



IMAGINATION大學計劃

RVfpga實驗5

影像處理：C語言和組合語言

1. 簡介

在本實驗中，您將編譯用於執行影像處理常式的RISC-V程式設計專案。這些專案將包含多個原始程式檔，其中一些用C語言編寫，另一些用組合語言編寫。我們將展示C函數與組合語言常式之間如何相互呼叫。

2. 影像處理教程

本實驗首先檢查隨附的RGB圖像（圖1的左側）處理程式，然後產生對應的灰階圖像（圖1的右側）。該程式用C語言和RISC-V組合語言編寫，經配置後在PlatformIO環境中執行。可從以下位置獲取該程式：

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

原始程式碼位於src子目錄。

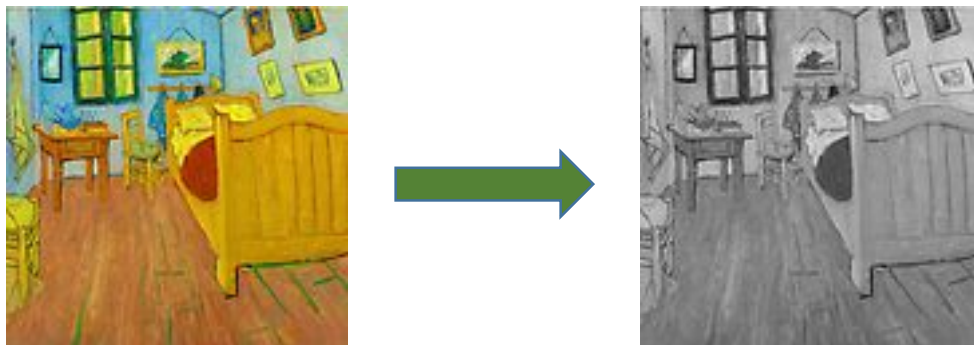


圖1. 將RGB圖像轉換為灰階圖像。

A. 專案結構及main函數

該程式由以下來源檔案組成：**main.c**、**VanGogh_128.c**和**assemblySubroutines.S**。**.c**檔案包含函數（例如，用於執行圖像轉換的函數）和變數宣告（例如輸入圖像，宣告為不帶正負號char陣列）。**assemblySubroutines.S**檔包含函數的組合語言實作，用於將rgb圖像轉換為灰階圖像，稱為：**ColourToGrey_Pixel**。

圖2所示為本專案的main函數。該函數先呼叫函數**initColourImage**（將使用輸入圖像資料建立一個N x M矩陣），再將彩色圖像轉換為灰階圖像（函數**ColourToGrey**），最後輸出一則訊息並進入無限迴圈（`while (1);`）。

```

49  int main(void) {
50      // Create an NxM matrix using the input image
51      initColourImage(ColourImage);
52
53      // Transform Colour Image to Grey Image
54      ColourToGrey(ColourImage, GreyImage);
55
56      // Initialize Uart
57      uartInit();
58      // Print message on the serial output
59      printfNexys("Created Grey Image");
60
61      while(1);
62
63      return 0;
64  }

```

圖2. ImageProcessing專案中的main函數

B. RGB圖像和灰階圖像

圖像由像素矩陣組成，其中矩陣的每個元素代表特定比例下的像素值。在RGB中，每個像素由三個值組成，分別對應於紅色（**R**）、綠色（**G**）和藍色（**B**）元件的發光強度。因此，彩色圖像的每個像素將是一個三元組向量。在本專案中，RGB像素類型使用以下定義：

```

typedef struct {
    unsigned char R;
    unsigned char G;
    unsigned char B;
} RGB;

```

這段程式碼定義了一個名為RGB的結構。在C中，struct資料類型是由單一名稱指定的變數（可能是不同類型）所組成的集合。該結構包含三個相同類型（unsigned char）的欄位，名為R、G和B。每個顏色通道由8位元表示，以便區分每個顏色通道中256個不同的強度級別，每個像素總共24位元（24 bpp）。這是目前比較常見的一種數位影像處理格式。

為了表示灰階圖像，我們使用0到255範圍的單一值（單通道）指示每個像素的亮度。在本影像處理專案中，我們使用二維字元陣列表示灰階圖像：

```
unsigned char GreyImage[N][M];
```

C. 將彩色圖像轉換為灰階圖像

使用以下加權求和公式來執行兩個顏色空間（RGB和灰階）之間的轉換：

$$\text{grey} = 0.299 * R + 0.587 * G + 0.114 * B$$

該公式基於<https://www.mathworks.com/help/matlab/ref/rgb2gray.html>中描述的演算法。

對於每個像素，我們通過將每個顏色通道乘以公式中提供的權重來計算灰階值。權重的總和（ $0.299+0.587+0.114$ ）為1，因此所得的灰階值將在0-255的範圍內，可以用單一位元組表示。

為了使用公式中提供的權重，需要使用實數進行運算，但是SweRV EH1處理器不支援浮點數。一種方法是使用浮點數模擬（如《入門指南》第5.H節分所示的DotProduct程式），但本實驗中使用的是基於整數算術的方法。權重將轉換為整數，且權重的總和為2的乘幂（本例中為 2^{10} ）。為了將權重轉換為整數，需要將每個浮點權重乘以 2^{10} 並四捨五入為最接近的整數：

- $0.299 \times 2^{10} = 306.176 \approx 306$ （R的權重）
- $0.587 \times 2^{10} = 601.088 \approx 601$ （G的權重）
- $0.114 \times 2^{10} = 116.736 \approx 117$ （B的權重）

當然，要將最終灰階值減小到0-255範圍，必須將總和除以 2^{10} （只需將值右移10位即可輕鬆完成）。因此，可以使用以下公式實現最終轉換：

$$\text{grey} = (306 \times R + 601 \times G + 117 \times B) \gg 10$$

請注意，假設常數的總和（ $306+601+117$ ）為1024，則所得的灰階值仍將在0-255範圍內。

圖3顯示了ColourToGrey函數（左側）和ColourToGrey呼叫的ColourToGrey_Pixel子常式（右側）的程式碼。



```

38  extern int ColourToGrey_Pixel(int R, int G, int B);
39
40  void ColourToGrey(RGB Colour[N][M], unsigned char Grey[N][M]) {
41      int i,j;
42
43      for (i=0;i<N;i++)
44          for (j=0;j<M;j++)
45              Grey[i][j] = ColourToGrey_Pixel(Colour[i][j].R, Colour[i][j].G, Colour[i][j].B);
46  }
    
```

```

1  .globl ColourToGrey_Pixel
2
3  .text
4
5  ColourToGrey_Pixel:
6
7      li x28, 306
8      mul a0, a0, x28
9
10     li x28, 601
11     mul a1, a1, x28
12
13     li x28, 117
14     mul a2, a2, x28
15
16     add a0, a0, a1
17     add a0, a0, a2
18
19     srl a0, a0, 10
20
21     ret
22
23 .end
    
```

圖3. ColourToGrey函數（在main.c檔中實作）和ColourToGrey_Pixel子常式（在assemblySubroutines.S檔中實作）

在組合語言中，符號（變數和函數/子常式）預設情況下是區域的，即，對其他檔案不可見。要將這些區域符號轉換為全域符號，必須使用.global組合語言程式碼器虛擬指令將其匯出。圖3右側的第一行（.globl ColourToGrey_Pixel）用於匯出ColourToGrey_Pixel函數，以便其可供另一檔案（main.c）中的ColourToGrey函數使用。圖3左側的第一行（extern int ColourToGrey_Pixel(int R, int G, int B)）用於將ColourToGrey_Pixel函數宣告為該檔案的外部函數。

D. 程式執行與結果可視化

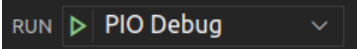
在格雷碼轉換完成之後，可以在程式執行結束之前將一些儲存區域的內容轉儲到檔案中。為此，我們將使用GDB偵錯工具的dump命令。要執行專案程式碼並獲取圖像結果，請按以下步驟操作：


1. 開啟VSCode和PlatformIO。
2. 在頂部功能表列上，按一下「**File**」（檔案）→「**Open Folder**」（開啟資料夾），然後導覽至目錄[RVfpgaPath]/RVfpga/Labs/Lab5。選擇目錄ImageProcessing（不要開啟，只需選擇它），然後按一下視窗頂端的「OK」（確定）。PlatformIO現在將開啟專案。
3. 開啟platformio.ini並取消註釋board_build.bitstream_file，然後輸入bit檔的目錄位置。例如，使用在實驗1中建立的bit檔。

```
board_build.bitstream_file =
[RVfpgaPath]/RVfpga/Labs/Lab1/Project1/Project1.runs/impl_1/rvfpganexys.bit
```

4. 開啟src目錄下的所有原始程式檔（**main.c**和**assemblySubroutines.S**）並對其進行分析，以便清楚瞭解程式的運作方式。
5. 按一下左側功能表功能區中的PlatformIO圖示，展開「**Project Tasks**」（專案任務）→ **env:swervolf_nexys** →「**Platform**」（平台），然後按一下「**Upload Bitstream**」（上傳位元串流），將RVfpgaNexys下載到Nexys A7開發板。請記住，也可以使用Verilator和Whisper在模擬中執行這些程式。
6. 在PlatformIO中執行程式。您可以在開發板上進行操作（在這種情況下，必須先按照上一步的操作將RVfpgaNexys上傳到Nexys A7），也可以使用Whisper模擬器進行操作（如「RVfpga入門指南」中所述）。無論採用哪種方式，都應先按一下PlatformIO

左列中的「**Run**」（執行）按鈕 ，然後通過按一下播放按鈕

 啟動偵錯工具。

程式執行到**main**函數開頭處會停止，因此需按一下「**Continue**」（繼續）按鈕恢復執行 。

一小段時間（大約1秒）過後，程式將完成上述灰階圖像轉換，並將到達末尾的無限迴圈（**while(1);**）（請參閱圖2）。透過按一下「**Pause**」（暫停）按鈕

 暫停執行。

7. 通過在「**Debug Console**」（偵錯控制台）中執行以下命令（請參閱圖4，其中顯示了這兩條命令的執行）匯出灰階圖像（GreyImage）：

```
cd AdditionalFiles

dump value GreyImage.dat GreyImage
```

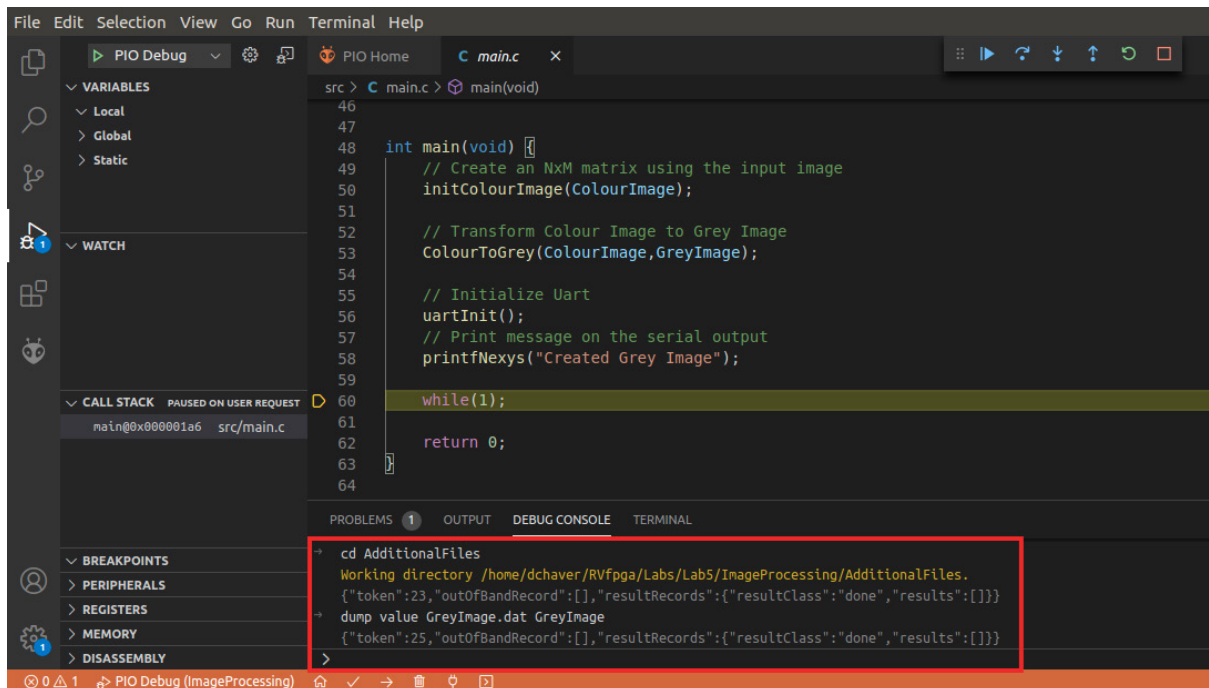


圖4. 將灰階圖像匯出到檔案

8. 將.dat檔案轉換為可在系統中檢視的.ppm檔案。

在**LINUX**中：通過開啟終端機並輸入以下命令（請參閱圖5）來執行此操作：

```
cd [RVfpgaPath]/RVfpga/Labs/Lab5/ImageProcessing/AdditionalFiles
gcc -o dump2ppm dump2ppm.c
./dump2ppm GreyImage.dat GreyImage.ppm 128 128 1
```

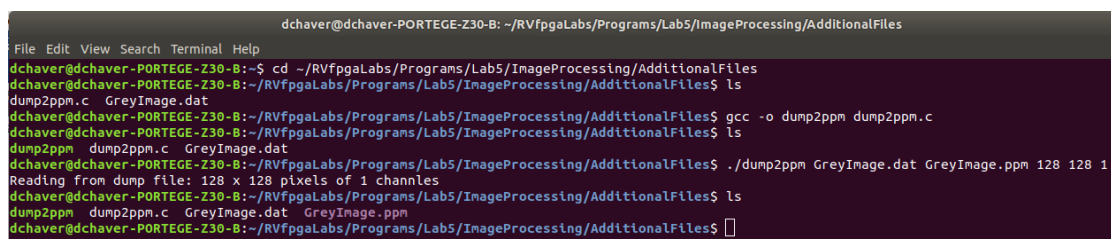


圖5. 將圖像轉換為.ppm格式

在**WINDOWS**中：通過以下任一方式來執行此操作：

1.使用 `[RVfpgaPath]\RVfpga\Labs\Lab5\ImageProcessing\AdditionalFiles` 中提供的 `dump2ppm.exe` 可執行檔開啟命令 `shell`，移至相應資料夾，並使用與上面相同的引數執行可執行檔：

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

或

2.使用 `Cygwin`（如果已按照「`RVfpga` 入門指南」中的說明進行安裝）來編譯 `dump2ppm.c` 程式。然後，在 `Cygwin` 終端機中或命令 `shell` 中（如上面的選項1所示）執行程式（`dump2ppm.exe`）。

9. 使用 `GIMP`（`GNU` 影像處理程式）開啟 `.ppm` 檔案。如果尚未安裝該程式，請造訪以下網站下載安裝程式：

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

灰階圖像應與圖1右側顯示的圖像類似（也可以在 `[RVfpgaPath]\RVfpga\Labs\Lab5\ImageProcessing\AdditionalFiles\VanGogh_128.ppm` 中存取輸入彩色圖像，該圖像應與圖1左側顯示的圖像類似）。

3. 練習

練習1. 對其他輸入圖像執行程式。可以使用以下位置提供的圖像：

`[RVfpgaPath]/RVfpga/Labs/Lab5/ImageProcessing/src/TheScream_256.c`（可以在以下位置檢視相應的.ppm圖像：

`[RVfpgaPath]/RVfpga/Labs/Lab5/ImageProcessing/AdditionalFiles/TheScream_256.ppm`。還可以如前文所述，通過執行程式`dat2ppm`來建立該圖像。）

練習2. 建立一個C函數來統計VanGogh灰階圖像中接近白色（>235）的元素數量和接近黑色（<20）的元素數量。使用Western Digital的PSP和BSP庫在序列控制台上輸出兩個數字（如實驗2第3節所述）。

練習3. 將`ColourToGrey_Pixel`組合語言程式碼子常式轉換為C函數，然後將C函數`ColourToGrey`轉換為用於呼叫`ColourToGrey_Pixel` C函數的組合語言程式碼子常式。

- 在C語言中，會預設將所有函數和全域變數匯出為全域符號，因此可以使用子常式`ColourToGrey`中的`ColourToGrey_Pixel`函數。
- 要使用組合語言存取矩陣，必須在給定陣列起始位址的情況下計算元素 (i,j) 的位址。根據ANSI C標準，二維陣列按列儲存在記憶體中。因此，可通過將陣列的起始位址與偏移量 $(i*M + j)*B$ 相加來計算 i 列、 j 欄像素的位址，其中 M 是欄數， B 是每個像素占用的位元組數：RGB圖像中為三個位元組，灰階圖像中僅為一個位元組。

練習4. 將「Blur Filter」（模糊濾鏡）套用至VanGogh彩色影像（您可以在網路上找到許多資訊；例如，您可以使用以下網站提供的資訊：

https://lodev.org/cgtutor/filtering.html#Find_Edges）。

請注意，要將.dat圖像轉換為.ppm圖像，必須對`dump2ppm`命令呼叫進行一些修改，考慮使用3個通道而不是僅使用1個通道：

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

此外，還可以將過濾後的圖像與原始圖像進行比較，原始圖像位於以下位置：

`[RVfpgaLabsPath]/RVfpgaLabs/Programs/Lab5/ImageProcessing/AdditionalFiles/VanGogh_128.ppm`