

**Imagination大学计划**

**RVfpga实验5**

**图像处理：C语言和汇编语言**

# 简介

# 在本实验中，您将编译用于执行图像处理程序的RISC-V编程项目。这些项目将包含多个源文件，其中一些用C语言编写，另一些用汇编语言编写。我们将展示C函数与汇编程序之间如何相互调用。

# 图像处理教程

本实验首先检查随附的RGB图像（图1的左侧）处理程序，然后生成对应的灰度图像（图1的右侧）。该程序用C语言和RISC-V汇编语言编写，经配置后在PlatformIO环境中运行。可从以下位置获取该程序：

*[RVfpgaPath]/RVfpga/Labs/Lab5/ImageProcessing*

源代码位于*src*子目录。



图1. 将RGB图像转换为灰度图像。

1. **项目结构及*main*函数**

该程序由以下源文件组成：**main.c**/**VanGogh\_128.c**和**assemblySubroutines.S**。*c*文件包含函数（例如，用于执行图像转换的函数）和变量声明（例如输入图像，声明为无符号char数组）。**assemblySubroutines.S**文件包含函数的汇编语言实现，用于将rgb图像转换为灰度图像，称为：*ColourToGrey\_Pixel*。

图2所示为本项目的main函数。该函数先调用函数*initColourImage*（将使用输入图像数据创建一个N x M矩阵），再将彩色图像转换为灰度图像（函数*ColourToGrey*），最后输出一条消息并进入无限循环（while (1);）。

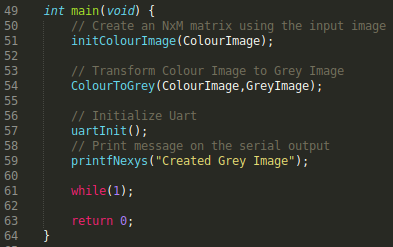
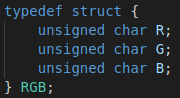


图2. 图像处理项目中的*main*函数

1. **RGB图像和灰度图像**

图像由像素矩阵组成，其中矩阵的每个元素代表特定比例下的像素值。在RGB中，每个像素由三个值组成，分别对应于红色（**R**）、绿色（**G**）和蓝色（**B**）分量的发光强度。因此，彩色图像的每个像素将是一个三分量向量。在本项目中，RGB像素类型使用以下定义：



这段代码定义了一个名为*RGB*的结构。在C中，struct数据类型是由单个名称指定的变量（可能是不同类型）的集合。该结构包含三个相同类型（unsigned char）的字段，名为*R*、*G*和*B*。每个颜色通道由8位表示，以便区分每个颜色通道中256个不同的强度级别，每个像素总共24位（24 bpp）。这是当前比较常见的一种数字图像处理格式。

为了表示灰度图像，我们使用0到255范围的单个值（单通道）指示每个像素的亮度。在本图像处理项目中，我们使用二维字符数组表示灰度图像：



1. **将彩色图像转换为灰度图像**

使用以下加权求和公式来执行两个颜色空间（RGB和灰度）之间的转换：

灰色 = 0.299\*R + 0.587\*G + 0.114\*B

该公式基于<https://www.mathworks.com/help/matlab/ref/rgb2gray.html>中描述的算法。

对于每个像素，我们通过将每个颜色通道乘以公式中给出的权重来计算灰度值。权重的总和（0.299+0.587+0.114）为1，因此所得的灰度值将在0-255的范围内，可以用单个字节表示。

为了使用公式中给出的权重，需要使用实数进行运算，但是**SweRV EH1**处理器不支持浮点。一种方法是使用浮点仿真（如《入门指南》第5.H部分所示的DotProduct程序），但本实验中使用的是基于整数算术的方法。权重将转换为整数，并且权重的总和为2的幂（本例中为210）。为了将权重转换为整数，需要将每个浮点权重乘以210并四舍五入为最接近的整数：

* 0.299\*210 = 306.176 ≈ **306 （R的权重）**
* 0.587\*210 = 601.088 ≈ **601 （G的权重）**
* 0.114\*210 = 116.736 ≈ **117 （B的权重）**

当然，要将最终灰度值减小到0-255范围，必须将总和除以210（只需将值右移10位即可轻松完成）。因此，可以使用以下公式实现最终转换：

灰色 = (306\*R + 601\*G + 117\*B) >> 10

请注意，假设常量的总和（306+601+117）为1024，则所得的灰度值仍将在0-255范围内。

图3给出了*ColourToGrey*函数（左侧）和*ColourToGrey*调用的*ColourToGrey\_Pixel*子程序（右侧）的代码。

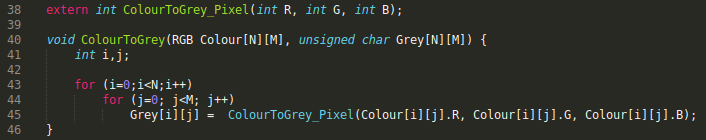
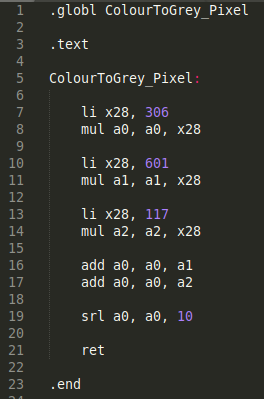
 

图3. *ColourToGrey*函数（在*main.c*文件中实现）和*ColourToGrey\_Pixel*子程序  
（在*assemblySubroutines.S*文件中实现）

在汇编语言中，符号（变量和函数/子程序）默认情况下是局部的，即，对其他文件不可见。要将这些局部符号转换为全局符号，必须使用.globl汇编器伪指令将其导出。图3右侧的第一行（.globl ColourToGrey\_Pixel）用于导出*ColourToGrey\_Pixel*函数，以便其可供另一文件（*main.c*）中的*ColourToGrey*函数使用。图3左侧的第一行（extern int ColourToGrey\_  
Pixel(int R, int G, int B)）用于将*ColourToGrey\_Pixel*函数声明为该文件的外部函数。

1. **程序执行与结果可视化**

在格雷码转换完成之后，可以在程序执行结束之前将一些存储区域的内容转储到文件中。为此，我们将使用GDB调试器的dump命令。要运行项目代码并获取图像结果，请按以下步骤操作：

1. 打开VSCode和PlatformIO。
2. 在顶部菜单栏上，单击“*File*”（文件）→“*Open Folder*”（打开文件夹），然后导航至目录*[RVfpgaPath]/RVfpga/Labs/Lab5*。选择目录*ImageProcessing*（不要打开，只需将其选中），然后单击窗口顶部的“OK”（确定）。PlatformIO现在将打开项目。
3. 打开platformio.ini并取消注释board\_build.bitstream\_file，然后输入bit文件的目录位置。例如，使用在实验1中创建的bit文件。

board\_build.bitstream\_file = [RVfpgaPath]/RVfpga/Labs/Lab1/Project1/Project1.runs/impl\_1/rvfpganexys.bit

1. 打开src目录下的所有源文件（*main.c*和*assemblySubroutines.S*）并对其进行分析，以便您清楚地了解程序的工作方式。
2. 单击左侧功能区菜单中的PlatformIO图标，展开“Project Tasks”（项目任务）→ env:swervolf\_nexys →“Platform”（平台），然后单击“Upload Bitstream”（上传比特流），将RVfpgaNexys下载到Nexys A7开发板。请记住，也可以使用Verilator和Whisper在仿真中运行这些程序。
3. 在PlatformIO中执行程序。您可以在开发板上进行操作（在这种情况下，必须先按照上一步的操作将RVfpgaNexys上传到Nexys A7），也可以使用Whisper仿真器进行操作（如“RVfpga入门指南”中所述）。无论采用哪种方式，都应先单击PlatformIO左侧栏中的“Run”（运行）按钮，然后通过单击播放按钮   
   启动调试器。

程序执行到*main*函数开头处会停止，因此需单击“Continue”（继续）按钮恢复执行。

一小段时间（大约1秒）过后，程序将完成上述灰度图像转换，并将到达末尾的无限循环（while(1);）（见图2）。通过单击*“Pause”*（暂停）按钮   
暂停执行。

1. 通过在“Debug Console”（调试控制台）中运行以下命令（参见图4，其中显示了这两条命令的执行）导出灰度图像（GreyImage）：

cd AdditionalFiles

dump value GreyImage.dat GreyImage

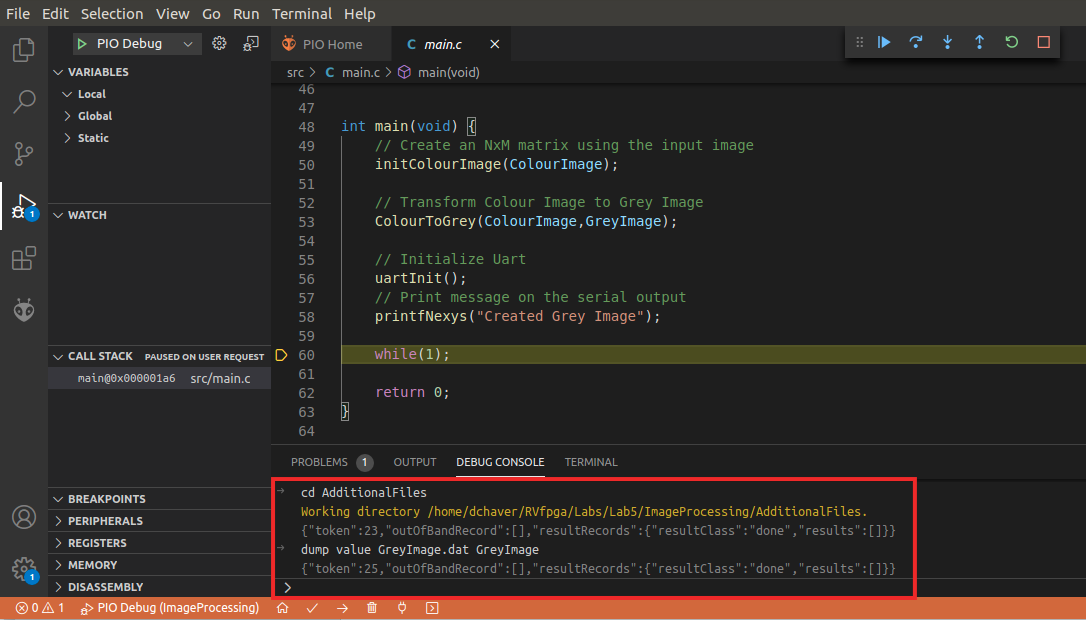


图4. 将灰度图像导出到文件

1. 将.dat文件转换为可在系统中查看的.ppm文件。

在**LINUX中：**通过打开终端并输入以下命令（见图5）来执行此操作：

cd *[RVfpgaPath]/RVfpga/Labs/Lab5/ImageProcessing/AdditionalFiles*

gcc -o dump2ppm dump2ppm.c

./dump2ppm GreyImage.dat GreyImage.ppm 128 128 1

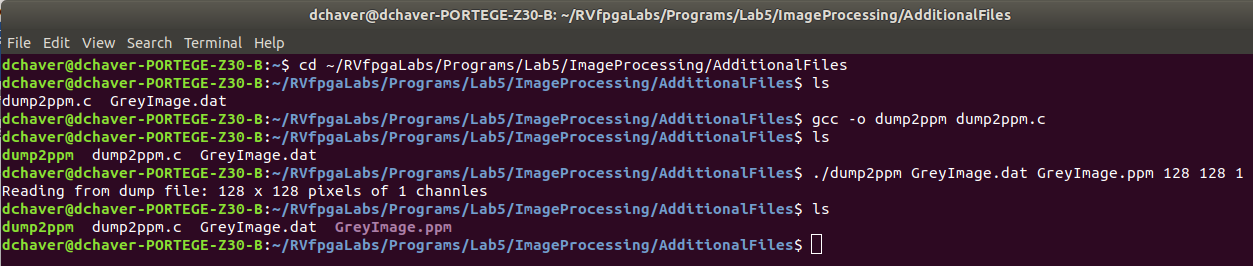


图5. 将图像转换为*.ppm*格式

**在WINDOWS中：**通过以下任一方式来执行此操作：

1. 使用*[RVfpgaPath]\RVfpga\Labs\Lab5\ImageProcessing\AdditionalFiles*中提供的dump2ppm.exe可执行文件。打开命令shell，转到相应文件夹，并使用与上面相同的参数运行可执行文件：

dump2ppm.exe GreyImage.dat GreyImage.ppm 128 128 1

或

2. 使用Cygwin（如果已按照“RVfpga入门指南”中的说明进行安装）来编译dump2ppm.c程序。然后，在Cygwin终端中或命令shell中（如上面的选项1所示）运行程序   
（dump2ppm.exe）。

1. 使用GIMP（GNU图像处理程序）打开.ppm文件。如果尚未安装该程序，请访问以下网站下载安装程序：

https://www.gimp.org/downloads/

灰度图像应与图1右侧显示的图像类似（也可以在   
*[RVfpgaPath]/RVfpga/Labs/Lab5/ImageProcessing/AdditionalFiles/VanGogh\_128.ppm*中访问输入彩色图像，该图像应与图1左侧显示的图像类似）。

# 练习

**练习1.** 对其他输入图像执行程序。可以使用以下位置提供的图像：   
*[RVfpgaPath]/RVfpga/Labs/Lab5/ImageProcessing/src/TheScream\_256.c*（可以在以下位置查看相应的.ppm图像：   
*[RVfpgaPath]/RVfpga/Labs/Lab5/ImageProcessing/AdditionalFiles/TheScream\_256.ppm.*还可以如前文所述通过运行程序dat2ppm来创建该图像。）

**练习2.** 创建一个C函数来统计*VanGogh*灰度图像中接近白色（>235）的元素数量和接近黑色（<20）的元素数量。使用Western Digital的PSP和BSP库在串行控制台上输出两个数字（如实验2第3部分所述）。

**练习3.** 将**ColourToGrey\_Pixel**汇编子程序转换为C函数，然后将C函数**ColourToGrey**转换为用于调用**ColourToGrey\_Pixel** C函数的汇编子程序。

* + 在C语言中，所有函数和全局变量都默认导出为全局符号，因此可以使用子程序*ColourToGrey*中的*ColourToGrey\_Pixel*函数。
  + 要使用汇编语言访问矩阵，必须在给定数组起始地址的情况下计算元素(*i*,*j*)的地址。根据ANSI C标准，二维数组按行存储在存储器中。因此，可通过将数组的起始地址与偏移量*(i\*M + j)\*B*相加来计算*i*行、*j*列像素的地址，其中*M*是列数，*B*是每个像素占用的字节数：RGB图像中为三个字节，灰度图像中仅为一个字节。

**练习4.** 将“**Blur Filter**”（模糊滤镜）应用于*VanGogh*彩色图像（您可以在网上找到许多信息；例如，您可以使用以下网站提供的信息：[https://lodev.org/cgtutor/filtering.html#Find\_Edges](https://lodev.org/cgtutor/filtering.html" \l "Find_Edges_)）。

请注意，要将.dat图像转换为.ppm图像，必须对dump2ppm命令调用进行一些修改，考虑使用3个通道而不是仅使用1个通道：

./dump2ppm FilterColourImage.dat FilterColourImage.ppm 128 128 **3**

此外，还可以将过滤后的图像与原始图像进行比较，原始图像位于以下位置：   
*[RVfpgaLabsPath]/RVfpgaLabs/Programs/Lab5/ImageProcessing/AdditionalFiles/VanGogh\_128.ppm*