TREADY
    .s_axis_video_TDATA(s_axis_video_TDATA),        // input wire [7 : 0] s_axis_video_TDATA
    .s_axis_video_TKEEP(s_axis_video_TKEEP),        // input wire [0 : 0] s_axis_video_TKEEP
    .s_axis_video_TSTRB(s_axis_video_TSTRB),        // input wire [0 : 0] s_axis_video_TSTRB
    .s_axis_video_TUSER(s_axis_video_TUSER),        // input wire [0 : 0] s_axis_video_TUSER
    .s_axis_video_TLAST(s_axis_video_TLAST),        // input wire [0 : 0] s_axis_video_TLAST
    .s_axis_video_TID(s_axis_video_TID),            // input wire [0 : 0] s_axis_video_TID
    .s_axis_video_TDEST(s_axis_video_TDEST),        // input wire [0 : 0] s_axis_video_TDEST
      .m_axis_video_TVALID(m_axis_video_TVALID),    // output wire m_axis_video_TVALID
      .m_axis_video_TREADY(m_axis_video_TREADY),    // input wire m_axis_video_TREADY
      .m_axis_video_TDATA(m_axis_video_TDATA),      // output wire [23 : 0] m_axis_video_TDATA
      .m_axis_video_TKEEP(m_axis_video_TKEEP),      // output wire [2 : 0] m_axis_video_TKEEP
      .m_axis_video_TSTRB(m_axis_video_TSTRB),      // output wire [2 : 0] m_axis_video_TSTRB
      .m_axis_video_TUSER(m_axis_video_TUSER),      // output wire [0 : 0] m_axis_video_TUSER
      .m_axis_video_TLAST(m_axis_video_TLAST),      // output wire [0 : 0] m_axis_video_TLAST
      .m_axis_video_TID(m_axis_video_TID),          // output wire [0 : 0] m_axis_video_TID
      .m_axis_video_TDEST(m_axis_video_TDEST)       // output wire [0 : 0] m_axis_video_TDEST
);
```

对 IP 的接口简单说明如下。

- ap_clk 为同步时钟信号；ap_rst_n 为低电平有效的复位信号；interrupt 为中断信号，目前保留不用。
- s_axi_CTRL_* 为 AXI4-Lite 总线接口，用于 IP 核的寄存器配置。通过这个接口，可以实现 IP 核的分辨率设定、Bayer Raw 输入模式设定和开关等设定。上电初始，必须通过这个接口进行配置后 IP 核才能工作。
- s_axis_video_* 和 m_axis_video_* 为 AXI4-Stream Video 总线接口。其中 s_axis_video_* 为输入到 IP 核的 Bayer Raw 数据流以及控制信号，m_axis_video_* 为 IP 核输出的经过转换的 RGB 数据流以及控制信号。

3. AXI4-Lite 总线接口简介

AXI4-Lite 总线是 ARM 推出的一个轻量级 AXI 总线接口标准，控制相对简单。针对 Demosaic IP 核的 AXI4-Lite 总线接口信号描述如表 5.2 所示。

表 5.2 AXI4-Lite 总线接口信号描述

信号名称	方向	位宽	功能描述
s_axi_ctrl_awvalid	输入	1 位	写地址通道的写地址有效信号
s_axi_ctrl_awread	输出	1 位	写地址通道的写地址准备好信号，指示 IP 核已经准备好接收写入地址
s_axi_ctrl_awaddr	输入	32 位	写地址通道的写入地址总线
s_axi_ctrl_wvalid	输入	1 位	写数据通道的写数据有效信号
s_axi_ctrl_wready	输出	1 位	写数据通道的写数据准备好信号，指示 IP 核已经准备好接收写入数据
s_axi_ctrl_wdata	输入	32 位	写数据通道的写入数据总线
s_axi_ctrl_bresp	输出	2 位	写响应通道的响应信息，指示写入操作的完成状态
s_axi_ctrl_bvalid	输出	1 位	写响应通道的响应信息有效信号
s_axi_ctrl_bready	输入	1 位	写响应通道的准备好信号，指示 FPGA 逻辑准备好接收 IP 的响应信息
s_axi_ctrl_arvalid	输入	1 位	读地址通道的读地址有效信号
s_axi_ctrl_arready	输出	1 位	读地址通道的读地址准备好信号，指示 IP 核已经准备好接收读地址
s_axi_ctrl_araddr	输入	32 位	读地址通道的读地址总线
s_axi_ctrl_rvalid	输出	1 位	读数据通道的读数据有效信号
s_axi_ctrl_rready	输入	1 位	读数据通道的读数据准备好信号，指示 FPGA 已经准备好接收读出的数据
s_axi_ctrl_rdata	输出	32 位	读数据通道的读数据总线
s_axi_ctrl_rresp	输出	2 位	读数据通道的响应信息，指示读操作的完成状态

关于 AXI 总线接口的控制时序介绍，可以参考第 3 章的内容。

4. AXI4-Stream Video 总线接口简介

AXI4-Stream Video 总线接口信号及功能描述如表 5.3 所示。

表 5.3 AXI4-Stream Video 总线接口信号及功能描述

信号名称	方向	位宽	功能描述
s_axis_video_tdata	输入	8 位	输入视频数据总线
s_axis_video_tvalid	输入	1 位	输入视频数据有效信号
s_axis_video_tready	输出	1 位	主机准备好接收输入视频数据
s_axis_video_tuser	输入	1 位	输入视频帧起始信号
s_axis_video_tlast	输入	1 位	输入视频行结束信号
s_axi_video_tstrb	输入	1 位	输入数据的字节有效信号

续表

信号名称	方向	位宽	功能描述
s_axi_video_tkeep	输入	1位	输入视频流数据的字节有效信号
s_axi_video_tid	输入	1位	输入视频数据流的识别号
s_axi_video_tdest	输入	1位	输入视频数据的路由信息
m_axis_video_tdata	输出	8位	输出视频数据总线
m_axis_video_tvalid	输出	1位	输出视频数据有效
m_axis_video_tready	输入	1位	从机准备好接收输出视频数据
m_axis_video_tuser	输出	1位	输出视频帧起始信号
m_axis_video_tlast	输出	1位	输出视频行结束信号
m_axi_video_tstrb	输出	1位	输出数据的字节有效信号
m_axi_video_tkeep	输出	1位	输出视频流数据的字节有效信号
m_axi_video_tid	输出	1位	输出视频数据流的识别号
m_axi_video_tdest	输出	1位	输出视频数据的路由信息

一个基本的握手传输时序波形如图5.9所示。每个时钟周期(ACLK)的上升沿,主机送出的数据有效信号(VALID,对应接口中的*_tvalid信号)拉高,且从机反馈的准备好信号(READY,对应接口中的*_tready信号)也为高,那么此时的数据总线(DATA,对应接口中的*_tdata信号)上的数据有效且被从机接收。主机发出的帧起始信号(SOF,对应接口中的*_tuser信号)或行结束信号(EOL,对应接口中的*_tlast信号),会一直保持到VALID和READY信号同时拉高,即从机正常接收到该信号。

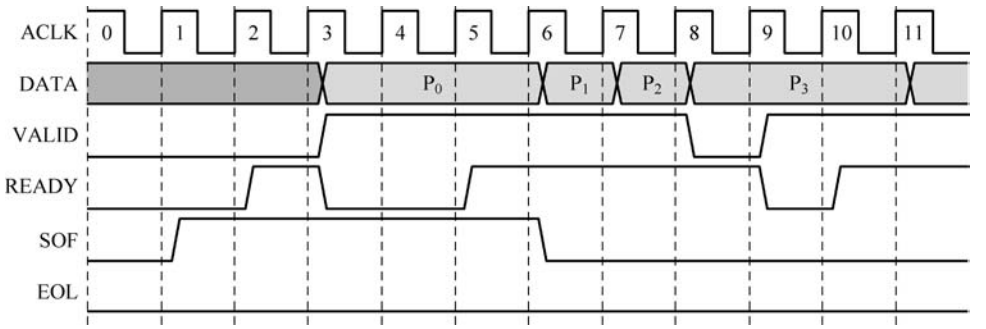


图5.9 基本的握手传输时序波形

如图5.10所示,主机发出的帧起始信号(SOF,对应接口中的*_tuser信号)拉高,则表示一帧图像(或者说一幅图像)的第一个有效数据正在传输。主机发出的行结束信号(EOL,对应接口中的*_tlast信号)拉高,则表示一行图像的最后一个有效数据正在传输。对于一帧图像的传输,只有一个SOF信号,而有多少行的图像数据就有多少个EOL信号。

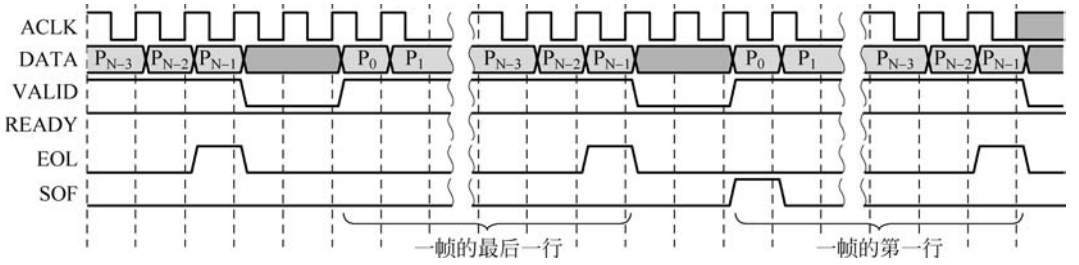


图 5.10　EOL 和 SOF 信号的使用

5.1.4　FPGA 测试脚本解析

at7_img_ex03 工程中的仿真测试脚本 at7_bayer2rgb_sim.v 主要由以下几部分代码实现对 Sensor Demosaic IP 核的仿真验证。

- 接口的声明和参数的定义；
- 例化 Demosaic 的 IP 核 v_demosaic_0；
- $readmemh 语句读取 Bayer Raw 图像；
- Initial 语句产生测试激励；
- 时钟产生；
- $fopen 语句创建结果存储文件 w1_file；
- $fwrite 语句将 RGB 图像写入 w1_file 文件中。

Initial 语句中基本的处理流程如图 5.11 所示。Demosaic IP 核提供了 AXI4-Lite 总线接口，供读写 IP 核内的寄存器，对 IP 核进行初始化配置。Initial 语句中首先对所有的寄存器、接口做初始化赋值；然后依次读取 Demosaic IP 核中的 0x00、0x10、0x18、0x28 寄存器的默认值，再写配置数据到这些寄存器中，最后再读取这些寄存器值以确认正确写入并生效了；接着产生一幅 640×480 像素的 Bayer Raw 图像到 IP 核例化的模块中；最后延时 1ms，结束仿真测试。

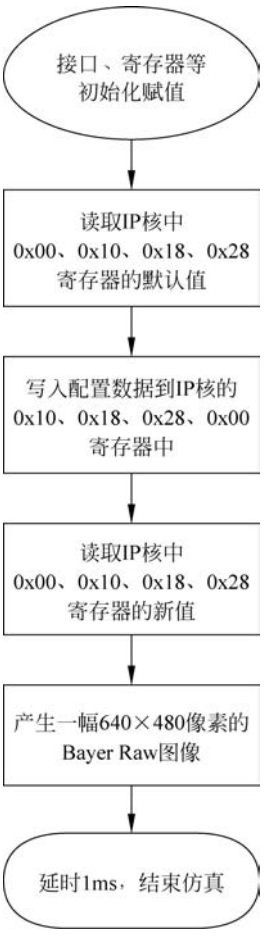


图 5.11　Sensor Demosaic IP 核仿真流程图

5.1.5　FPGA 仿真说明

首先，MATLAB 中运行 at7_img_ex03\matlab 文件夹下的脚本 image_txt_generation.m，将 Bayer Raw 图像 mandi_bayer_raw.tif 生成十六进制数据存储到 image_in_hex.txt 文本中。

复制 image_in_hex.txt 文本，粘贴到 at7_img_ex03\at7.sim 文件夹下。

Vivado 19.1 版本中打开工程 at7_img_ex03，如图 5.12 所示，确认 Sources→Simulation Sources→sim_1 下的 at7_bayer2rgb_sim.v 模块为 top module，选择 Flow Navigator→SIMULATION→Run Simulation 启动仿真。整个过程编译时间较长，需要耐心等待。

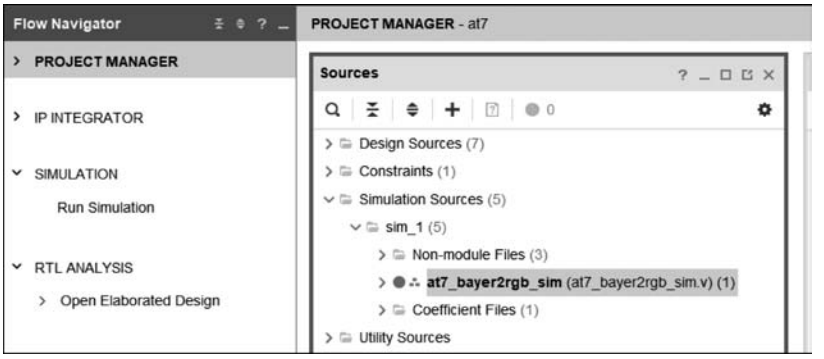


图 5.12　at7_img_ex03 工程界面

运行仿真后，波形如图 5.13 所示。

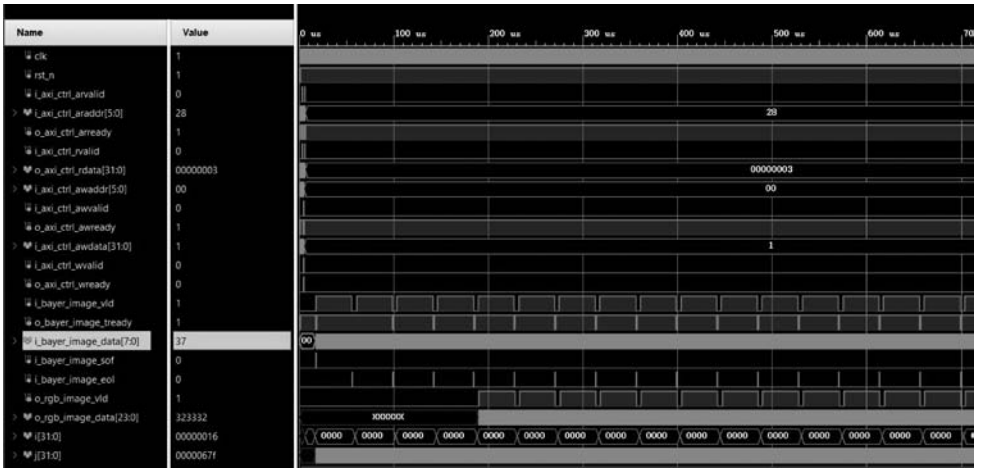


图 5.13　at7_img_ex03 仿真波形

初始化时，AXI4-Lite 读数据的时序波形如图 5.14 所示。

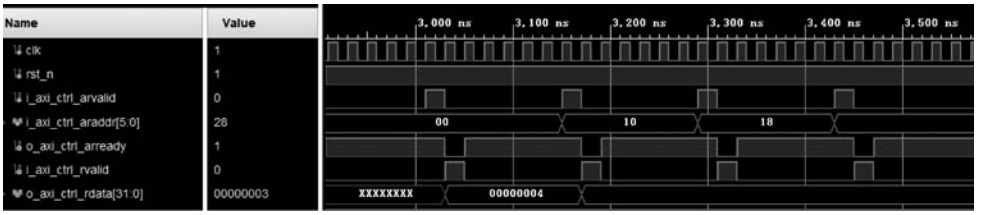


图 5.14　AXI4-Lite 读数据的时序波形

初始化时，AXI4-Lite 写数据的时序波形如图 5.15 所示。

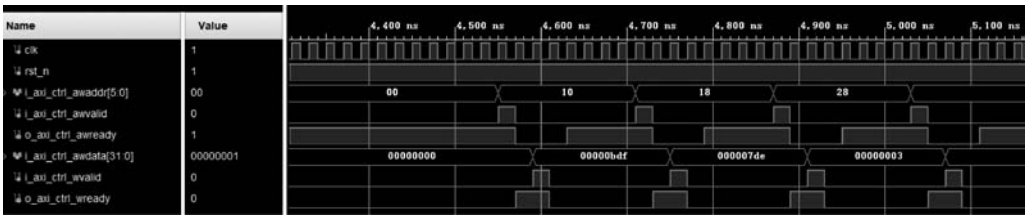


图 5.15　AXI4-Lite 写数据的时序波形

仿真运行结束后，在文件夹 at7_img_ex03\at7.sim\sim_1\behav\xsim 下生成的 FPGA 输出的 RGB 色彩数据存放在文本 FPGA_CFA_Image.txt 中。

运行 matlab 文件夹下的脚本 draw_image_from_FPGA_result.m，可以将 FPGA 转换出的 RGB 图像（文本 FPGA_CFA_Image.txt）和 MATLAB 转换出的 RGB 图像一同绘制出来供比对。

draw_image_from_FPGA_result.m 源码如下：

```
clc;clear all;close all;

IMAGE_WIDTH = 3039;
IMAGE_HIGHT = 2014;
IMAGE_SIZE = 3 * IMAGE_WIDTH * IMAGE_HIGHT;

% load CFA image data from txt
fid1 = fopen('FPGA_CFA_Image.txt', 'r');
img = fscanf(fid1,'%x');
fclose(fid1);

img = uint8(img);

img2 = reshape(img,3,IMAGE_WIDTH,IMAGE_HIGHT);
img3 = permute(img2,[3,2,1]);

I = uint8(img3);

imwrite(I,'FPGA_CFA_Image.tif');
I = imread('FPGA_CFA_Image.tif');

% load origin bayer raw image
J = imread('mandi_rgb.tif');

% show origin bayer raw and RGB image
figure(1)
```

```
subplot(1,2,1);imshow(J);
title('RGB Image with MATLAB')

subplot(1,2,2);imshow(I);
title('RGB Image with FPGA');
```

MATLAB 和 FPGA 分别做插值产生的 RGB 图像比对见图 5.16，肉眼看上去效果基本相当。

图 5.16　MATLAB 和 FPGA 产生的 RGB 图像比对(见彩插)

5.2　色彩滤波矩阵的 FPGA 实现

5.2.1　FPGA 功能概述

5.1 节的实例，我们结合 MATLAB 对 Vivado 2019.1 版本上的 Sensor Demosaic IP 核进行了仿真验证，初步了解这个 Demosaic IP 核的使用。在本实例中，要把这个 IP 核应用到实际工程中进行验证。

如图 5.17 所示，这是整个视频采集和处理系统的功能框图。上电初始，FPGA 通过 SCCB 接口对 OV5640 进行寄存器初始化配置。这些初始化的基本参数，即初始化地址对应的初始化数据都存储在 FPGA 内，以查找表(LUT)的形式逐个写入 OV5640 中。在初始化配置完成后，OV5640 就能够持续输出标准 Bayer Raw 的视频数据流，FPGA 通过对其同步信号，如时钟、行频和场频信号进行检测，从数据总线上实时地采集图像数据。

在 FPGA 内部，采集到的视频数据先通过一个异步 FIFO，将原本时钟域为 25MHz 下同步的数据流转换到 50MHz 下。这个 FIFO 中的数据一旦达到一定数量，会被读取到 2 个不同的后续模块中处理：其中一个模块将 Bayer Raw 格式的图像写入 DDR3 中缓存，LCD 显示驱动模块将读取 DDR3 中 Bayer Raw 图像以灰度形式显示到 VGA 显示器的左侧；另一个模块会在原始图像缓存到 DDR3 之前，把这个 Bayer Raw 格式的图像经过 Sensor Demosaic IP 核(CFA 处理)处理后，转为 RGB 色彩图像，写入 DDR3 另一片存储空间中，LCD 显示驱动模块将会读取 DDR3 中的这部分 RGB 图像显示到 VGA 显示器的右侧。

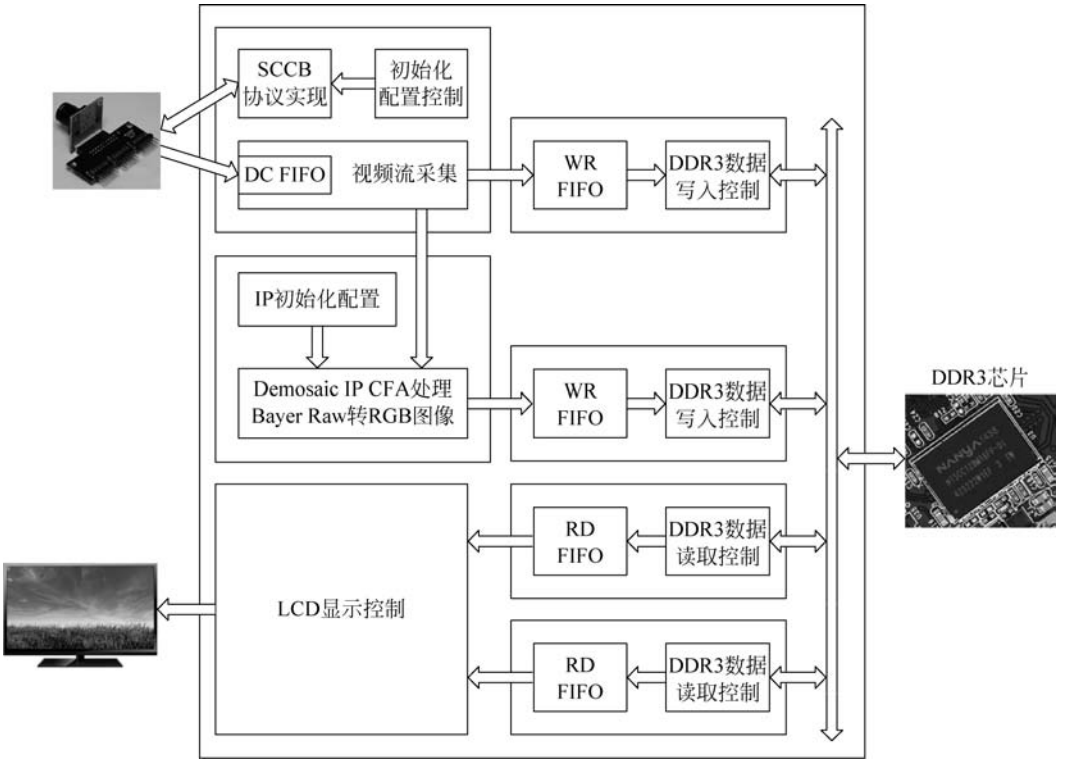


图 5.17　色彩滤波矩阵功能框图

5.2.2　FPGA 设计说明

FPGA 工程的层次结构如图 5.18 所示。

图 5.18　at7_img_ex04 工程源码层次

at7_img_ex04 工程模块的功能描述如表 5.4 所示。

1. Demosaic IP 配置

打开 Vivado，在 IP Catalog 窗口 Search 中输入 sensor，可以在 Vivado Repository→Video & Image Processing 下看到名为 Sensor Demosaic 的 IP 核，双击它。

弹出 Demosaic IP 核配置页面如图 5.19 所示，单击 OK 完成配置。

表 5.4　at7_img_ex04 工程模块功能描述

模块名称	功能描述
clk_wiz_0	该模块是 PLL IP 核的例化模块，该 PLL 用于产生系统中所需要的不同频率时钟信号
mig_7series_0	该模块是 DDR3 控制器 IP 核的例化模块。FPGA 内部逻辑读写访问 DDR3 都是通过该模块实现，该模块包含与 DDR3 芯片连接的物理层接口
Image_controller	该模块及其子模块实现 SCCB 接口对 OV5640 的初始化、OV5640 输出图像的采集控制等。这个模块内部例化了两个子模块：I2C_OV5640_Init_RGB565 模块实现 SCCB 接口通信协议和初始化配置，其下例化的 I2C_Controller 模块实现 SCCB 协议，I2C_OV5640_RGB565_Config 模块用于产生图像传感器的初始配置数据，SCCB 接口的初始化配置控制实现则在 I2C_OV5640_Init_RGB565 模块中实现；image_capture 模块实现图像采集功能
ddr3_cache	该模块主要用于缓存读或写 DDR3 的数据，其下例化了两个 FIFO。该模块衔接 FPGA 内部逻辑与 DDR3 IP 核（mig_7series_0.v 模块）之间的数据交互
bayer2rgb	该模块实现 Bayer Raw 图像转换为 RGB888 图像的处理。该模块例化了 Sensor Demosaic IP 核，通过 AXI4-Lite 接口对 IP 核初始化，通过 AXI4-Stream Video 接口实现 FPGA 逻辑与 IP 核之间的图像传输
lcd_driver	该模块驱动 VGA 显示器，同时产生读取 DDR3 中图像数据的控制逻辑
led_controller	该模块控制 LED 闪烁，指示工作状态

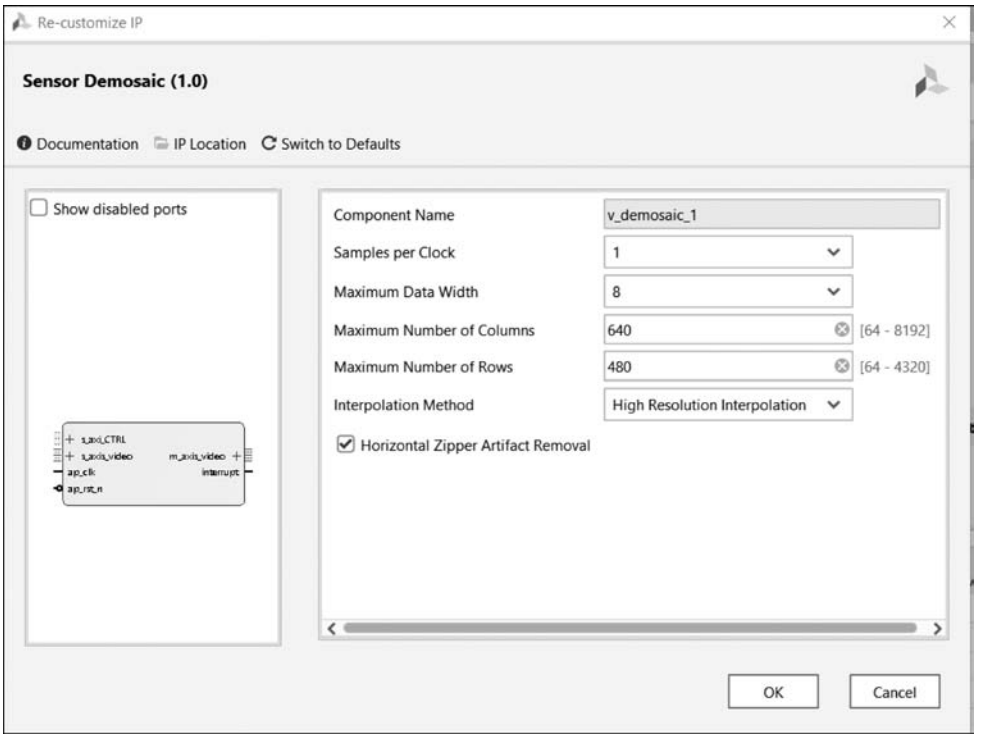


图 5.19　Demosaic IP 核配置页面

- 设定采样时钟数(Samples per Clock)为1,数据位宽(Maximum Data Width)为8(位)。
- 最大列分辨率(Maximum Number of Columns)为640(单位：像素)。
- 最大行分辨率(Maximum Number of Rows)为480(单位：行)。
- 插值方式选择高分辨率插值法(High Resolution Interpolation)。

2. bayer2rgb.v模块代码解析

bayer2rgb.v模块实现Sensor Demosaic IP核的例化和初始化操作。该模块功能框图如图5.20所示。上电后定时计数逻辑工作,产生4组(8个)定时脉冲,使用AXI4-Lite接口对Demosaic IP核进行初始化操作,对其4个主要的寄存器做初始化赋值。输入的Bayer Raw视频流通过Sensor Demosaic IP核处理后,输出RGB格式的视频流。同时,使用Bayer Raw图像帧结束信号i_bayer_image_eof作为复位计数器的启动脉冲,以产生RGB图像的复位信号o_rgb_image_rst。

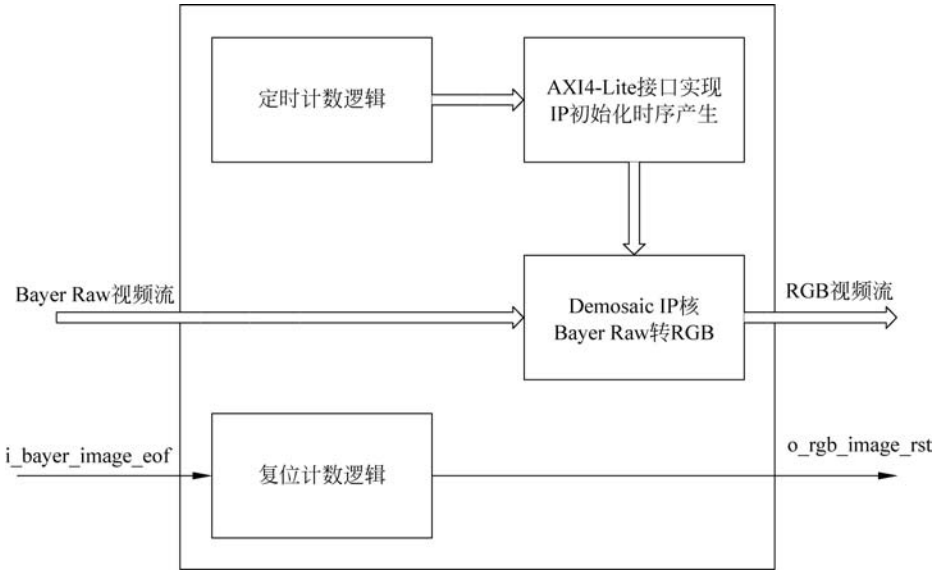


图5.20　bayer2rgb.v模块功能框图

该模块的接口定义如下。i_bayer_image_*接口来自OV5640摄像头采集到的Bayer Raw格式图像。o_rgb_image_*接口为经过Demosaic IP核处理后转成Color RGB的彩色图像。

```
`timescale 1ns/1ps
module bayer2rgb(
        input clk,
        input rst_n,
        //input Image Data Flow
        input i_bayer_image_vld,
        output o_bayer_image_tready,
```

```
        input[7:0] i_bayer_image_data,
        input i_bayer_image_sof,
        input i_bayer_image_eof,
        input i_bayer_image_eol,
        //output Image Data Flow
        output reg o_rgb_image_rst,
        output o_rgb_image_vld,
        output[23:0] o_rgb_image_data
    );
```

bayer2rgb.v 模块接口定义如表 5.5 所示。

表 5.5 bayer2rgb.v 模块接口定义

信 号 名	方向	描 述
clk	Input	时钟信号
rst_n	Input	复位信号,低电平有效
i_bayer_image_vld	Input	Bayer Raw 图像数据有效信号
o_bayer_image_tready	Output	Demosaic IP 核传输准备好
i_bayer_image_data[7:0]	Input	Bayer Raw 图像数据
i_bayer_image_sof	Input	Bayer Raw 图像帧起始信号
i_bayer_image_eof	Input	Bayer Raw 图像帧结束信号
i_bayer_image_eol	Input	Bayer Raw 图像行结束信号
o_rgb_image_rst	Output	RGB 图像复位信号,高电平有效
o_rgb_image_vld	Output	RGB 图像数据有效信号
o_rgb_image_data[23:0]	Output	RGB 图像数据,位 23～16 为 R 数据,位 15～8 为 B 数据,位 7～0 为 G 数据

bayer2rgb.v 模块内部寄存器等接口声明如下。

```
reg[15:0] cnt;
reg[5:0] i_axi_ctrl_awaddr;
reg i_axi_ctrl_awvalid;
wire o_axi_ctrl_awready;
reg[31:0] i_axi_ctrl_awdata;
reg i_axi_ctrl_wvalid;
wire o_axi_ctrl_wready;
```

分辨率参数定义如下。

```
parameter IMAGE_WIDTH = 32'd640;
parameter IMAGE_HIGHT = 32'd480;
```

Demosaic IP 核的例化如下,具体接口的定义可以参考 at7_img_ex03 实例(见 5.1.3 节)的介绍。

```
////////////////////////////////////////////////////
//demosaic IP 例化

v_demosaic_1      uut_v_demosaic_1 (
  .s_axi_CTRL_AWADDR  (i_axi_ctrl_awaddr),            // input wire [5 : 0] s_axi_CTRL_AWADDR
  .s_axi_CTRL_AWVALID (i_axi_ctrl_awvalid),           // input wire s_axi_CTRL_AWVALID
  .s_axi_CTRL_AWREADY (o_axi_ctrl_awready),           // output wire s_axi_CTRL_AWREADY
  .s_axi_CTRL_WDATA   (i_axi_ctrl_awdata),            // input wire [31 : 0] s_axi_CTRL_WDATA
  .s_axi_CTRL_WSTRB   (4'hf/ * s_axi_CTRL_WSTRB * /),  // input wire [3 : 0] s_axi_CTRL_WSTRB
  .s_axi_CTRL_WVALID  (i_axi_ctrl_wvalid),            // input wire s_axi_CTRL_WVALID
  .s_axi_CTRL_WREADY  (o_axi_ctrl_wready),            // output wire s_axi_CTRL_WREADY
  .s_axi_CTRL_BRESP   (/ * s_axi_CTRL_BRESP * /),     // output wire [1 : 0] s_axi_CTRL_BRESP
  .s_axi_CTRL_BVALID  (/ * s_axi_CTRL_BVALID * /),    // output wire s_axi_CTRL_BVALID
  .s_axi_CTRL_BREADY  (1'b1/ * s_axi_CTRL_BREADY * /),  // input wire s_axi_CTRL_BREADY
  .s_axi_CTRL_ARADDR  (6'd0),                         // input wire [5 : 0] s_axi_CTRL_ARADDR
  .s_axi_CTRL_ARVALID (1'b0),                         // input wire s_axi_CTRL_ARVALID
  .s_axi_CTRL_ARREADY (),                             // output wire s_axi_CTRL_ARREADY
  .s_axi_CTRL_RDATA   (),                             // output wire [31 : 0] s_axi_CTRL_RDATA
  .s_axi_CTRL_RRESP   (/ * s_axi_CTRL_RRESP * /),     // output wire [1 : 0] s_axi_CTRL_RRESP
  .s_axi_CTRL_RVALID  (),                             // output wire s_axi_CTRL_RVALID
  .s_axi_CTRL_RREADY  (1'b1/ * s_axi_CTRL_RREADY * /),  // input wire s_axi_CTRL_RREADY
  .ap_clk             (clk),                          // input wire ap_clk
  .ap_rst_n           (rst_n),                        // input wire ap_rst_n
  .interrupt          (/ * interrupt * /),            // output wire interrupt
  .s_axis_video_TVALID(i_bayer_image_vld),            // input wire s_axis_video_TVALID
  .s_axis_video_TREADY(o_bayer_image_tready),         // output wire s_axis_video_TREADY
  .s_axis_video_TDATA (i_bayer_image_data),           // input wire [7 : 0] s_axis_video_TDATA
  .s_axis_video_TKEEP (1'b1/ * s_axis_video_TKEEP * /),  // input wire [0 : 0] s_axis_video_TKEEP
  .s_axis_video_TSTRB (1'b1/ * s_axis_video_TSTRB * /),  // input wire [0 : 0] s_axis_video_TSTRB
  .s_axis_video_TUSER (i_bayer_image_sof),            // input wire [0 : 0] s_axis_video_TUSER
  .s_axis_video_TLAST (i_bayer_image_eol),            // input wire [0 : 0] s_axis_video_TLAST
  .s_axis_video_TID   (1'b1/ * s_axis_video_TID * /),  // input wire [0 : 0] s_axis_video_TID
  .s_axis_video_TDEST (1'b1/ * s_axis_video_TDEST * /),  // input wire [0 : 0] s_axis_video_TDEST
  .m_axis_video_TVALID(o_rgb_image_vld),              // output wire m_axis_video_TVALID
  .m_axis_video_TREADY(1'b1/ * m_axis_video_TREADY * /), // input wire m_axis_video_TREADY
  .m_axis_video_TDATA (o_rgb_image_data),             //output wire [23 : 0] m_axis_video_TDATA
  .m_axis_video_TKEEP (/ * m_axis_video_TKEEP * /),   // output wire [2 : 0] m_axis_video_TKEEP
  .m_axis_video_TSTRB (/ * m_axis_video_TSTRB * /),   // output wire [2 : 0] m_axis_video_TSTRB
  .m_axis_video_TUSER (/ * m_axis_video_TUSER * /),   // output wire [0 : 0] m_axis_video_TUSER
  .m_axis_video_TLAST (/ * m_axis_video_TLAST * /),   // output wire [0 : 0] m_axis_video_TLAST
  .m_axis_video_TID   (/ * m_axis_video_TID * /),     // output wire [0 : 0] m_axis_video_TID
  .m_axis_video_TDEST (/ * m_axis_video_TDEST * /)    // output wire [0 : 0] m_axis_video_TDEST
);
```

以下初始化计数与时序控制逻辑实现 Sensor Demosaic IP 核的初始化配置，将其设定为 640×480 分辨率，输入 Bayer Raw 格式为 GR/BG，启动运行。这部分代码的基本功能如

图 5.21 所示，类似一个软件顺序执行的初始化控制。

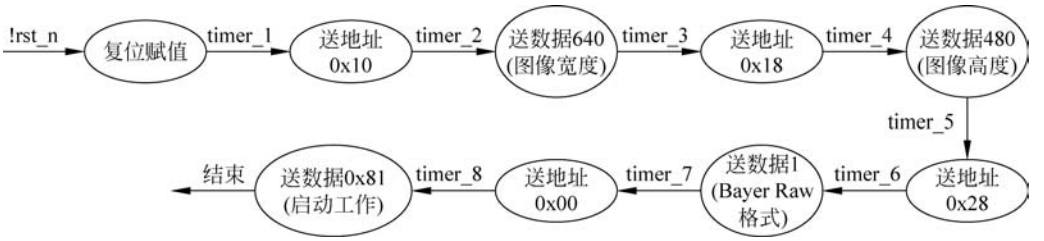


图 5.21 Sensor Demosaic IP 核的初始化配置流程

```
////////////////////////////////////////////////////
//demosaic IP 初始化

always @(posedge clk)
    if(!rst_n) cnt <= 16'd0;
    else if(cnt < 16'hffff) cnt <= cnt + 1'b1;

wire timer_1 = (cnt == 16'h8000);
wire timer_2 = (cnt == 16'h8004);

wire timer_3 = (cnt == 16'h9000);
wire timer_4 = (cnt == 16'h9004);

wire timer_5 = (cnt == 16'ha000);
wire timer_6 = (cnt == 16'ha004);

wire timer_7 = (cnt == 16'hb000);
wire timer_8 = (cnt == 16'hb004);

always @(posedge clk)
    if(!rst_n) begin
        i_axi_ctrl_awaddr <= 6'd0;
        i_axi_ctrl_awvalid <= 1'b0;
        i_axi_ctrl_awdata <= 32'd0;
        i_axi_ctrl_wvalid <= 1'b0;
    end
    //register 0x0010 (active width) = IMAGE_WIDTH
    else if(timer_1) begin
        i_axi_ctrl_awaddr <= 6'h10;
        i_axi_ctrl_awvalid <= 1'b1;
        i_axi_ctrl_awdata <= 32'd0;
        i_axi_ctrl_wvalid <= 1'b0;
    end
    else if(timer_2) begin
```

```
        i_axi_ctrl_awaddr <= 6'd0;
        i_axi_ctrl_awvalid <= 1'b0;
        i_axi_ctrl_awdata <= IMAGE_WIDTH;
        i_axi_ctrl_wvalid <= 1'b1;
    end
    //register 0x0018 (active height) = IMAGE_HIGHT
    else if(timer_3) begin
        i_axi_ctrl_awaddr <= 6'h18;
        i_axi_ctrl_awvalid <= 1'b1;
        i_axi_ctrl_awdata <= 32'd0;
        i_axi_ctrl_wvalid <= 1'b0;
    end
    else if(timer_4) begin
        i_axi_ctrl_awaddr <= 6'd0;
        i_axi_ctrl_awvalid <= 1'b0;
        i_axi_ctrl_awdata <= IMAGE_HIGHT;
        i_axi_ctrl_wvalid <= 1'b1;
    end
    //register 0x0028 (bayer phase) = 0 - RG/GB, 1 - GR/BG, 2 - GB/RG, 3 - BG/GR
    else if(timer_5) begin
        i_axi_ctrl_awaddr <= 6'h28;
        i_axi_ctrl_awvalid <= 1'b1;
        i_axi_ctrl_awdata <= 32'd0;
        i_axi_ctrl_wvalid <= 1'b0;
    end
    else if(timer_6) begin
        i_axi_ctrl_awaddr <= 6'd0;
        i_axi_ctrl_awvalid <= 1'b0;
        i_axi_ctrl_awdata <= 32'd1; //GR/BG
        //i_axi_ctrl_awdata <= 32'd2; //GB/RG
        //i_axi_ctrl_awdata <= 32'd3; // BG/GR
        i_axi_ctrl_wvalid <= 1'b1;
    end
    //register 0 (ctrl): bit7 = 1 (auto_restart)
    else if(timer_7) begin
        i_axi_ctrl_awaddr <= 6'd0;
        i_axi_ctrl_awvalid <= 1'b1;
        i_axi_ctrl_awdata <= 32'd0;
        i_axi_ctrl_wvalid <= 1'b0;
    end
    else if(timer_8) begin
        i_axi_ctrl_awaddr <= 6'd0;
        i_axi_ctrl_awvalid <= 1'b0;
        i_axi_ctrl_awdata <= 32'h0000_0081;
        i_axi_ctrl_wvalid <= 1'b1;
    end
```

```
        else begin
            i_axi_ctrl_awaddr <= 6'd0;
            i_axi_ctrl_awvalid <= 1'b0;
            i_axi_ctrl_awdata <= 32'd0;
            i_axi_ctrl_wvalid <= 1'b0;
        end

    ////////////////////////////////////////////////////
    //延时计数器,产生复位信号
    reg[11:0] dly;

    always @(posedge clk)
        if(!rst_n) dly <= 12'd0;
        else if(i_bayer_image_eof) dly <= 12'd1;
        else if(dly != 12'd0) dly <= dly + 1'b1;
        else dly <= 12'd0;

    always @(posedge clk)
        if(!rst_n) o_rgb_image_rst <= 1'b0;
        else if((dly >= 12'd3200) && (dly <= 12'd3300)) o_rgb_image_rst <= 1'b1;
        else o_rgb_image_rst <= 1'b0;

    endmodule
```

5.2.3 FPGA 板级调试

连接好 OV5640 摄像头模块、VGA 模块和 FPGA 开发板,同时连接好 FPGA 的下载器并给板子供电。

使用 Vivado 2019.1 打开工程 at7_img_ex04,将 at7_img_ex04\at7.runs\impl_1 文件夹下的 at7.bit 文件烧录到板子上。如图 5.22 所示,可以看到 VGA 显示器同时显示左右两个图像,左侧图像为原始 Bayer Raw 灰度图像(看上去会有很明显的点阵式的感觉),右侧显示转换后的 RGB 彩色图像(比较接近真实的彩色图像)。

图 5.22 OV5640 色彩滤波矩阵实现效果图(见彩插)

5.3 伽马校正的 FPGA 实现

5.3.1 伽马(Gamma)介绍

1. Gamma 的由来

Gamma 是一个出现较早的技术。话说由于早期 CRT 显示器输入单位电压并不会产生等量的亮度(所以是非线性),为了正确地显示画面颜色亮度,刻意制定一个曲线关系(x 轴为输入,y 轴为输出),让最终输出的影像为线性颜色亮度的影像。即使现代能够制造出线性反应的液晶屏,这种现象仍然深深地影响影像处理,不管是后制、合成、调色或是 3D 算图渲染都离不开 Gamma 技术。

人类接收外界信息,视觉占了所有感官的一半以上。无论是输入还是输出或是介于其中,Gamma 已经深入人们的生活中,过去使用 CRT 显示器,现在使用 LCD 液晶屏,它们有何差异?为什么 Mac 要使用 Gamma 1.8,而 PC 要使用 Gamma 2.2 呢?为什么做设计的人比较偏好苹果系列计算机?

Gamma 是一个描述阶调(Tone)特性的对数。字典里定义 Gamma 为一个数,指示影像明暗的对比等级,它可以是一条直线。还可以把 Gamma 描述为非线性指数函数,这个函数是以两个变数来定义:$f(x)=x^y$。Gamma 描述一条线性曲线或是在对数尺度的一条直线。

简单来说,Gamma 描述了相机或显示屏的非线性(Nonlinear)特性。当一个相机接收到两倍的光强时,相机并不会把这个光转换成两倍的 RGB 值。其中有感光本身的问题,也有显示(主要是早期的 CRT 屏幕)的问题,它们都可能使得像素的光亮强度与输入的电压强度并不是呈现线性关系。

现代液晶显示器(Thin Film Transistor Liquid Crystal Display,TFT-LCD)本身虽然没有先天的 Gamma 问题,但是为了要适应传统的工作流程,TFT-LCD 液晶屏会刻意模拟出 Gamma 的效果。

2. 现实世界中的 Gamma

所有的屏幕都有非线性的输入/输出反应,这是有意为之。

大多数的 2D 软件都会以线性的颜色模型来处理,所以它假定,255 数值的亮度是 128 数值的两倍。但由于显示器对于信号的输入/输出是非线性的,所以产生的亮度会是不正确的。事实上,大多数屏幕(Gamma=2.2),如果想要显示出数值为 255 亮度的 50%亮度,那就必须要输入$(0.5^{1/2.2})\times 255=186$ 的数值。如果不考虑 Gamma 的问题,输入 128 数值,就只会产生约$(128/255)^{2.2}=22\%$的亮度。

数码相机基本上具有线性的输入/输出效果,但因为通常我们会在计算机屏幕上查看拍摄出来的照片,所以数码相机会故意在照片里面嵌入 Gamma。JPG 格式是带有 Gamma 的,但是 Raw 格式是线性的,当把 Raw 格式转成 JPG 格式时就会产生非线性的照片了。因

此，如果用 2D 软件打开拍出的 JPG 图片时，必须要把 Gamma 补偿回来（去 Gamma）。

如果图片是在 2D 软件产生的（基本上这张图片是线性的），当把这张照片显示在带 Gamma 的屏幕上，也是要做 Gamma 补偿的。

Gamma 不是缺陷，它是一个功能，因为人的眼睛对光线的亮度具有非线性的感光反应。如果每个颜色只有 8 位来记录，如何利用这 8 位正确地重现人眼的感光效果很重要，它必须是非线性的编码方式。即使是新一点的屏幕仍然有 Gamma：通常显示卡会用 8 位来处理每种颜色避免色带问题，这 8 位必须每个强度看起来间距是相等的。

现今大多数计算机屏幕都以 sRGB（standard RGB）的标准来显示，也就是 Gamma 2.2。大多数的数码相机也以 sRGB 存储相片，如果是扫描进来的图或是合成图像就不会带有 Gamma 2.2。但几乎所有的浮点纪录 HDR 资料都是线性的，即 Gamma 为 1.0。

对于图片而言，Gamma 代表了强度是如何被记录的。换句话说，图片的 Gamma 是为了要让图片在屏幕上能正确地显示出来。

有些图片会带有 Gamma 标签，但这是不可靠的，因为很多绘图软件会忽略这个标签。因此，要准确知道图的 Gamma 数值并不容易。如果显示器有 Gamma 2.2 而显示的图片看起来很正常，那该图片可能本身就带有 Gamma 2.2。

3. Gamma 校正

Gamma 校正的思路是在最终的颜色输出上应用显示器 Gamma 的倒数。如图 5.23 所示，那条向上的虚曲线，它是显示器 Gamma 曲线的翻转曲线。在颜色送到显示器的时候把每个颜色输出都加上这个翻转的 Gamma 曲线，这样应用了显示器的 Gamma 以后最终的颜色将会变为线性的。我们所得到的中间色调就会更亮，所以虽然显示器使它们变暗，但是我们又将其平衡回来了。

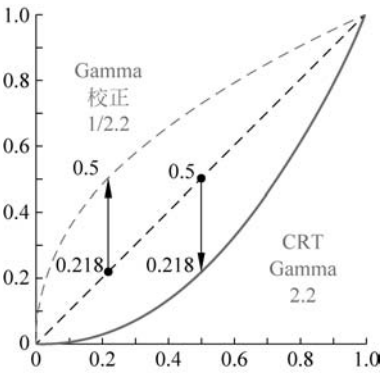


图 5.23　Gamma 曲线图（见彩插）

来看另一个例子。取值为（0.5，0.0，0.0）的暗红色。在将颜色送到显示器之前，先对颜色应用 Gamma 校正曲线。线性的颜色显示在显示器上相当于降低了 2.2 次幂的亮度，所以倒数就是 1/2.2 次幂。Gamma 校正后的暗红色就会成为 $(0.5,0.0,0.0)^{1/2.2}=(0.5,0.0,0.0)^{0.45}=(0.73,0.0,0.0)$。校正后的颜色接着被发送给显示器，最终显示出来的颜色是 $(0.73,0.0,0.0)^{2.2}=(0.5,0.0,0.0)$。可以发现使用了 Gamma 校正，显示器最终会显示出我们在应用中设置的那种线性的颜色。

2.2 通常是大多数显示设备的平均 Gamma 值。基于 Gamma 2.2 的颜色空间叫作 sRGB 颜色空间。每个显示器的 Gamma 曲线都有所不同，但是 Gamma 2.2 在大多数显示器上表现都不错。出于这个原因，游戏经常都会为玩家提供改变游戏 Gamma 设置的选项，以适应每个显示器。

4. Gamma 计算公式

过去的 CRT 显示器是使用电子显像管，通过控制电流大小来调整显示屏幕上的亮度。然而亮度和电流之间的关系并非是线性的，也就是说电流强度变为 2 倍，显示的亮度并非是 2 倍，而是由式(5.1)决定：

$$\mathrm{Vout} = \mathrm{Vin}^{\mathrm{Gamma}} \tag{5.1}$$

其中，Gamma 为 CRT 显示器的伽马值。

然而对于现实中的大部分摄像机或成像设备而言，输入能量和记录在图片文件中的颜色亮度之间的关系却是线性的，这就导致显示器显示的图像与摄像设备捕捉的实际图像不一致，为了校正这个差异，摄像机在保存图像时会自动对数据进行 Gammer 校正，公式如下：

$$\mathrm{Vout} = \mathrm{Vin}^{1/\mathrm{Gamma}} \tag{5.2}$$

Gamma 依旧为显示器的伽马值。这样，当显示器显示图像时，式(5.2)的输出作为式(5.1)的输入，最后抵消了显示器的 Gamma 值造成的误差。

$$\mathrm{Vdisplay} = (\mathrm{Vcamera}^{1/\mathrm{Gamma}})^{\mathrm{Gamma}} = \mathrm{Vcamera}$$

5.3.2 MATLAB 生成 Gamma 校正的 LUT

at7_img_ex05/matlab 文件夹下的 MATLAB 脚本文件 gammaCorrection.m 可用于生成 0.45(1/2.2)的 Gamma 校正数据，这组 256 个点的数据以 Vivado 中可用的 ROM 初始化文件(gamma_lut.coe)形式保存下来。

gammaCorrection.m 的代码如下。

```
for r = 0:1:255
  I(r+1) = round(255*exp((log(r/255))*0.45),0);
  %I(r+1) = round(255*((r/255)^0.45),0);
  %J(r+1) = round(255*exp((log10(r/255))*0.45),0);
end

%% output peak
fid20 = fopen('gamma_lut.coe', 'wt');
fprintf(fid20, 'memory_initialization_radix = 16;\n');
fprintf(fid20, 'memory_initialization_vector = \n');
for r = 1:1:255
    fprintf(fid20, '%.2x,\n', I(r));
end
fprintf(fid20, '%.2x;\n', I(256));
fid20 = fclose(fid20);
```

0.45 的 Gamma 校正数据绘制的曲线如图 5.24 所示。

5.3.3 FPGA 功能概述

如图 5.25 所示，这是整个视频采集和处理系统的功能框图。上电初始，FPGA 需要通

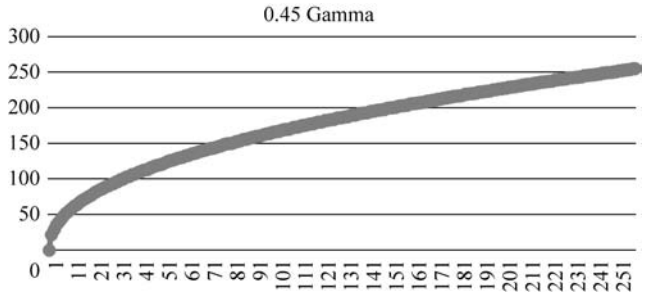


图 5.24　0.45 的 Gamma 校正曲线

过 SCCB 接口对 CMOS Sensor 进行寄存器初始化配置。这些初始化的基本参数，即初始化地址对应的初始化数据都存储在 FPGA 内。在初始化配置完成后，CMOS Sensor 就能够持续输出标准 Bayer Raw 的视频数据流，FPGA 通过对其同步信号，如时钟、行频和场频进行检测，从而从数据总线上实时地采集图像数据。

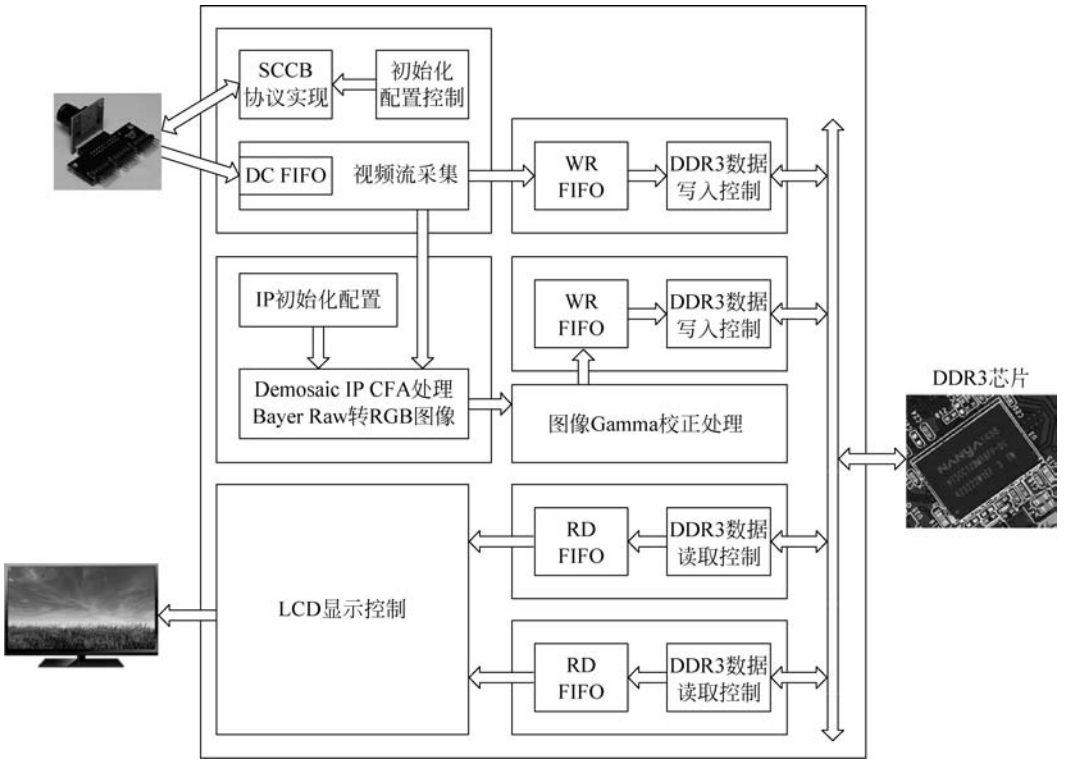


图 5.25　OV5640 伽马校正功能框图

在 FPGA 内部，采集到的视频数据先通过一个 FIFO，将原本时钟域为 25MHz 下同步的数据流转换到 50MHz 下。接着将这个数据再送入写 DDR3 缓存的异步 FIFO 中，这个

FIFO 中的数据一旦达到一定数量，会被读取并进行 Bayer Raw 转 Color RGB 的处理，随后图像送往两个不同的模块：一个模块将图像直接写入 DDR3 中，最终被读取并显示到 VGA 显示器的左侧；另一个模块会在 Color RGB 图像缓存到 DDR3 之前，对图像进行 Gamma 校正处理，然后再写入 DDR3 另一片存储空间中，最终被读取的 Gamma 校正后图像显示到 VGA 显示器的右侧。

5.3.4　FPGA 设计说明

at7_img_ex05 工程源码的层次结构如图 5.26 所示。

图 5.26　at7_img_ex05 工程源码层次结构

at7_img_ex05 工程模块功能描述如表 5.6 所示。

表 5.6　at7_img_ex05 工程模块功能描述

模块名称	功能描述
clk_wiz_0	该模块是 PLL IP 核的例化模块，该 PLL 用于产生系统中所需要的不同频率时钟信号
mig_7series_0	该模块是 DDR3 控制器 IP 核的例化模块。FPGA 内部逻辑读写访问 DDR3 都是通过该模块实现，该模块包含与 DDR3 芯片连接的物理层接口
image_controller	该模块及其子模块实现 SCCB 接口对 OV5640 的初始化、OV5640 输出图像的采集控制等。这个模块内部例化了两个子模块，I2C_OV5640_Init_RGB565 模块实现 SCCB 接口通信协议和初始化配置，其下例化的 I2C_Controller 模块实现 SCCB 协议，I2C_OV5640_RGB565_Config 模块用于产生图像传感器的初始配置数据，SCCB 接口的初始化配置控制实现则在 I2C_OV5640_Init_RGB565 模块中实现；image_capture 模块实现图像采集功能
ddr3_cache	该模块主要用于缓存读或写 DDR3 的数据，其下例化了两个 FIFO。该模块连接 FPGA 内部逻辑与 DDR3 IP 核（mig_7series_0 模块）之间的数据交互
bayer2rgb	该模块实现 Bayer Raw 图像转换为 RGB888 图像的处理。该模块例化了 Demosaic IP 核，通过 AXI4-Lite 接口对 IP 核初始化，通过 AXI4-Stream Video 接口实现 FPGA 逻辑与 IP 核之间的图像传输
gamma_correction	该模块使用一个预定义好的 ROM 存储 Gamma 校正的查找表（LUT），用输入的图像数据作为 ROM 的地址进行查表，输出的新数据便是 Gamma 校正后的图像
lcd_driver	该模块驱动 LCD，同时产生读取 DDR3 中图像数据的控制逻辑
led_controller	该模块控制 LED 闪烁，指示工作状态

1. Gamma 校正模块

Gamma 校正在 gamma_correction.v 模块中实现，功能实现其实很简单，只要使用一个预定义好的 ROM 存储 Gamma 校正的查找表，用输入的图像数据作为 ROM 的地址进行查表，输出的新数据便是 Gamma 校正后的图像。

该模块的层次结构如图 5.27 所示。

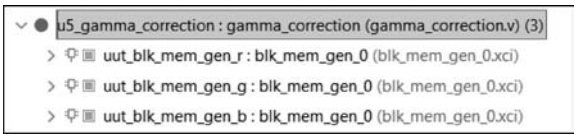


图 5.27　gamma_correction.v 模块层次结构

该模块的内部功能框图如图 5.28 所示。R、G、B 这 3 个通道分别需要一个 ROM 存储查找表，它们的查找表内容是一致的，因此只需要配置一个 ROM，加载查找表并做 3 次例化即可。输入 R、G、B 的数据作为 ROM 的地址，获取相应地址的输出即 Gamma 校正后的结果。

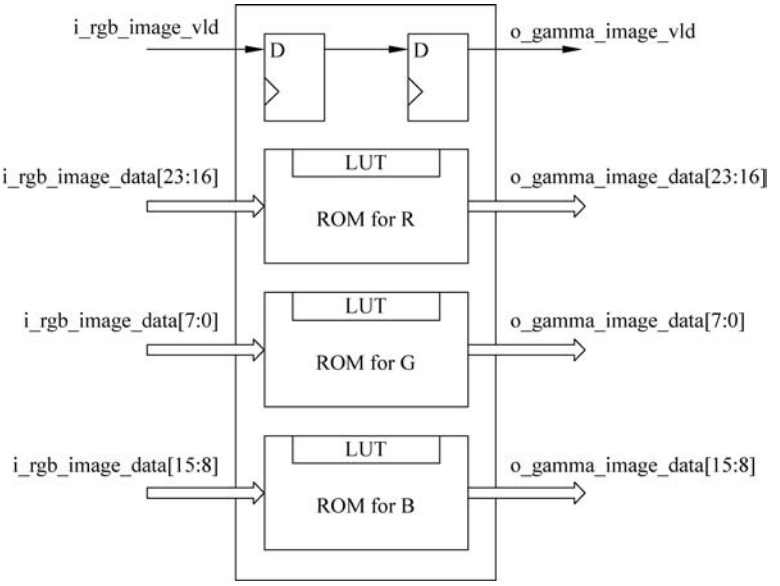


图 5.28　gamma_correction.v 模块内部功能框图

2. ROM 添加与配置

gamma_correction.v 模块中配置了 3 个 ROM(R、G、B 通道各 1 个)，这 3 个 ROM 的配置完全一致。

在 IP Catalog 中找到 Block Memory Generator 的 IP，如图 5.29 所示，双击添加。

弹出 Basic 配置页面，如图 5.30 所示，配置 Memory Type 为 Single Port ROM。

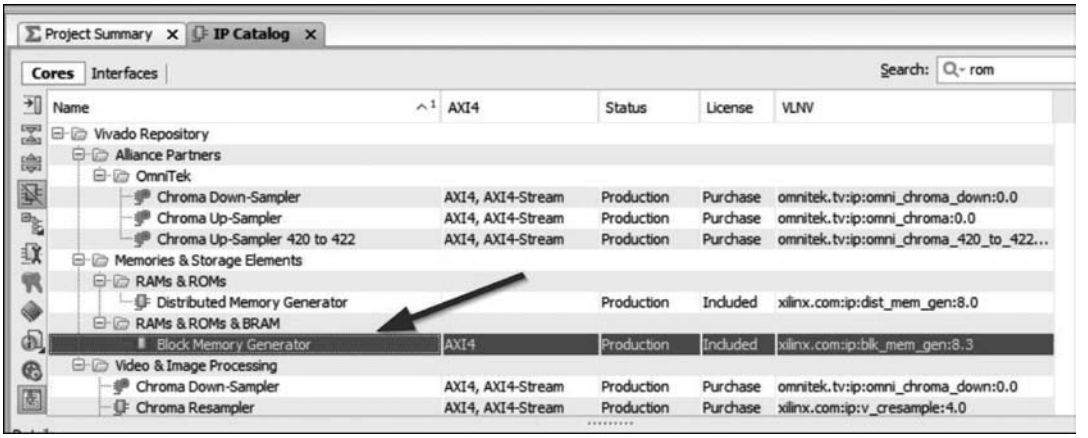


图 5.29　IP Catalog 中的 Block Memory Generator IP

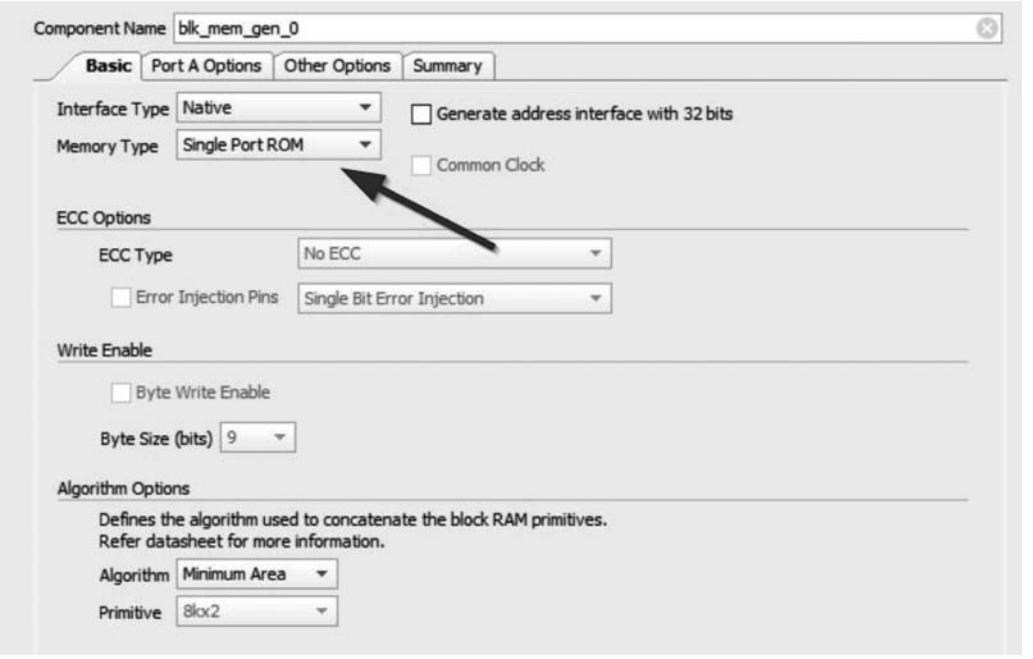


图 5.30　Block Memory Generator IP 基本配置页面

如图 5.31 所示，Port A Options 页面配置 Memory Size 为 256×8 位。

如图 5.32 所示，Other Options 页面勾选 Load Init File，加载预先准备好的 gamma_lut.coe 作为 ROM 初始化文件。注意这个 gamma_lut.coe 的生成可以使用 at7_img_ex05/matlab 下的 MATLAB 脚本文件 gammaCorrection.m 生成。

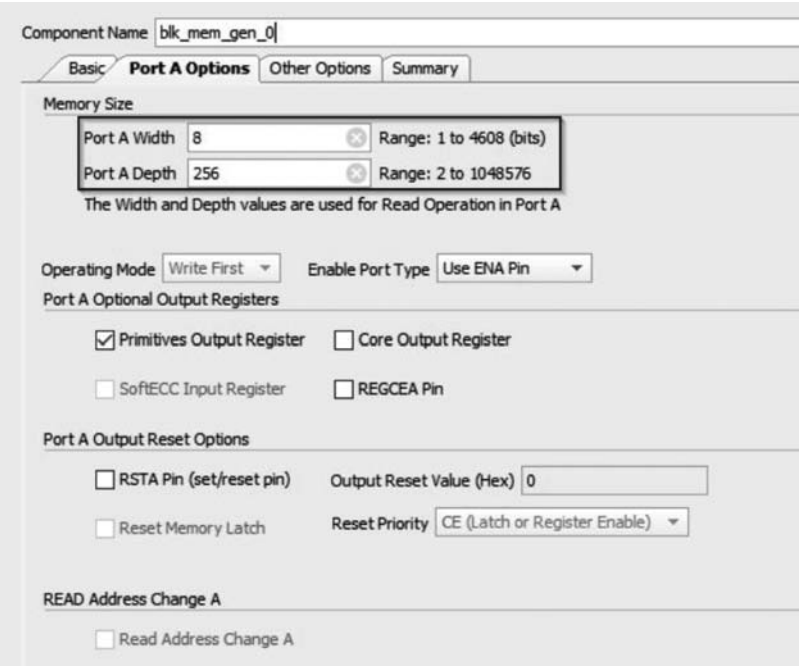


图 5.31　Block Memory Generator IP 端口 A 配置页面

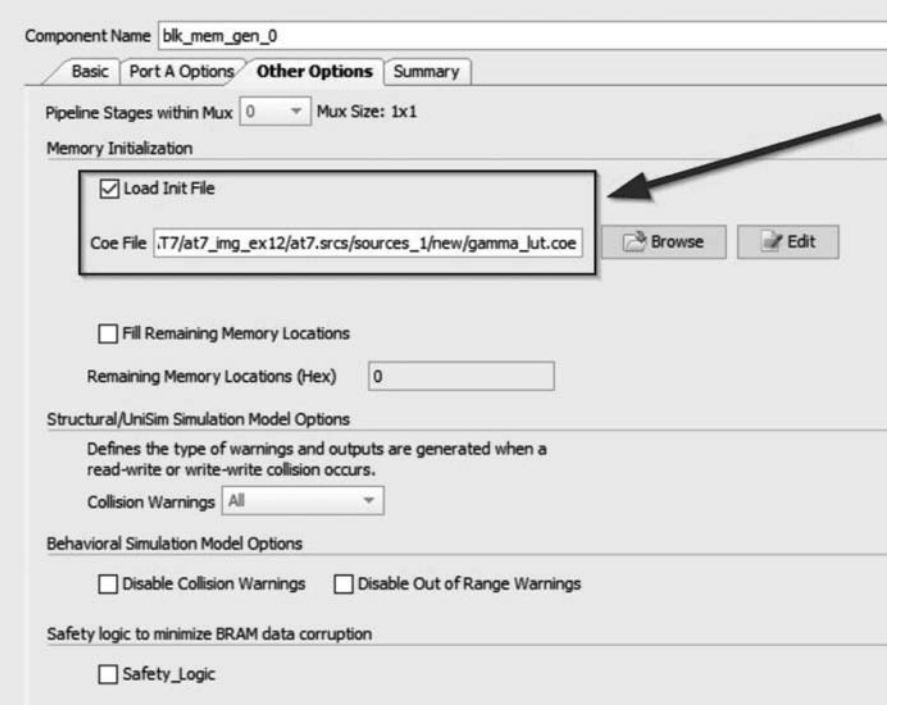


图 5.32　Block Memory Generator IP 的其他选项配置页面

5.3.5　FPGA 板级调试

连接好 OV5640 摄像头模块、VGA 模块和 FPGA 开发板，同时连接好 FPGA 的下载器并给板子供电。

使用 Vivado 2019.1 打开工程 at7_img_ex05，将 at7_img_ex05\at7.runs\impl_1 文件夹下的 at7.bit 文件烧录到板子上。如图 5.33 所示，可以看到 VGA 显示器的左右两侧同时有两个图像，左侧图像为原始的图像，右侧图像为进行 Gamma 校正后的图像。设计者可以使用不同的 Gamma 校正表，以实现不同的 Gamma 效果。

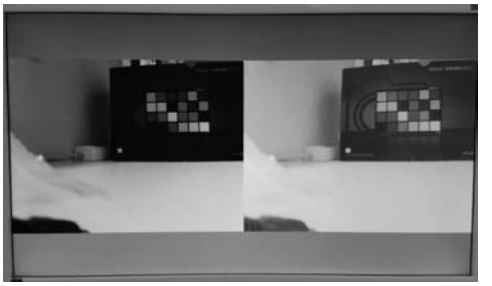


图 5.33　Gamma 校正图像效果（见彩插）

5.4　白平衡校正的 FPGA 实现

5.4.1　白平衡介绍

1. 为什么需要白平衡

白平衡的英文名称为 White Balance，简写为 WB。对于摄影和图像处理，白平衡调节的主要就是为了让图片中的白色渲染更接近实际的白色，或者通过白平衡的调节来改变图像色调，营造某种氛围。为了获得更准确的白色，在相机和修图工具中都有自定义白平衡的功能，而在我们大多数手机上却并没有自定义白平衡选项，对于手机来说，白平衡的校准是由软件自动完成的，并不需要我们参与，所以这也导致该功能最容易被我们忽略，但其实白平衡对整幅图像色彩的准确度起着至关重要的作用。

不同性质的光源会在画面中产生不同的色彩倾向，比如说，蜡烛的光线会使画面偏橘黄色，而黄昏过后的光线则会为景物披上一层蓝色的冷调。由于我们的视觉系统会自动对不同的光线作出补偿，所以无论在暖调还是冷调的光线环境下，我们看一张白纸永远还是白色的。但相机则不然，它只会直接记录呈现在它面前的色彩，这就会导致画面色彩偏暖或偏冷。因此，为了让实际环境中白色的物体在所拍摄的画面中也呈现出“真正”的白色，就需要有“白平衡校正”。

如图 5.34 所示，在不同的光照条件下，即存在着不同的色温，物体“偏光”的程度是不同的。理想情况下，当使用图像采集设备做图像采集时，若知道当前的光照条件（色温），那么通过一定的校正手段，就可以将图像的“偏光”校正回来，这便是白平衡的基本思想。当然现实情况下，我们往往不知道当前的色温状况，那么就需要一套通过当前图像计算出色温信息的算法，然后再去做校正，这便是很多成像设备中的“自动白平衡”功能。

图 5.34　不同光照条件下的色温(见彩插)

注：色温是表示光源光色的尺度，其单位为 K(kelvin)。

2. 如何进行白平衡校正

白平衡是一个很抽象的概念，最通俗的理解就是让白色所成的像依然为白色，如果白是“白”，那其他景物的成像也就会接近人眼的色彩视觉习惯。调整白平衡的过程叫作白平衡调整。

彩色图像传感器内部有三个感光元件，它们分别感受蓝色、绿色、红色的光线，在默认情况下这三个感光电路电子放大比例是相同的，为 1∶1∶1 的关系，白平衡的调整就是根据被调校的景物改变了这种比例关系。比如被调校景物的蓝、绿、红色光的比例关系是 2∶1∶1(蓝光比例多，色温偏高)，那么白平衡调整后的比例关系为 1∶2∶2，调整后的电路放大比例中明显蓝的比例减少，增加了绿和红的比例，所拍摄的影像经过这样的白平衡调整处理之后，蓝、绿、红的比例才会相同。也就是说，如果被调校的白色偏一点蓝，那么白平衡调整就改变正常的比例关系减弱蓝电路的放大，同时增加绿和红的放大比例，使所成影像依然为白色。这是白平衡校正最基本的原理。

很多图像采集设备内都有自动白平衡功能，有一套对当前采光状况做计算判断后生成白平衡校正参数并加以应用的算法。而这个实例中，我们并不去研究这一整套复杂的自动校准算法，而是用最简单、基本的一组运算，实时地做白平衡调整，让大家去感受手动调整白平衡的乐趣。

按照前面的理论，假设原始采集图像的色彩数据分别为 Ri、Gi、Bi，白平衡处理后的色彩数据分别为 Ro、Go、Bo，白平衡调整后色彩的最大取值为 VIO_R、VIO_G、VIO_B，那么它们的公式如下：

$$Ro = (VIO\_G/VIO\_R) \times Ri$$
$$Go = (VIO\_G/VIO\_G) \times Gi = Go$$
$$Bo = (VIO\_G/VIO\_B) \times Bi$$

这组公式如何理解呢？当我们需要做白平衡调整(实际光照通常不是理想的光照条

件),并且根据当前的色温推断(或者查表)采集到的图像色彩的最大取值为 VIO_R、VIO_G、VIO_B(大多数色温下,色彩值是到不了最大值的),那么只要把这三个色彩通道的每个像素值按照相应的比例关系调整到一样的"幅值"后,白平衡校正就做到了。而 G 色彩是人眼最敏感的,通常是以 G 色彩(VIO_G)当前的"幅值"作为标准,VIO_R 和 VIO_B 要做必要的运算以归一化到以 VIO_G 作为"幅值"。因此,从公式上看,(VIO_G/VIO_R)和(VIO_G/VIO_B)就是要算出这个归一化的比例关系,然后再乘以当前的色彩值 Ri 和 Bi,就获得归一化以后的色彩值(白平衡校正后的色彩值)。

5.4.2　FPGA 功能概述

如图 5.35 所示,这是整个视频采集和处理系统的功能框图。上电初始,FPGA 需要通过 SCCB 接口对 CMOS Sensor 进行寄存器初始化配置。这些初始化的基本参数,即初始化地址对应的初始化数据都存储在 FPGA 内。在初始化配置完成后,CMOS Sensor 就能够持

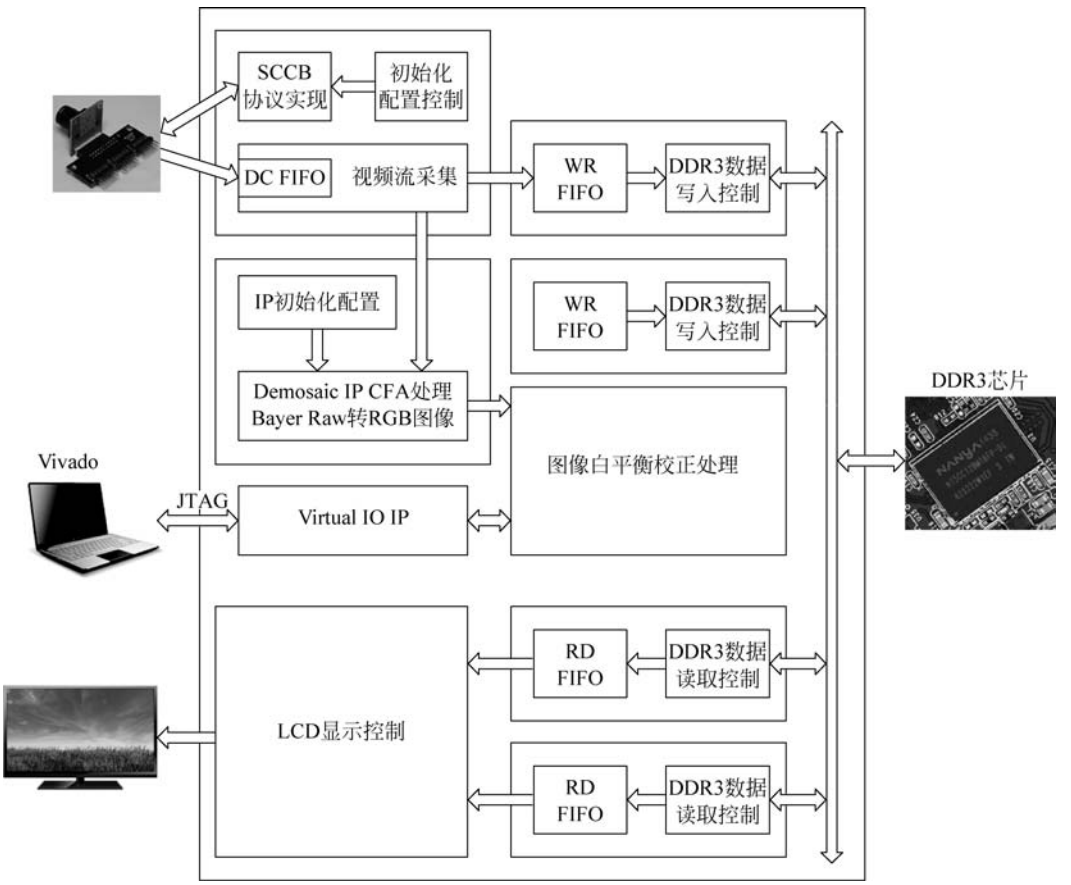


图 5.35　白平衡校正功能框图

续输出标准 Bayer Raw 的视频数据流，FPGA 通过对其同步信号，如时钟、行频和场频进行检测，从而从数据总线上实时地采集图像数据。

在 FPGA 内部，采集到的视频数据先通过一个 FIFO，将原本时钟域为 25MHz 下同步的数据流转换到 50MHz 下。接着将这个数据再送入写 DDR3 缓存的异步 FIFO 中，这个 FIFO 中的数据一旦达到一定数量，会被读取并进行 Bayer Raw 转 Color RGB 的处理，随后图像送往两个不同的模块：一个模块是将图像直接写入 DDR3 中，最终被读取并显示到 VGA 显示器的左侧；另一个模块会在图像缓存到 DDR3 之前，对图像进行白平衡校正处理，然后再写入 DDR3 另一片存储空间中，最终被读取的白平衡校正后图像显示在 VGA 显示器的右侧。与此同时，白平衡校正值的输入是通过连接 JTAG 接口，在 Vivado 中的 Virtual IO 在线调试界面进行配置实现的。

5.4.3 FPGA 设计说明

at7_img_ex06 工程源码的层次结构如图 5.36 所示。

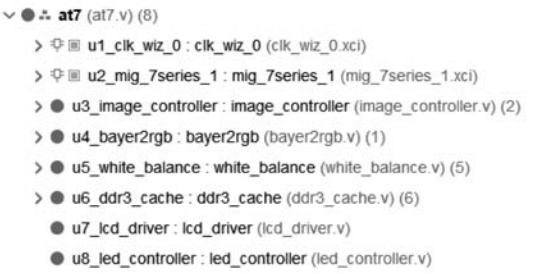


图 5.36　at7_img_ex06 工程源码层次结构

at7_img_ex06 工程模块及功能描述如表 5.7 所示。

表 5.7　at7_img_ex06 工程模块及功能描述

模块名称	功能描述
clk_wiz_0	该模块是 PLL IP 核的例化模块，该 PLL 用于产生系统中所需要的不同频率时钟信号
mig_7series_0	该模块是 DDR3 控制器 IP 核的例化模块。FPGA 内部逻辑读写访问 DDR3 都是通过该模块实现，该模块对外直接控制 DDR3 芯片
Image_controller	该模块及其子模块实现 SCCB 接口对 OV5640 的初始化、OV5640 输出图像的采集控制等。这个模块内部例化了两个子模块，I2C_OV5640_Init_RGB565 模块实现 SCCB 接口通信协议和初始化配置，其下例化的 I2C_Controller 模块实现 SCCB 协议，I2C_OV5640_RGB565_Config 模块用于产生图像传感器的初始配置数据，SCCB 接口的初始化配置控制实现则在 I2C_OV5640_Init_RGB565 模块中实现；image_capture 模块实现图像采集功能
bayer2rgb	该模块实现 Bayer Raw 图像转换为 RGB888 图像的处理。该模块例化了 Demosaic IP 核，通过 AXI4-Lite 接口对 IP 核初始化，通过 AXI4-Stream Video 接口实现 FPGA 逻辑与 IP 核之间的图像传输

续表

模块名称	功能描述
ddr3_cache	该模块主要用于缓存读或写 DDR3 的数据，其下例化了两个 FIFO。该模块连接 FPGA 内部逻辑与 DDR3 IP 核(mig_7series_0 模块)之间的数据交互
white_balance	该模块实现白平衡处理功能，例化的 VIO 模块，其 3 个输入值作为白平衡校正计算的主要参数
lcd_driver	该模块驱动 LCD，同时产生读取 DDR3 中图像数据的控制逻辑
led_controller	该模块控制 LED 闪烁，指示工作状态

如图 5.37 所示，这是 white_balance.v 模块的功能框图。VIO IP 输入的参数进行除法运算后，分别与 R 和 B 的输入(i_rgb_image_data[23:16]和 i_rgb_image_data[15:8])进行乘法运算，最后对结果做溢出处理判断；G 的输入(i_rgb_image_data[7:0])只需要缓存几拍，与 R 和 B 的乘除运算保持同步即可；输入图像有效信号 i_rgb_imag_vld 也只需要打 3 拍和输出数据保持同步，输出 o_wb_imag_vld 即可。

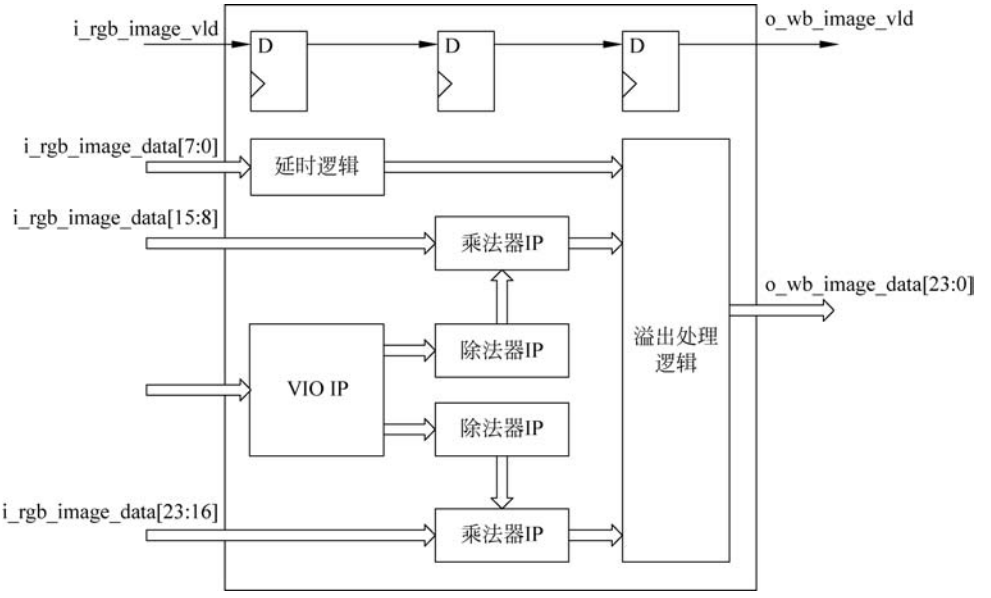


图 5.37　white_balance.v 模块的功能框图

关于 R 和 B 的运算，已经推导的公式如下：

$$Ro=(VIO\_G/VIO\_R)\times Ri$$

$$Go=(VIO\_G/VIO\_G)\times Gi=Go$$

$$Bo=(VIO\_G/VIO\_B)\times Bi$$

Go 由于赋值不变，所以无须做乘除运算。对于 Go 和 Bo 的运算，由于 VIO_G/VIO_R 或 VIO_G/VIO_B 的取值是一个小数，在 FPGA 中实现小数运算，必须先放大为整数。因

此，第一步除法运算获得中间结果的公式如下：

$$Rc = VIO\_G \times 1024 / VIO\_R$$

$$Bc = VIO\_G \times 1024 / VIO\_B$$

在 FPGA 中，“×1024”的运算，通过左移 10 位来实现。除法器的 IP 核中，被除数表示为{6'd0，vio_g，10'd0}，以此实现 VIO_G×1024。

第二步乘法运算后，相应地需要缩小为原来的 1/1024，公式如下：

$$Ro = Rc \times Ri / 1024$$

$$Bo = Bc \times Bi / 1024$$

在 FPGA 中，“/1024”的运算，通过右移 10 位来实现，即最终的乘法运算结果，取 mul_r[17:10]和 mul_b[17:10]作为最终输出。

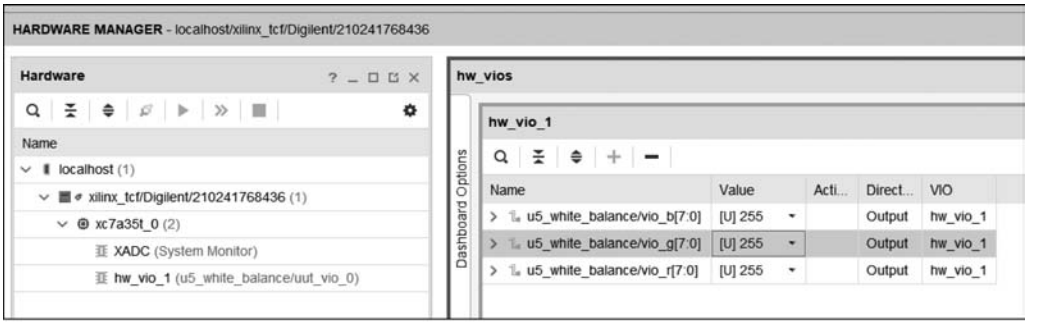
对于溢出判断部分，我们用最终获取的 Ro、Go、Bo 分别和 VIO_G 进行比较，若大于 VIO_G 则以 VIO_G 的值替代，否则保持原值。

5.4.4 FPGA 板级调试

连接好 OV5640 摄像头模块、VGA 模块和 FPGA 开发板，同时连接好 FPGA 的下载器并给板子供电。

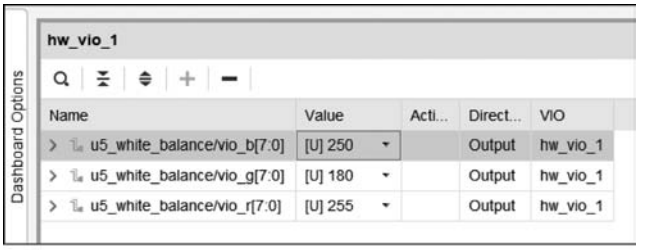
使用 Vivado 2019.1 打开工程 at7_img_ex06，将 at7_img_ex06\at7.runs\impl_2 文件夹下的 at7.bit 文件以及 debug_nets.ltx 文件烧录到 FPGA 器件中，可以看到 VGA 显示器同时显示左右两个图像，左侧图像为原始图像，右侧显示进行白平衡校正后的图像。通过 VIO 调试界面可以调整白平衡校正的三个最大 R、G、B 通道取值，可以改变右侧白平衡校正图像的显示效果。

如图 5.38(a)所示，双击打开 hw_vio_1 的调试界面，将 vio_b、vio_g、vio_r 这 3 组信号添加到调试主界面中，默认情况下的取值都是 255，即显示器上的左和右两个视频图像是一样的效果。如图 5.38(b)所示，将当前的 vio_b、vio_g、vio_r 取值分别设置为 250、180 和 255。



(a) 默认数值

图 5.38 VIO 调试界面数值



(b) 修改数值

图 5.38　(续)

如图 5.39 所示,修改了 VIO 参数后,原本整体有些发红的图像(图 5.39 左侧),在做了白平衡校正后,图像(图 5.39 右侧)发红的现象有很大改善,基本能够还原真实的白色了(当然了,按照我们当前的基本算法,色彩的整体亮度会有一定的牺牲)。

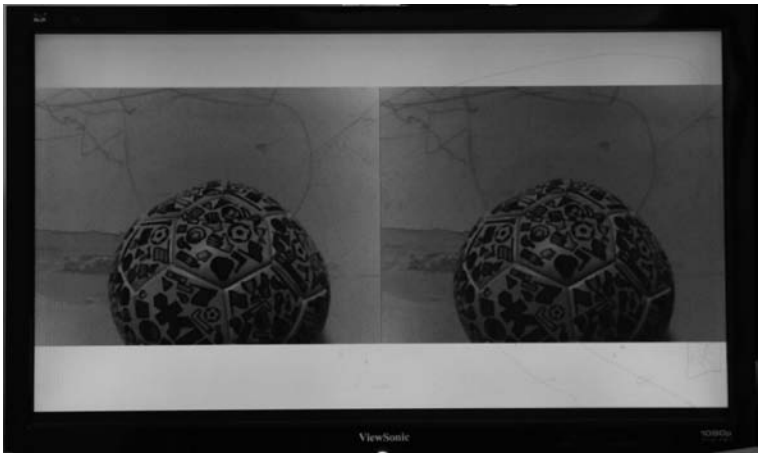


图 5.39　白平衡校正效果图(见彩插)

5.5　色彩空间转换与图像增强 IP 核的仿真

5.5.1　图像增强 IP 简介

Xilinx 的 Vivado 中集成的图像增强(Image Enhancement)IP 可以有效降低图像噪声并增强图像边缘。该 IP 使用了 2D 滤波方式,可以在达到更好的图像噪声抑制的同时,保留并增强图像边缘。

对于一个比较经典的图像前端处理,图像增强也是一个必不可少的步骤。在这个实例中,我们需要让图像分别经过 RGB to YCbCr 模块、图像增强模块和 YCbCr to RGB 模块的处理,这 3 个模块在 Vivado 中都有可用的 IP 核。

图像增强 IP 的功能框图如图 5.40 所示。该 IP 输入和输出的图像数据必须为 YUV444 或 YUV422 模式；待处理图像进入 IP 后，首先需要多行缓存，然后分别通过降噪(Noise Reduction)、边缘检测(Edge Map Morphology)、边缘增强(Edge Enhance)模块。完成处理后的图像再拟合在一块，最后会通过可选的光环抑制(Anti-halo)和锯齿消除(Anti-alias)模块，完成最终图像输出。边缘增强和噪声抑制实际上是两个完全相反的图像处理方式，为了保证两个模块能够更好地实现增强图像的效果，在这个 IP 中，第一步做的是图像的形态检测(Edge Map Morphology)，然后再根据这个结果，对图像中需要降噪的部分和边缘增强的部分分别处理。

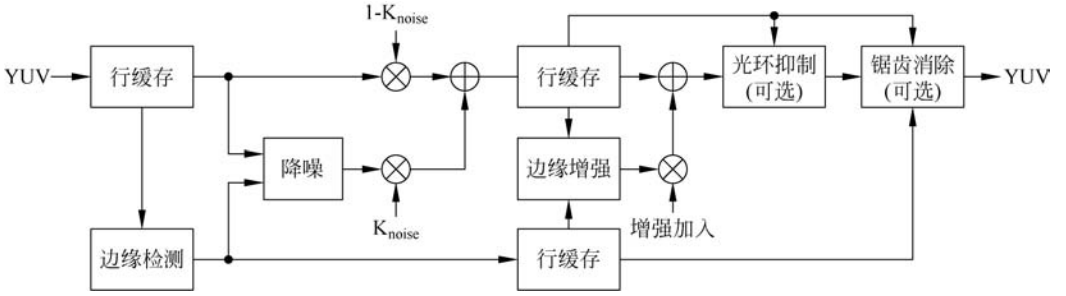


图 5.40　图像处理 IP 功能框图

1. 图像形态检测

图像形态检测是整个图像增强的第一步，它用于指示后续需要对图像进行的降噪或边缘增强操作。图像形态检测主要包括下面两步：

(1) 经过二维的 FIR 滤波器，从水平、垂直以及两个对角共 4 个维度提取边缘信息。

(2) 使用拉长、正交的结构单元和形态学处理，用于提供清晰的各个方向边缘信息。

2. 降噪处理

降噪处理是基于中心像素点以及特定的临近像素点的滤波实现的。算法实现类似高斯的定向低通滤波。噪声门限由 IP 核的设置决定。图像形态检测信息标定出的边缘不会做任何的降噪处理。降噪处理功能框图如图 5.41 所示。

3. 边缘增强

IP 核设定的边缘增强参数决定了边缘增强的幅度。根据边缘检测形态信息，边缘增强模块对标记处的边缘做拉普拉斯滤波，实现边缘增强效果。边缘增强功能框图如图 5.42 所示。

4. 光环抑制与锯齿消除

光环抑制(Anti-halo)和锯齿消除(Anti-alias)模块是可选的功能块。前面进行噪声抑制和边缘增强后的图像，可能存在图像边缘被放大或抑制的情况。光环抑制和锯齿消除，通过每个新的像素值与原图像的像素值以及邻近 8 个像素值的比较，以判断其是否需要进行相应的处理并实现图像的优化。

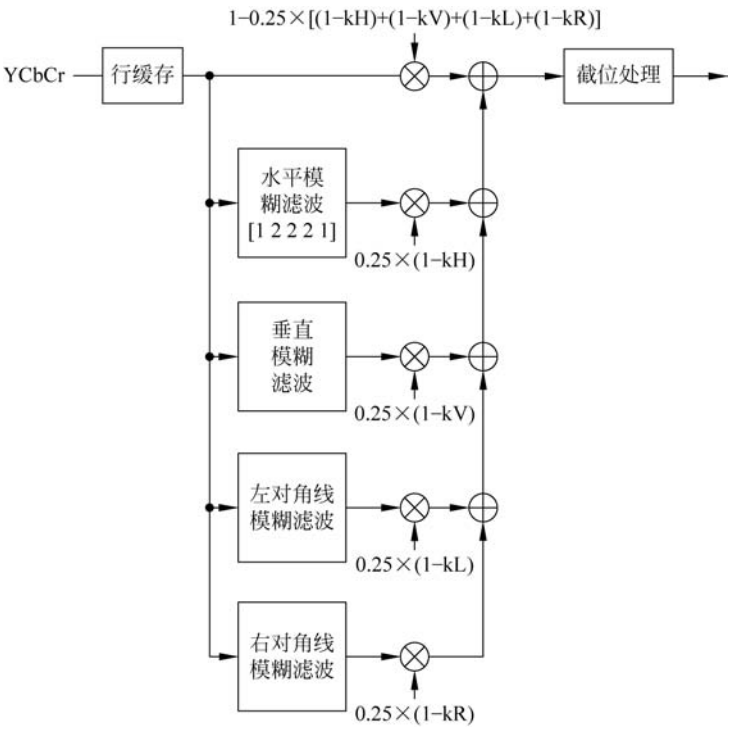


图 5.41　降噪处理功能框图

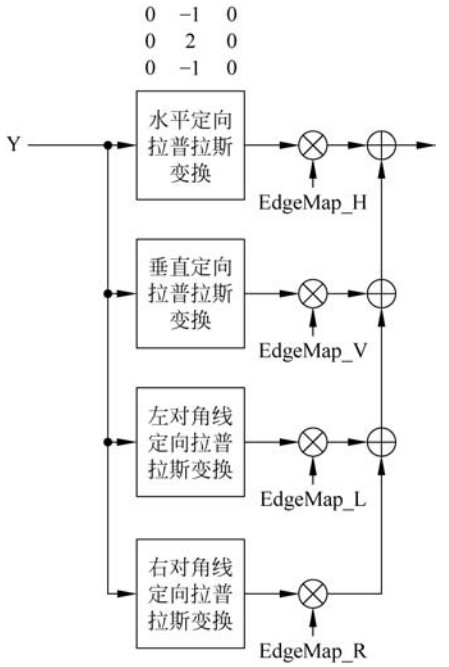


图 5.42　边缘增强功能框图

如图 5.43 所示，原图在图像增强后可能出现光环现象，那么经过光环抑制后图像就能够实现最优化。

(a) 原图　　(b) 光环　　(c) 光环抑制

图 5.43　原图、光环图像和光环抑制图像

5.5.2　IP 添加与配置

Vivado 的 IP Catalog 中，在 Video & Image Processing 分类下，如图 5.44 所示，可以看到有很多可用的图像处理 IP 核。我们需要用到的 RGB to YCbCr、Image Enhancement 和 YCbCr to RGB 这 3 个 IP 核，在该分类下都可以找到。

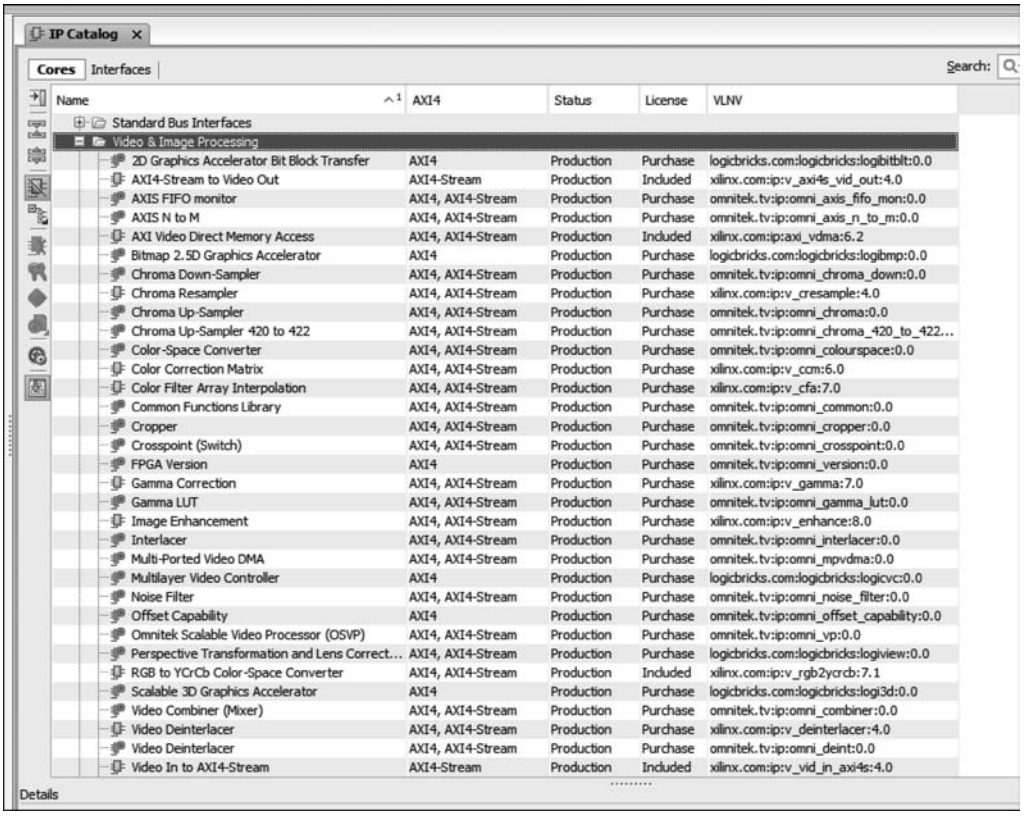


图 5.44　Video & Image Processing IP 核

1. RGB to YCbCr IP 配置

双击 IP Catalog 中的 RGB to YCrCb Color-Space Converter 这个 IP 核，进入 Features 配置页面，如图 5.45 所示。设置图像位宽 8 位，分辨率 640×480 像素，YUV(YCbCr)格式，输出图像取值范围 0～255。

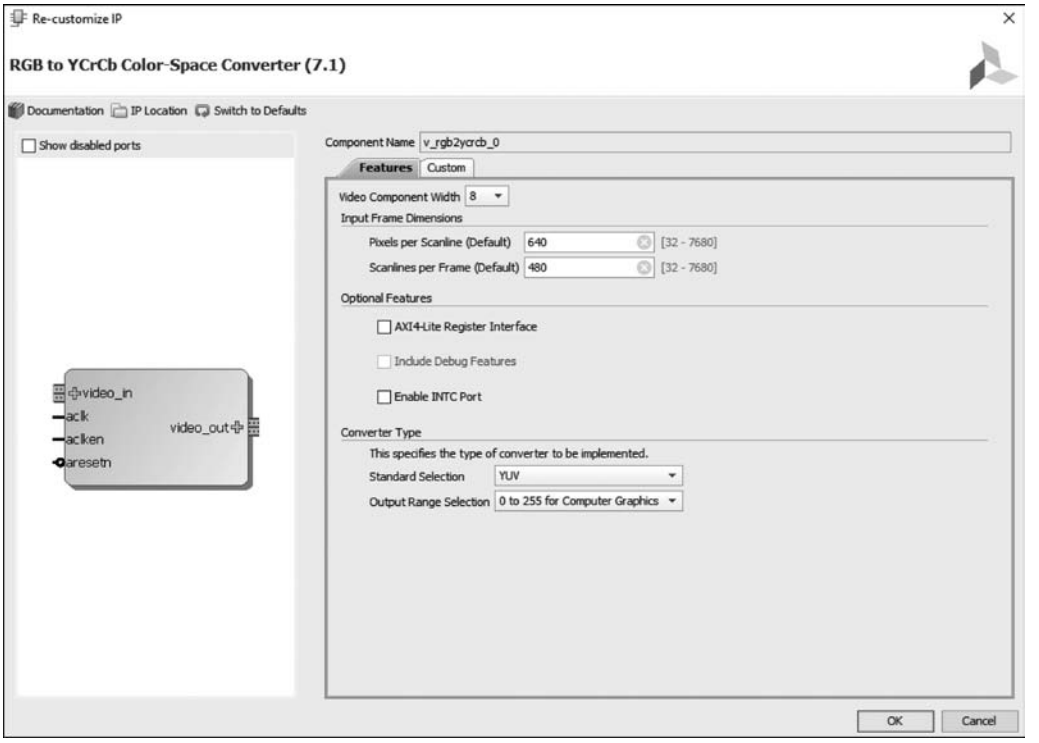


图 5.45　RGB to YCrCb IP 核的 Features 配置页面

如图 5.46 所示，在 Custom 配置页面中，可以看到 RGB to YCbCr 转换的基本公式参数。

2. Image Enhancement IP 配置

双击 IP Catalog 中的 Image Enhancement IP 核，进入配置页面如图 5.47 所示。设定图像位宽 8 位，图像分辨率 640×480 像素，图像噪声抑制(Image Noise Reduction)水平(取值 0～255)，图像边缘增强(Image Edge Enhancement)水平(取值 0～1.0)，以及可选的光环抑制(Halo Suppression)和锯齿消除(Anti-Alias Filtering)。

3. YCbCr to RGB IP 配置

双击 IP Catalog 中的 YCrCb to RGB Color-Space Converter 这个 IP 核，如图 5.48 所示，进入 Feature 配置页面，设置图像位宽 8 位，分辨率 640×480 像素，YUV 格式，输出图像取值范围 0～255。

如图 5.49 所示，Custom 配置页面中，可以看到 YCbCr to RGB 转换的基本公式参数。

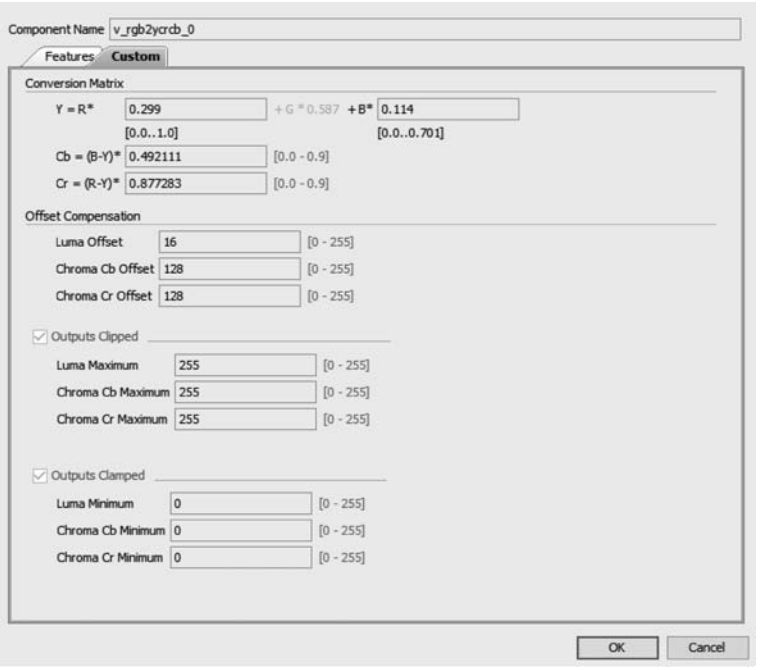


图 5.46 RGB to YCrCb IP 核的 Custom 配置页面

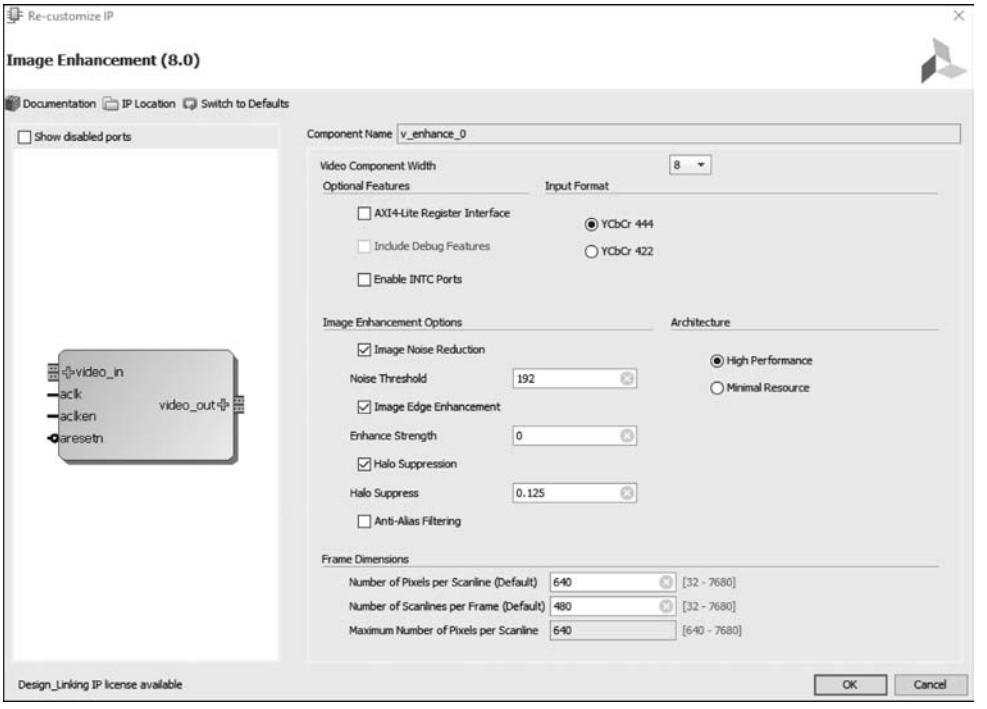


图 5.47 Image Enhancement IP 核配置页面

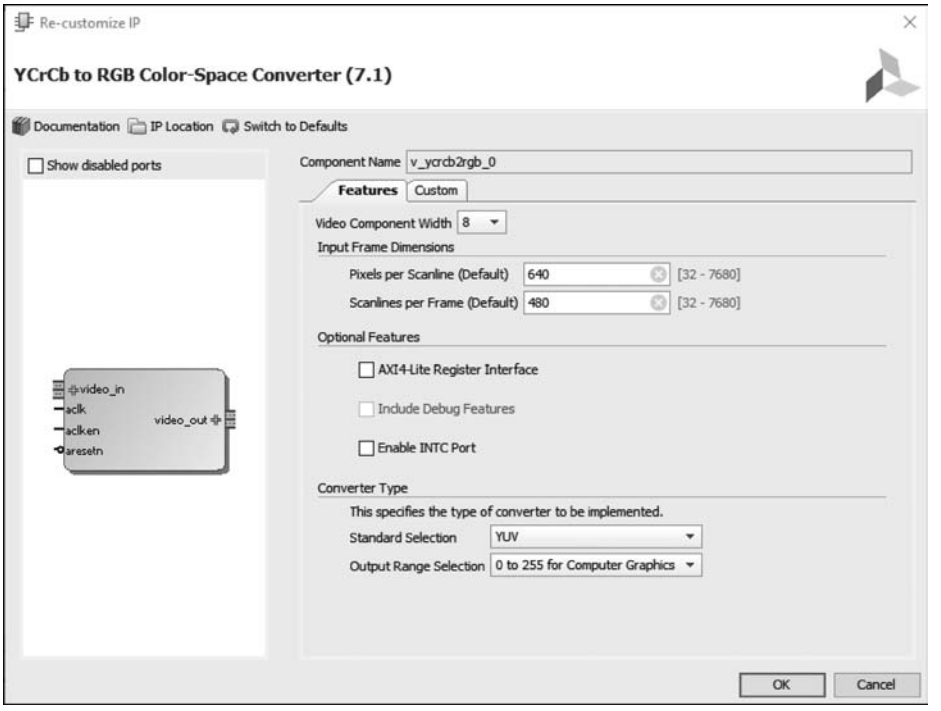


图 5.48　YCrCb to RGB IP 核 Features 配置页面

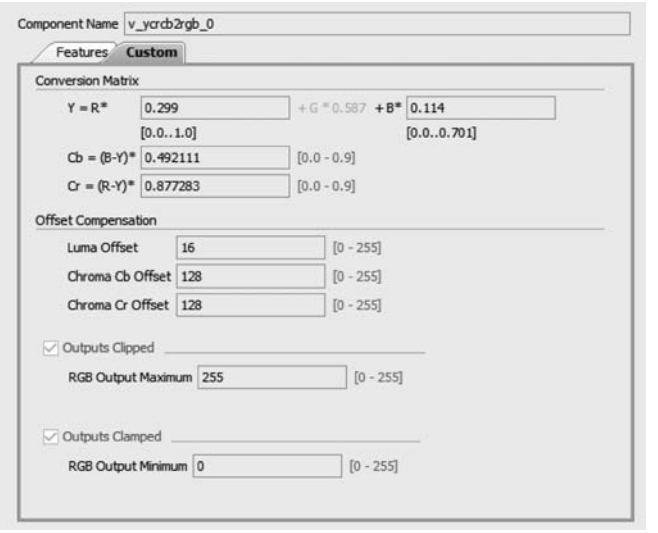


图 5.49　YCrCb to RGB IP 核 Custom 配置页面

5.5.3 协同仿真的 MATLAB 脚本说明

使用 at7_img_ex07\matlab 文件夹下的 MATLAB 源码 image_txt_generation.m 产生作为 FPGA 仿真输入的测试图像数据，存储在文本 image_in_hex.txt 中。FPGA 仿真测试运行后，将会产生图像增强后的图像数据，存储在文本 FPGA_Enhancement_Image.txt 中，使用 MATLAB 的脚本 draw_image_from_FPGA_result.m 可以调用这个文本中的图像数据，同时显示图像增强前后的效果供比对。MATLAB 产生与调用文本的示意图如图 5.50 所示。

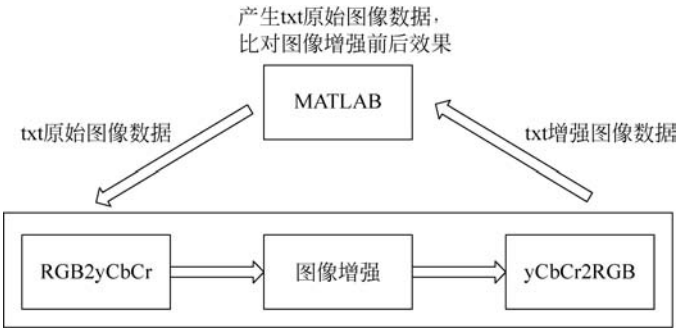


图 5.50 MATLAB 产生与调用文本示意图

1. 测试激励图像产生

工程路径“\at7_img_ex07\matlab”下的 image_txt_generation.m 脚本，在 MATLAB 中运行后，会将同路径下的 test.bmp 图像的色彩信息转换为 txt 文本（同路径下的 image_in_hex.txt）以十六进制保存。

```
clc;clear all;close all;

IMAGE_WIDTH = 640;
IMAGE_HIGHT = 480;

%load origin image
%I = imread('Lena_gray_niose.bmp');
I = imread('test.bmp');

I = rgb2gray(I);

%fclose(fid1);

%% output image data in hex file
raw_image = reshape(I, IMAGE_HIGHT, IMAGE_WIDTH);
raw_image = raw_image';
```

```
fid2 = fopen('image_in_hex.txt', 'wt');

fprintf(fid2, '%04x\n', raw_image);
fid2 = fclose(fid2);


%show origin image
figure,imshow(I);
title('Original image');
```

2. 测试结果比对

工程路径“\at7_img_ex07\matlab”下的 draw_image_from_FPGA_result.m 脚本，在 MATLAB 中运行后，会将同路径下的 test.bmp 图像以及 FPGA 仿真生成的 FPGA_Enhancement_Image.txt 文本中所存储的图像增强后的图像同时显示供查看比对。FPGA_Enhancement_Image.txt 文本在 FPGA 仿真完成后，需要从“at7_img_ex07\at7.sim\sim_1\behav”文件夹复制到“\at7_img_ex07\matlab”文件夹下。

```
clc;clear all;close all;

IMAGE_WIDTH = 640;
IMAGE_HIGHT = 480;

%load fft fiter image data from txt
fid1 = fopen('FPGA_Enhancement_Image.txt', 'r');
img = fscanf(fid1,'%x');
fclose(fid1);
img2 = reshape(img,IMAGE_WIDTH,IMAGE_HIGHT);
img2 = img2';

%load origin image
I = imread('test.bmp');
I = rgb2gray(I);


%show origin image
figure,imshow(I);
title('Original image');

%show fft fiter image with FPGA
figure,imshow(img2,[])
title('Image Enhancement with FPGA');
```

5.5.4 FPGA 仿真说明

如图 5.51 所示，MATLAB 产生的原始图像数据 image_in_hex.txt 需要在仿真开始前

放置在 at7_img_ex07\at7.sim 文件夹下。

图 5.51 仿真原始图像存储文本

使用 Vivado 打开 at7_img_ex07 工程，在 Sources 面板中，展开 Simulation Sources→sim_1，将 at7_image_enhance_sim.v 文件设置为 top module。选择 Flow Navigator 面板的 Simulation→Run Simulation 打开仿真页面。

如图 5.52 所示，仿真测试结果位于 at7_img_ex07\at7.sim\sim_1\behav\xsim 文件夹下。

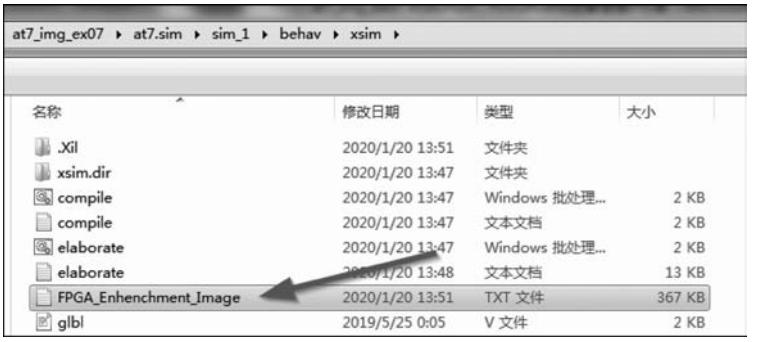


图 5.52 仿真结果存储文本

在设定 Noise Threshold = 192，Enhancement Strength = 0.0，Halo Suppression = 0.125 时，用 MATLAB 比对图像如图 5.53 所示。

图 5.53 比对图像 1

在设定 Noise Threshold＝192，Enhancement Strength＝0.125，Halo Suppression＝0.75 时，与 MATLAB 比对图像如图 5.54 所示。

图 5.54　比对图像 2

在设定 Noise Threshold＝192，Enhancement Strength＝1.0，Halo Suppression＝0.75 时，用 MATLAB 比对图像如图 5.55 所示。

图 5.55　比对图像 3

5.6　色彩空间转换的 FPGA 实现

5.6.1　功能概述

如图 5.56 所示，这是整个视频采集和处理系统的功能框图。上电初始，FPGA 需要通过 SCCB 接口对 CMOS Sensor 进行寄存器初始化配置。这些初始化的基本参数，即初始化地址对应的初始化数据都存储在 FPGA 内。在初始化配置完成后，CMOS Sensor 就能够持续输出标准 Bayer Raw 的视频数据流，FPGA 通过对其同步信号，如时钟、行频和场频进行检测，从而从数据总线上实时地采集图像数据。

在 FPGA 内部，采集到的视频数据先通过一个 FIFO，将原本时钟域为 25MHz 下同步的数据流转换到 50MHz 下。接着将这个数据再送入写 DDR3 缓存的异步 FIFO 中，这个 FIFO 中的数据一旦达到一定数量，会被读取并进行 Bayer Raw 转 Color 的处理，随后图像

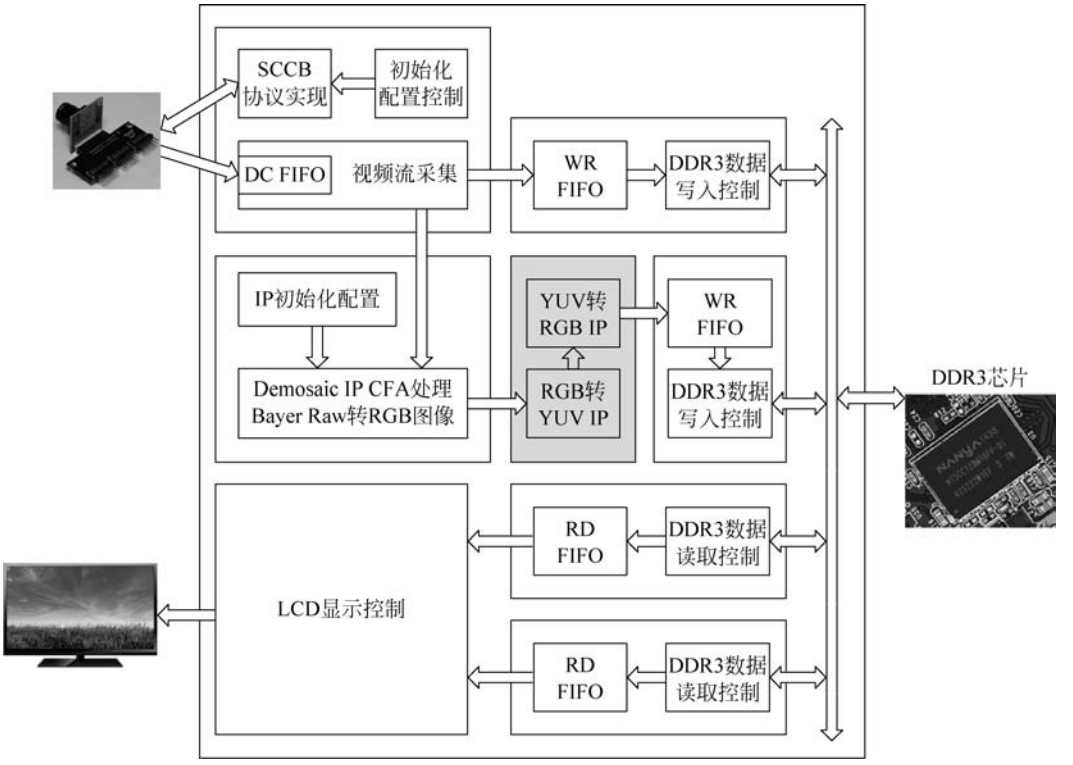


图 5.56　RGB2YUV 与 YUV2RGB 实现功能框图

送往两个不同的模块：一个模块将图像直接写入 DDR3 中，最终读取 DDR3 中的彩色图像显示到 VGA 显示器的左侧；另一个模块会在彩色图像缓存到 DDR3 之前，对图像进行 RGB 和 YUV 的互转（即 RGB 先转换为 YUV，然后 YUV 再转换为 RGB，此例子主要是为了应用这 2 个 IP 核）处理，然后再写入 DDR3 另一片存储空间中，最终读取 DDR3 中的这部分图像显示到 VGA 显示器的右侧。

5.6.2　RGB 与 YUV 介绍

1. 基本概念

1）什么是 RGB

对一种颜色进行编码的方法统称为“颜色空间”或“色域”。用最简单的话说，世界上任何一种颜色的“颜色空间”都可定义成一个固定的数字或变量。RGB（红、绿、蓝）只是众多颜色空间的一种。采用这种编码方法，每种颜色都可用三个变量来表示，即红色、绿色和蓝色的强度。存储或显示彩色图像时，RGB 是最常见的一种方案。

2）什么是 YUV

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），最后发送端将这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。由此可见，RGB 和 YUV 都属于颜色空间（或者叫“色彩空间”）。

3）YUV 与 YCbCr 是否一样

YCbCr 其实是 YUV 经过缩放和偏移的翻版，其中的 Y 与 YUV 中的 Y 含义一致，Cb 和 Cr 同样都指色彩，只是在表示方法上不同而已。YCbCr 中的 Y 是指亮度分量，Cb 指蓝色色度分量，而 Cr 指红色色度分量。YCbCr 应用领域很广，JPEG、MPEG 均采用此格式。YCbCr 可以被认为是与 YUV 等同的一种色彩表示方式，一般人们所讲的 YUV 大多是指 YCbCr。

2. RGB 和 YUV 的优缺点

RGB 缺乏与早期黑白显示系统的良好兼容性。因此，许多电子电器厂商普遍采用的做法是，将 RGB 转换成 YUV 颜色空间，以维持兼容，再根据需要转换回 RGB 格式，以便在计算机显示器上显示彩色图形。

YUV 主要用于优化彩色视频信号的传输，使其向后兼容老式黑白电视。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 3 个独立的视频信号同时传输）。

YUV 色彩空间最重要性的是它的亮度信号 Y 和色度信号 U、V 是分离的。如果只有 Y 信号分量而没有 U、V 分量，那么这样表示的图像就是黑白灰度图像。彩色电视采用 YUV 色彩空间正是为了用亮度信号 Y 解决彩色电视机与黑白电视机的兼容问题，使黑白电视机也能接收彩色电视信号。

3. YUV 和 RGB 的实现原理

RGB 是从颜色发光的原理来设计制定的，通俗点说它的颜色混合方式就好像有红、绿、蓝三盏灯，当它们的光相互叠合的时候，色彩相混，而亮度却等于两者亮度之和，越混合亮度越高，即加法混合。

红、绿、蓝三盏灯的叠加情况，中心三色最亮的叠加区为白色，加法混合的特点：越叠加越明亮。

红、绿、蓝三个颜色通道中每种颜色各分为 256 阶亮度，在 0 时“灯”最弱——是关掉的，而在 255 时“灯”最亮。当三色灰度数值相同时，产生不同灰度值的灰色调，即三色灰度都为 0 时，是最暗的黑色调；三色灰度都为 255 时，是最亮的白色调。

RGB 颜色称为加成色，因为将 R、G 和 B 添加在一起（即所有光线反射回眼睛）可产生白色。加成色用于照明光、电视和计算机显示器。例如，显示器通过红色、绿色和蓝色荧光粉发射光线产生颜色。绝大多数可视光谱都可表示为红、绿、蓝（RGB）三色光在不同比例和强度上的混合。这些颜色若发生重叠，则产生青、洋红和黄。

在 YUV 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”是通过 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。

4. YUV 的格式

YUV 码流的存储格式其实与其采样的方式密切相关，主流的格式有三种：YUV4:4:4、YUV4:2:2 和 YUV4:2:0，后两种格式都有一定的色彩信息丢失，但由于 YUV 格式中人眼敏感的亮度信息集中在了 Y 分量上，人眼相对不敏感的 U 和 V 分量的下采样并不会带来太大的人眼视觉冲击。因此，相比于 RGB 格式，YUV 格式的下采样在尽可能降低图像质量的前提下，大大降低了传输图像的数据带宽要求。

用三个图来直观地表示采样方式如图 5.57 所示，以黑点表示采样该像素点的 Y 分量，以空心圆圈表示采用该像素点的 UV 分量。

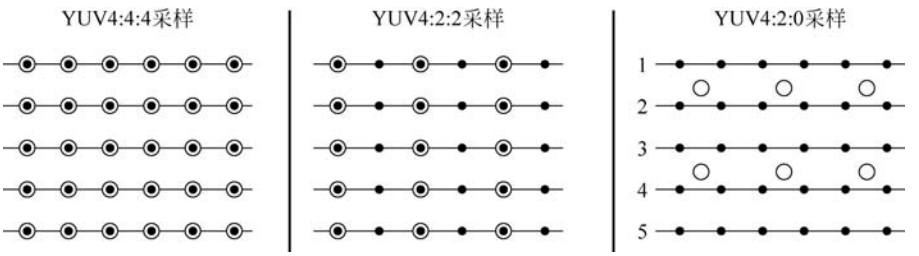


图 5.57　YUV 的三种采样示意图

YUV 4:4:4 采样，每一个 Y 对应一组 UV 分量。

YUV 4:2:2 采样，每两个 Y 共用一组 UV 分量。

YUV 4:2:0 采样，每四个 Y 共用一组 UV 分量。

5. RGB 和 YUV 转换公式

YUV 可以从 8 位 RGB 直接计算，如下：

```
Y = 0.299 R + 0.587 G + 0.114 B
U = - 0.1687 R - 0.3313 G + 0.5 B + 128
V = 0.5 R - 0.4187 G - 0.0813 B + 128
```

反过来，RGB 也可以直接从 YUV 计算：

```
R = Y + 1.402 (V-128)
G = Y - 0.34414 (U-128) - 0.71414 (V-128)
B = Y + 1.772 (U-128)
```

RGB 转换为 YCbCr 格式，如下：

```
Y' = 0.257 * R' + 0.504 * G' + 0.098 * B' + 16
```

```
Cb' = -0.148 * R' - 0.291 * G' + 0.439 * B' + 128
Cr' = 0.439 * R' - 0.368 * G' - 0.071 * B' + 128
```

反过来，RGB 也可以直接从 YCbCr 计算：

```
R' = 1.164 * (Y' - 16) + 1.596 * (Cr' - 128)
G' = 1.164 * (Y' - 16) - 0.813 * (Cr' - 128) - 0.392 * (Cb' - 128)
B' = 1.164 * (Y' - 16) + 2.017 * (Cb' - 128)
```

注意上面各个符号都带了一撇(')，表示该符号在原值基础上进行了 Gamma 校正，有助于弥补在抗锯齿的过程中线性分配 Gamma 值所带来的细节损失，使图像细节更加丰富。在没有采用 Gamma 校正的情况下，暗部细节不容易显现出来，而采用了这一图像增强技术以后，图像的层次更加明晰了。

5.6.3　FPGA 设计说明

at7_img_ex08 工程源码的层次结构如图 5.58 所示。

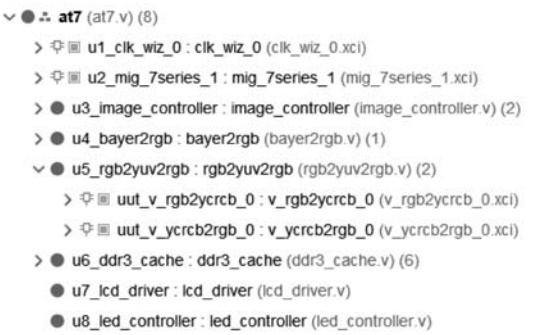


图 5.58　at7_img_ex08 工程源码层次结构

at7_img_ex08 工程模块及功能描述如表 5.8 所示。

表 5.8　at7_img_ex08 工程模块及功能描述

模 块 名 称	功 能 描 述
clk_wiz_0	该模块是 PLL IP 核的例化模块，该 PLL 用于产生系统中所需要的不同频率时钟信号
mig_7series_0	该模块是 DDR3 控制器 IP 核的例化模块。FPGA 内部逻辑读写访问 DDR3 都是通过该模块实现，该模块对外直接控制 DDR3 芯片
Image_controller	该模块及其子模块实现 SCCB 接口对 OV5640 的初始化、OV5640 输出图像的采集控制等。这个模块内部例化了两个子模块，I2C_OV5640_Init_RGB565.v 模块实现 SCCB 接口通信协议和初始化配置，其下例化的 I2C_Controller.v 模块实现 SCCB 协议，I2C_OV5640_RGB565_Config.v 模块用于产生图像传感器的初始配置数据，SCCB 接口的初始化配置控制实现则在 I2C_OV5640_Init_RGB565.v 模块中实现；image_capture.v 模块实现图像采集功能

续表

模块名称	功能描述
bayer2rgb	该模块实现 Bayer Raw 图像转换为 RGB888 图像的处理。该模块例化了 Demosaic IP 核，通过 AXI4-Lite 接口对 IP 核初始化，通过 AXI4-Stream Video 接口实现 FPGA 逻辑与 IP 核之间的图像传输
ddr3_cache	该模块主要用于缓存读或写 DDR3 的数据，其下例化了两个 FIFO。该模块连接 FPGA 内部逻辑与 DDR3 IP 核(mig_7series_0.v 模块)之间的数据交互
rgb2yuv2rgb	该模块依次对输入的 RGB 数据做 RGB 转 YUV、YUV 转 RGB 的处理
lcd_driver	该模块驱动 LCD，同时产生读取 DDR3 中图像数据的控制逻辑
led_controller	该模块控制 LED 闪烁，指示工作状态

1. RGB to YCbCr IP 配置

双击 IP Catalog 中的 RGB to YCrCb Color-Space Converter 这个 IP 核，进入 Features 配置页面，如图 5.59 所示。设置图像位宽 8 位，分辨率 640×480 像素，YUV(YCbCr)格式，输出图像取值范围 0～255。

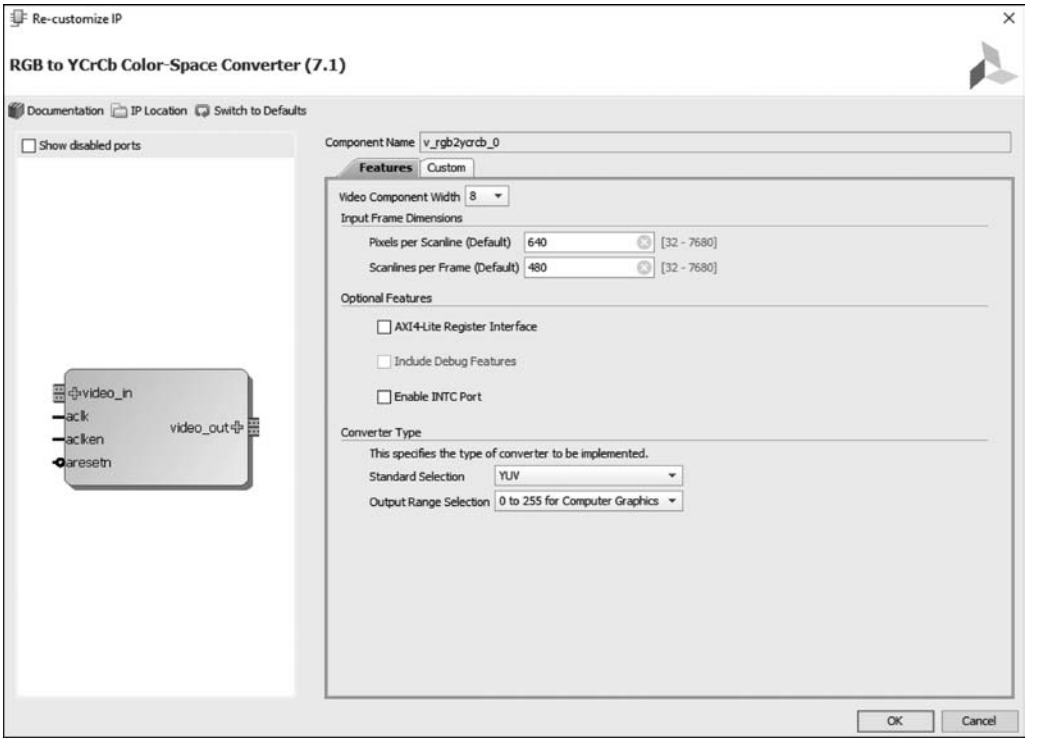


图 5.59　RGB to YCrCb IP 核的 Features 配置页面

如图 5.60 所示，在 Custom 配置页面中，可以看到 RGB to YCbCr 转换的基本公式参数。

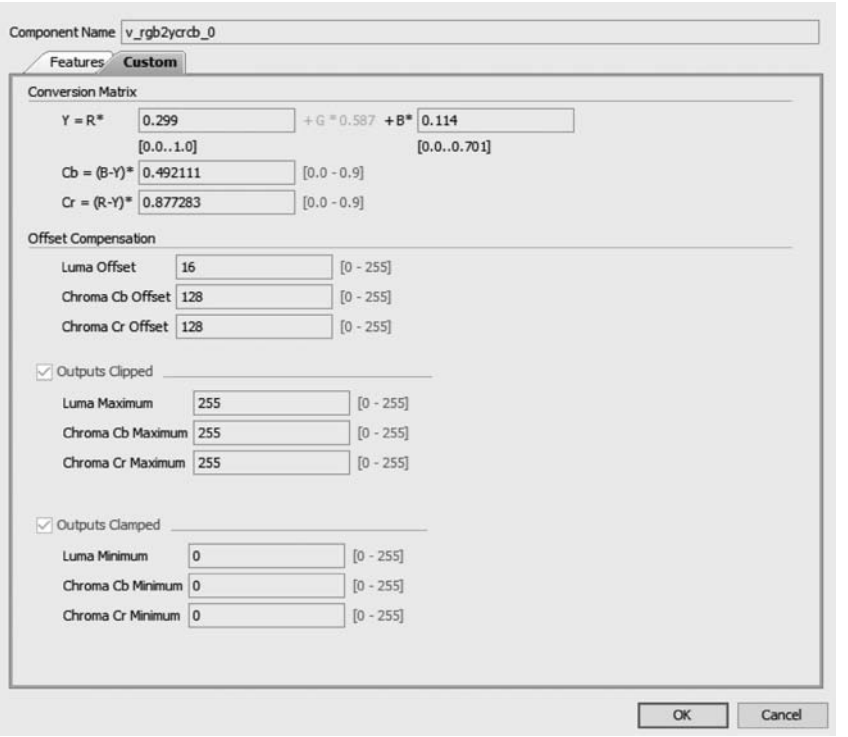


图 5.60　RGB to YCrCb IP 核的 Custom 配置页面

2. YCbCr to RGB IP 配置

双击 IP Catalog 中的 YCrCb to RGB Color-Space Converter 这个 IP 核，如图 5.61 所示，进入 Features 配置页面，设置图像位宽 8 位，分辨率 640×480 像素，YUV 格式，输出图像取值范围 0～255。

如图 5.62 所示，Custom 配置页面中，可以看到 YCbCr to RGB 转换的基本公式参数。

5.6.4　FPGA 板级调试

连接好 OV5640 摄像头模块、VGA 模块和 FPGA 开发板，同时连接好 FPGA 的下载器并给板子供电。

使用 Vivado 2019.1 打开工程 at7_img_ex08，将 at7_img_ex08\at7.runs\impl_1 文件夹下的 at7.bit 文件烧录到板子上，可以看到 VGA 显示器同时显示左右两个图像，左侧图像为原始图像，右侧图像为做过 RGB 和 YUV 互转的效果。由于在 RGB 和 YUV 转换过程中，有一定的图像精度损失，所以通过左右两个图像比对可以看出，右侧的图像质量相比左侧的图像会稍差一些。

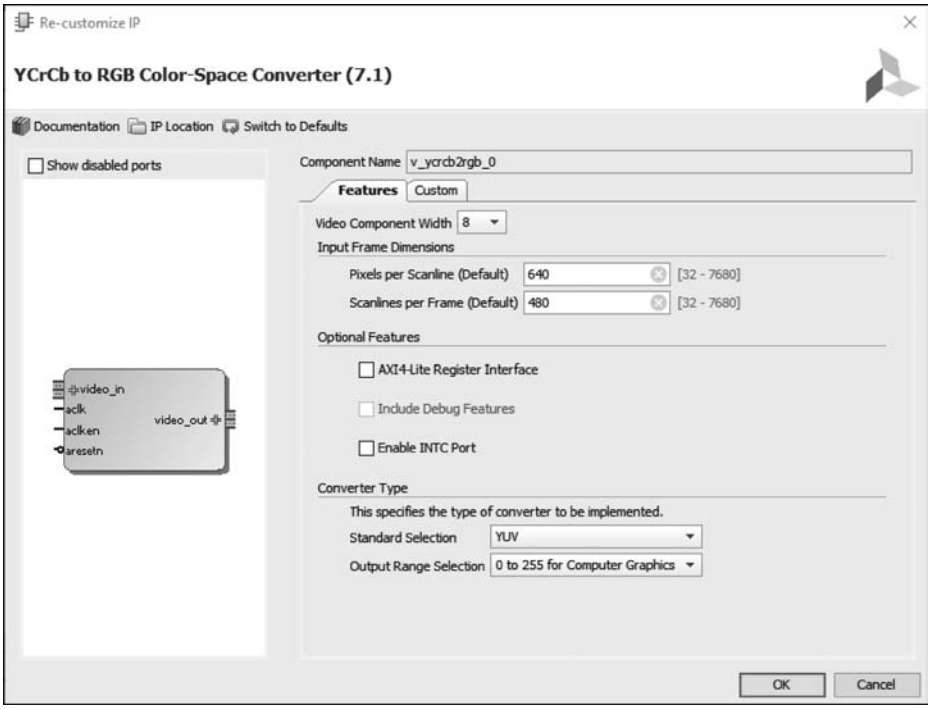


图 5.61　YCrCb to RGB IP 核 Features 配置页面

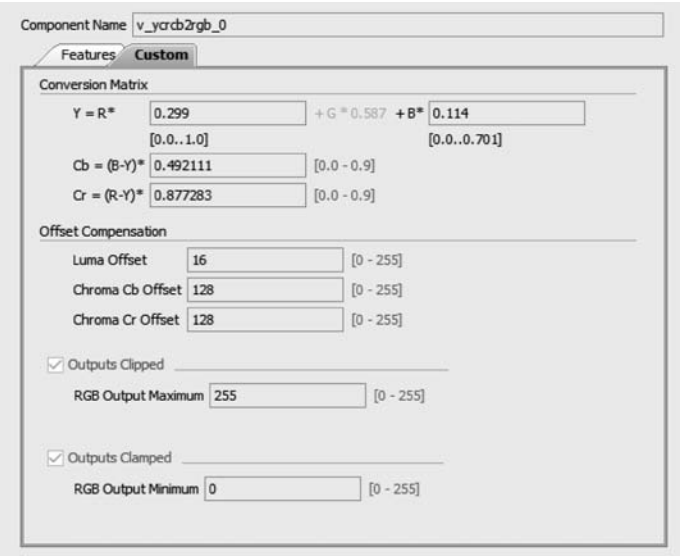


图 5.62　YCrCb to RGB IP 核 Custom 配置页面

5.7　坏点校正的 FPGA 实现

5.7.1　FPGA 功能概述

如图 5.63 所示，这是整个视频采集系统的功能框图。MT9V034 传感器默认的寄存器配置即可输出正常的视频流，FPGA 通过对其同步信号，如时钟、行频和场频进行检测，从而从数据总线上实时地采集图像数据。

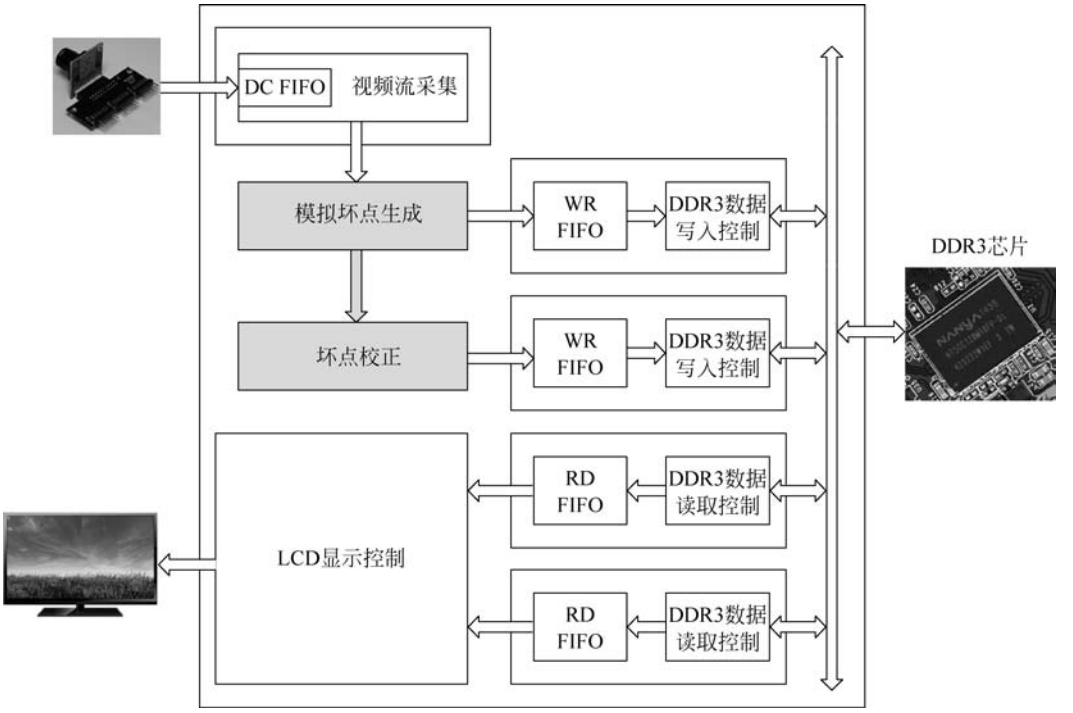


图 5.63　MT9V034 坏点校正工程功能框图

在 FPGA 内部，采集到的视频数据先通过一个 FIFO，将原本时钟域为 25MHz 下同步的数据流转换到 50MHz 下。接着视频数据流会做模拟坏点的生成，生成的视频数据流既会被直接写入 DDR3 中，也会被送往下一个模块做坏点校正，坏点校正后的视频数据流最后也写入 DDR3 的另一片地址空间中。与此同时，VGA 显示驱动模块产生 DDR3 数据读取的时序，将 DDR3 缓存的两组图像数据读出，并送往 VGA 显示器进行显示。在 VGA 显示器上，左侧的视频图像是模拟坏点后的图像，在图像上会有多个明显的黑色坏点；右侧的视频图像则是做了坏点校正的图像，黑色坏点已经消失。

5.7.2 FPGA 设计说明

at7_img_ex09 工程源码的层次结构如图 5.64 所示。

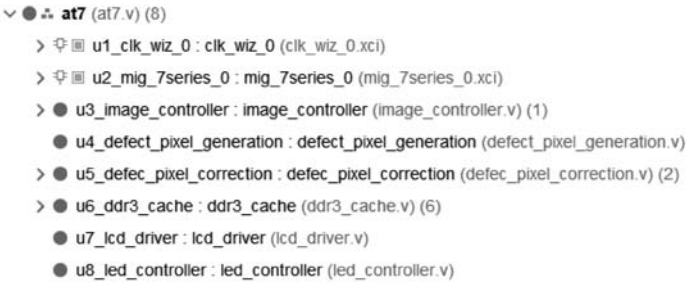


图 5.64 at7_img_ex09 工程源码层次结构

at7_img_ex09 工程模块及功能描述如表 5.9 所示。

表 5.9 at7_img_ex09 工程模块及功能描述

模 块 名 称	功 能 描 述
clk_wiz_0	该模块是 PLL IP 核的例化模块，该 PLL 用于产生系统中所需要的不同频率时钟信号
mig_7series_0	该模块是 DDR3 控制器 IP 核的例化模块。FPGA 内部逻辑读写访问 DDR3 都是通过该模块实现，该模块对外直接控制 DDR3 芯片
image_controller	该模块及其子模块实现 MT9V034 输出图像的采集控制等。image_capture 模块实现图像采集功能
ddr3_cache	该模块主要用于缓存读或写 DDR3 的数据，其下例化了两个 FIFO。该模块连接 FPGA 内部逻辑与 DDR3 IP 核(mig_7series_0 模块)之间的数据交互
defect_pixel_generation	该模块对输入的视频流固定的几个像素点坐标做特殊处理，使其成为黑色坏点
defec_pixel_correction	该模块使用邻近像素点求平均的算法对坏点坐标的像素做坏点校正处理，使输出图像的坏点消除
lcd_driver	该模块驱动 VGA 显示器，同时产生读取 DDR3 中图像数据的控制逻辑
led_controller	该模块控制 LED 闪烁，指示工作状态

1. 坏点生成

defect_pixel_generation.v 模块将原始采集到的 MT9V034 视频流人为地加入 4 个坏点，这 4 个坏点始终输出数据 0x00(黑色)，坏点坐标分别为(5,5)、(5,8)、(8,10)、(8,12)。

如图 5.65 所示，defect_pixel_generation.v 模块中，根据输入的图像同步信号(如复位信号、数据有效信号和行结束信号)进行每行的像素计数和数据行的计数，以此判断当前输出像素坐标是否为模拟产生坏点的坐标。对于模拟产生坏点的坐标，对应的输出像素数据为 0x00，即始终为黑色；而非坏点坐标的像素数据，则保持正常的视频图像数据。与此同时，由

于坏点像素的产生使得数据的输出相比输入有一个时钟周期的延时，所以在这个模块对应输出的图像同步信号（如复位信号、数据有效信号和行结束信号）都延时一个时钟周期后输出。

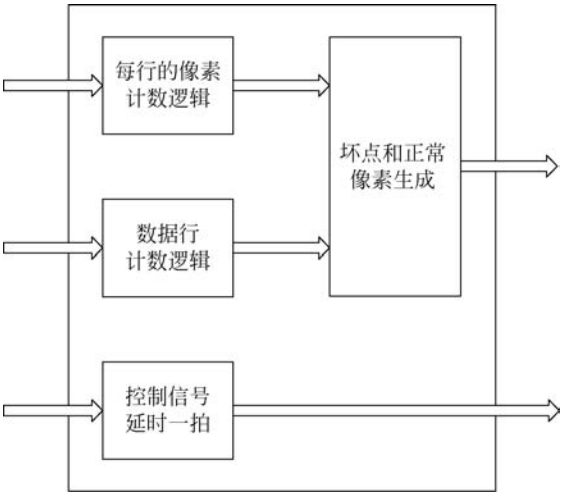


图 5.65　defect_pixel_generation.v 模块功能框图

这个模拟坏点产生的模块，只是为了模拟图像传感器的坏点。在实际的工程应用中，当然不需要也一定不会有这个模拟坏点的模块。我们这个工程实例中，坏点是人为加入的，那么坏点的坐标也都是事先设定好的，因此在后面的坏点校正模块中，我们也知道需要进行坏点校正的像素坐标，直接找到这个坐标进行处理即可。但在实际工程应用中，坏点是图像传感器本身随机存在的，传感器本身也不会携带任何坏点坐标信息，我们如何能判定并且获取坏点坐标信息呢？一般情况下，这类图像传感器在焊接到电路板制成成品后，都会经过一道图像校准的工序，这道工序的原理基本是这样的：让图像传感器拍摄标准的白板和黑板，将对应的图像数据进行后处理，以此检测并判定是否有坏点存在，并且标定坏点坐标，然后将这些坐标信息传递给产品。在产品初始化时，可以加载这些标定好的坏点坐标信息并加以使用，比如使用 FPGA 做图像采集和坏点校正，那么每个特定的产品都会有一组特定的坏点坐标信息保存在本地，在 FPGA 初始化时这组坏点坐标信息被加载，接着 FPGA 就使用这组坏点坐标信息进行相应的坏点校正。

2. 坏点校正

defec_pixel_correction.v 模块实现坏点校正功能，对模拟产生的 4 个坏点进行邻近像素点的均值填充。在该模块的设计中，若坏点像素处于边缘（即第 1 行、最后 1 行、第 1 列或最后 1 列），则不做处理。此外，对坏点像素，使用其周围像素点做如图 5.66 所示的运算（取上、下、左、右 4 个点做平均）。

例如，1～8 像素是(x,y)点周围邻近的 8 个像素点。若(x,y)这个点不处于边缘像素，且该像素是坏点，那么就用它左(8)、右(4)、上(2)、下(6)这 4 个像素点求平均值替代原来的(x,y)点。

1	2	3
8	(x, y)	4
7	6	5

$$1/4 \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

图 5.66　坏点像素值计算

该模块功能框图如图 5.67 所示，使用 2 个 FIFO 分别缓存前后行，即进入图像处理的 3 组数据流分别是第 $n-1$ 行、第 n 行和第 $n+1$ 行的图像，控制输入数据流和 2 个 FIFO 缓存的图像在同一个位置。此外，对前后 2 个像素的图像值进行缓存，这样便可实现坏点像素坐标以及前后列、上下行之间数据的同步，以此就能获取坏点像素坐标的邻近像素的平均值。通过对几个同步信号的判断，进行每行的像素计数和数据行的计数，以此判断当前坐标是否为坏点坐标，对坏点坐标输出的像素数据赋邻近像素值的平均值，非坏点坐标赋原值输出。

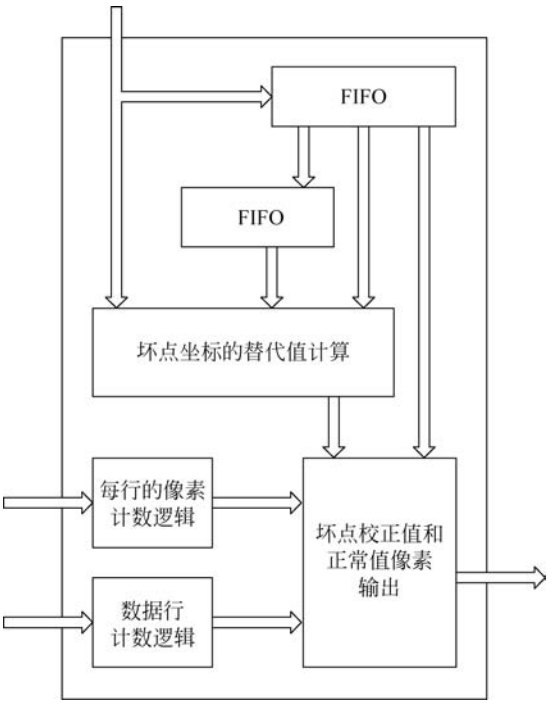


图 5.67　defec_pixel_correction. v 功能框图

5.7.3　FPGA 板级调试

连接好 MT9V034 摄像头模块、VGA 模块和 FPGA 开发板，同时连接好 FPGA 的下载器并给板子供电。

使用 Vivado 2019.1 打开工程 at7_img_ex09，将 at7_img_ex09\at7.runs\impl_1 文件夹下的 at7.bit 文件烧录到板子上，可以看到 VGA 显示器同时显示左右两幅图像，左侧图像为包含模拟黑色坏点的原始图像，将摄像头对准白色背景时，可以看到图像左上角有多个黑色坏点；右侧图像对坏点做了校正处理，图像看上去和正常没有坏点的效果几乎一样的。

5.8　图像直方图统计与实时显示的 FPGA 实现

5.8.1　FPGA 系统概述

如图 5.68 所示，这是整个视频采集系统的功能框图。MT9V034 传感器默认的寄存器配置即可输出正常的视频流，FPGA 通过对其同步信号，如时钟、行频和场频进行检测，从而从数据总线上实时地采集图像数据。

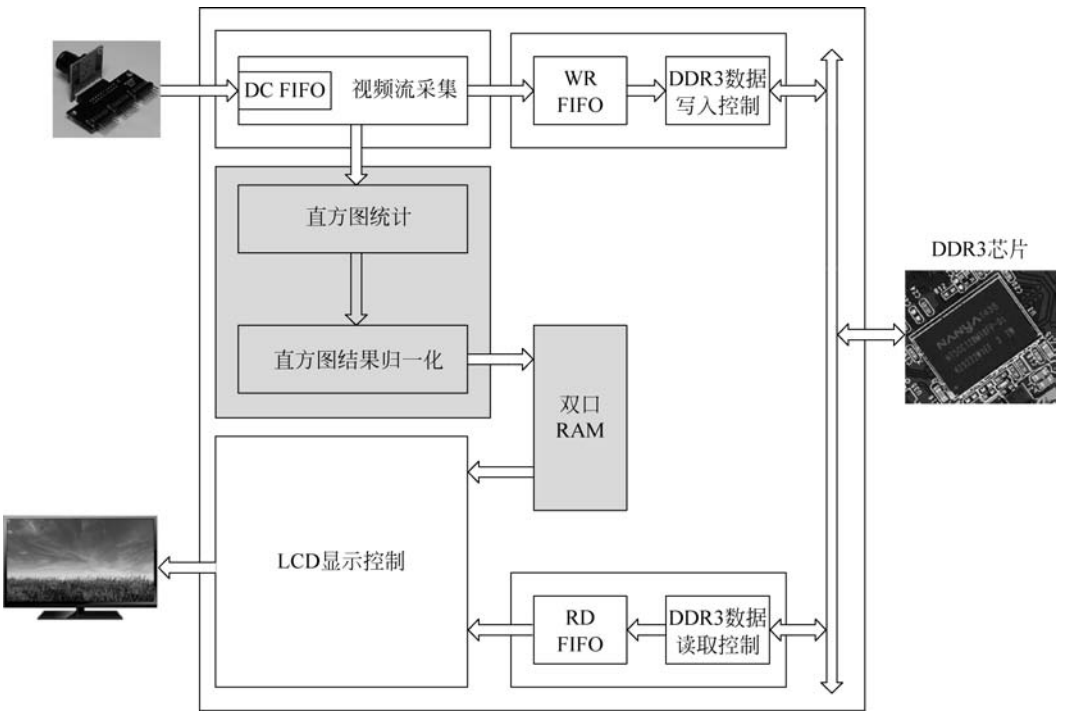


图 5.68　MT9V034 直方图显示工程功能框图

在 FPGA 内部，采集到的视频数据先通过一个 FIFO，将原本时钟域为 25MHz 下同步的数据流转换到 50MHz 下。接着将这个数据再送入写 DDR3 缓存的异步 FIFO 中，这个 FIFO 中的数据一旦达到一定数量，就会写入 DDR3 中。与此同时，读取 DDR3 中缓存的图像数据，缓存到 FIFO 中，并最终送往 VGA 显示驱动模块进行显示。VGA 显示驱动模块不断地发出读图像数据的请求，并驱动 VGA 显示器显示视频图像。

本实例除了前面提到的对原始图像做 DDR3 缓存和显示，还会在原始图像缓存到 DDR3 之前，对当前图像做直方图统计（以帧为单位做统计），统计后的直方图结果做归一化处理，便于后续 VGA 显示器显示的直方图绘制，归一化的直方图结果取值范围是 0～448，用 256 个 10 位数据表示，存入双口 RAM 中。根据 VGA 显示模块的请求，从双口 RAM 读取实时图像的归一化直方图统计结果在 VGA 显示器右侧绘制直方图。最终在 VGA 液晶显示器上，可以看到左侧图像是原始的图像，右侧图像是经过归一化处理的直方图图像。

5.8.2 FPGA 设计说明

at7_img_ex15 工程源码的层次结构如图 5.69 所示。

图 5.69　at7_img_ex15 工程源码层次结构

各个模块及功能描述如表 5.10 所示。

表 5.10　at7_img_ex15 工程模块及功能描述

模块名称	功能描述
clk_wiz_0	该模块是 PLL IP 核的例化模块，该 PLL 用于产生系统中所需要的不同频率时钟信号
mig_7series_0	该模块是 DDR3 控制器 IP 核的例化模块。FPGA 内部逻辑读写访问 DDR3 都是通过该模块实现，该模块对外直接控制 DDR3 芯片
image_controller	该模块及其子模块实现 MT9V034 输出图像的采集控制等。image_capture 模块实现图像采集功能
histogram_calculation	该模块对每帧输入的原始图像做 256 级的直方图统计，并对统计结果进行归一化处理
dual_ram_cache	该模块对直方图统计并归一化后的结果做缓存，写入双口 RAM 中，同时 LCD 驱动模块产生的读控制信号可以对双口 RAM 做读取控制
ddr3_cache	该模块主要用于缓存读或写 DDR3 的数据，其下例化了两个 FIFO。该模块连接 FPGA 内部逻辑与 DDR3 IP 核（mig_7series_0 模块）之间的数据交互
lcd_driver	该模块驱动 VGA 显示器，同时产生读取 DDR3 中图像数据的控制逻辑
led_controller	该模块控制 LED 闪烁，指示工作状态

工程文件夹 at7_img_ex15\at7. srcs\sources_1\new 下的 histogram_calculation. v 模块实现了图像的直方图统计与归一化处理。该模块有一个包含 6 个状态的状态机，如图 5.70 所示。

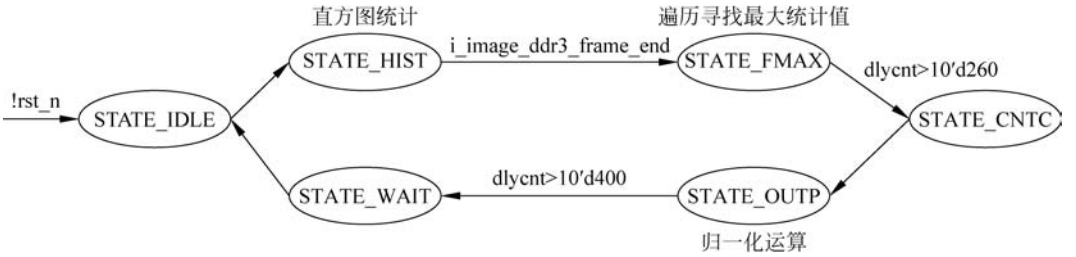


图 5.70　直方图统计状态机图

以这个状态机为主轴的设计思路如下：

(1) 上电初始状态 STATE_IDLE，复位结束后即进入下一状态 STATE_HIST。

(2) STATE_HIST 状态下，进行实时图像的 256 级直方图统计，统计结果存放在寄存器 histogram_cnt[255:0][19:0]中；图像接收信号 i_image_ddr3_frame_end 拉高时，切换到下一个状态 STATE_FMAX。

(3) STATE_FMAX 状态下，遍历一遍直方图统计结果寄存器 histogram_cnt[255:0][19:0]，找出最大值存放在寄存器 max_histogramcnt[19:0]中；找到最大值后，切换到下一状态 STATE_CNTC。

(4) STATE_CNTC 状态下，直接转换到下一个状态 STATE_OUTP。该状态主要为了清零计数器 dlycnt。

(5) STATE _OUTP 状态下，依次将 256 个直方图统计结果乘以 448(=256+128+64)，作为被除数，实际乘以 448 是通过 3 个移位结果(分别对统计结果左移 8 位、左移 7 位和左移 6 位)进行累加实现。而 max_histogramcnt[19:0]则作为除数，依次输出 256 个进行除法归一化后的直方图统计结果(o_image_hc_wren 拉高时 o_image_hc_wrdb[9:0]有效)。完成后进入下一状态 STATE_WAIT。

(6) STATE_WAIT 状态下，直接切换到 STATE_IDLE。

在第(5)步进行的归一化处理，其基本思想是找到 256 个直方图统计结果的最大值，作为归一化的 1(其他值都小于 1)；而其他结果都会以此为标准获取对应的归一化值；例如最大值若为 40000，那么归一化后为 1，某个统计结果是 1000，那么归一化后是 0.025；而实际我们需要将这个归一化后的直方图结果显示到 VGA 显示器上，VGA 显示器上希望最高的直方图可以取 448 像素来显示，那么用 448 乘以归一化后的结果即可。

实际 VGA 显示器是 720p 的驱动分辨率，最大可以给到 720 像素的高度，但是因为左侧的原始采集图像显示是 640×480 像素，为了显示美观，最好给出一个不超过 480 像素的最高直方图高度显示，而取 448 其实是考虑到它等于 256+128+64，可以不消耗 FPGA 的乘法器资源，用移位累加来实现。

5.8.3 MATLAB 与 FPGA 协同仿真说明

1. 直方图统计与归一化结果仿真

在 at7_img_ex15\at7.srcs\sources_1\new\testbench 文件夹下，测试脚本 sim_histogram_calculation.v 用于对模块 histogram_calculation.v 进行仿真。

Vivado 打开 at7_img_ex15 工程，在 Sources 面板中，展开 Simulation Sources→sim_1，将 sim_histogram_calculation.v 文件设置为 top module。在 Flow Navigator 面板中，展开 Simulation，单击 Run Simulation，在弹出的菜单中单击 Run Behavioral Simulation 进行仿真。

测试脚本中，读取 at7_img_ex15\at7.sim 文件夹下的 640×480 像素图像数据 image_in_hex.txt（该文件由 at7_img_ex15\matlab 文件夹下的 image_txt_generation.m 产生，原始图像为 test.bmp）。一组完整的图像数据经过 histogram_calculation.v 模块处理后，产生 256 个归一化直方图结果，写入 histogram_result.txt 文本中（仿真测试结果位于 project\at7_img_ex15\at7.sim\sim_1\behav 文件夹下）。

使用 at7_img_ex15\matlab 文件夹下的 draw_histogram_from_FPGA_result.m 脚本，可以同时比对 MATLAB 和 FPGA 统计的直方图输出结果如图 5.71 所示。由于 FPGA 统计结果是一个归一化结果，所以和 MATLAB 实际统计结果的数值可能不一样，但是从比对图上可以看出，它们的趋势和分布完全一致。

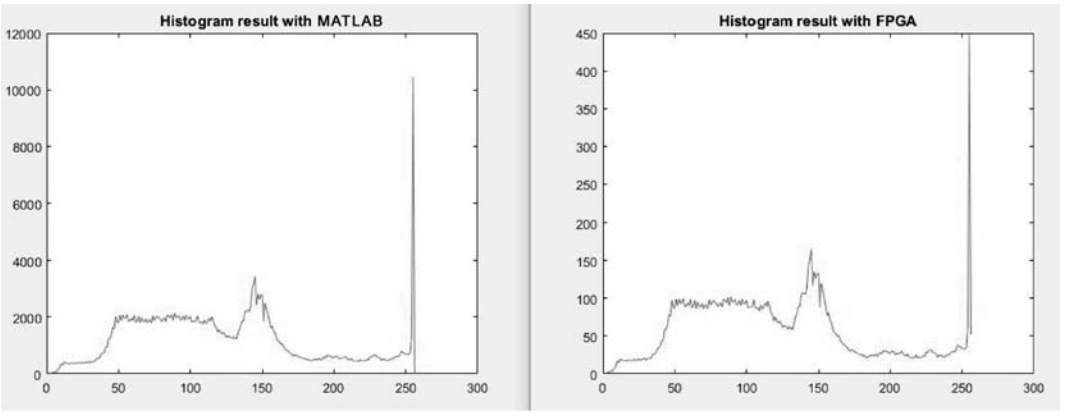


图 5.71 MATLAB 和 FPGA 统计的直方图比对

2. 图像与直方图显示结果仿真

在 at7_img_ex15\at7.srcs\sources_1\new\testbench 文件夹下，测试脚本 sim_at7.v 用于对模块 histogram_calculation.v、dual_ram_cache.v 和 lcd_driver.v 进行仿真。

Vivado 打开 at7_img_ex15 工程，在 Sources 面板中，展开 Simulation Sources→sim_1，将 sim_zstar.v 文件设置为 top module。在 Flow Navigator 面板中，展开 Simulation，单击 Run Simulation，在弹出的菜单中单击 Run Behavioral Simulation 进行仿真。

测试脚本中，读取 at7_img_ex15\at7.sim 文件夹下的 640×480 像素图像数据 image_in_hex.txt（该文件由 at7_img_ex15\matlab 文件夹下的 image_txt_generation.m 产生，原始图像为 test.bmp）。一组完整的图像数据经过 histogram_calculation.v 模块处理后，产生 256 个归一化直方图结果，缓存到 dual_ram_cache.v 模块的双口 RAM 中，lcd_driver.v 模块根据显示驱动需要读取双口 RAM 中的数据，将直方图显示在液晶屏的右侧。测试脚本中，根据 lcd_driver.v 模块的显示驱动信号，将一帧的显示图像写入 monitor_display_image.txt 文本中（仿真测试结果位于 project\at7_img_ex15\at7.sim\sim_1\behav 文件夹下）。

使用 at7_img_ex15\matlab 文件夹下的 draw_image_from_FPGA.m 脚本，可以打印 monitor_display_image.txt 文本中输出的图像。如图 5.72 所示，就是最终的 VGA 显示器中将会显示的界面示意图，左侧是原始图像，右侧是其直方图分布。可以看到，这个直方图分布情况和前面 MATLAB 计算出来的结果也是一致的。

图 5.72　MATLAB 对 VGA 显示器效果的模拟

5.8.4　FPGA 板级调试

连接好 MT9V034 摄像头模块、VGA 模块和 FPGA 开发板，同时连接好 FPGA 的下载器并给板子供电。

使用 Vivado 2019.1 打开工程 at7_img_ex15，将 at7_img_ex15\at7.runs\impl_1 文件夹下的 at7.bit 文件烧录到板子上，如图 5.73 所示，可以看到 VGA 显示器同时显示左右两个图像，左侧图像为原始图像，右侧图像为直方图。

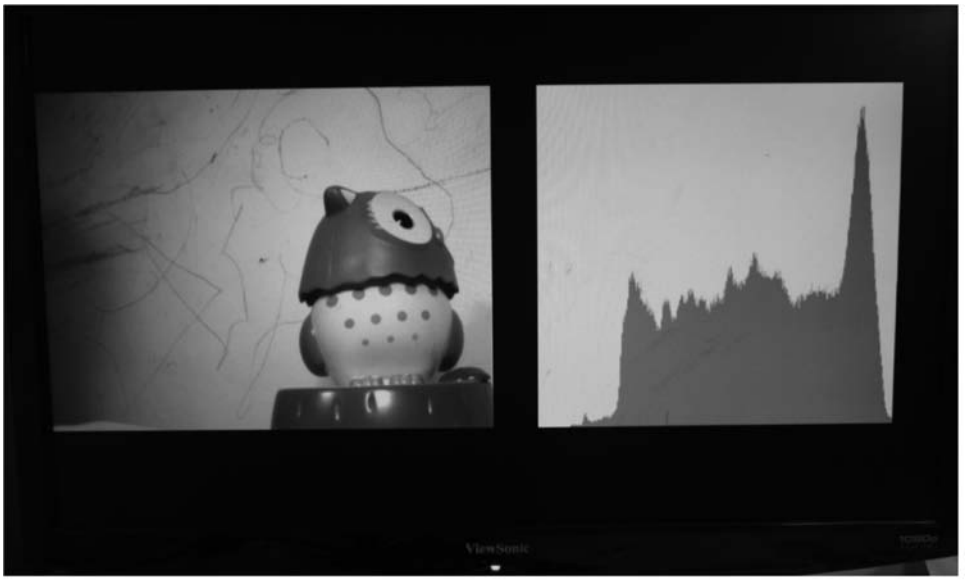


图 5.73　直方图实时显示效果图