

**THE IMAGINATION UNIVERSITY PROGRAMME**

**RVfpga Lab 5**

**이미지 프로세싱: C 및 어셈블리**

# 소개

# 이번 LAB에서는 이미지 처리 루틴을 수행하는 RISC-V 프로그래밍 프로젝트를 빌드합니다. 프로젝트에는 여러 소스 파일이 포함되며, 그중 일부는 C로 작성되고 일부는 어셈블리로 작성됩니다. C 함수가 어셈블리 루틴을 호출하는 방법과 그 반대의 경우를 보여줍니다.

# 이미지 프로세싱 튜토리얼

RGB 이미지 (그림 1의 왼쪽)를 처리하고 해당 이미지의 그레이 스케일 버전을 생성하는 프로그램 (그림 1의 오른쪽)을 검사하면서 이 LAB을 시작합니다. 이 프로그램은 C 및 RISC-V 어셈블리 언어로 작성되었으며 PlatformIO 환경에서 실행되도록 구성되어 있으며, 아래 폴더에 예제가 있습니다.

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

소스 코드는 *src* 하위 디렉토리에 있습니다.



그림 1. RGB 이미지를 그레이 스케일 이미지로 변환.

1. **프로젝트 구조 및 주요 기능**

이 프로그램은 **main.c, VanGogh\_128.c** 및 **assemblySubroutines.S**와 같은 소스 파일로 구성됩니다. .c 파일에는 함수 (예: 이미지 변환을 수행하는 함수)와 변수 선언 (예: 부호 없는 char 배열로 선언된 입력 이미지)이 포함되어 있습니다. **assemblySubroutines.S** 파일에는 이미지를 RGB에서 그레이 스케일로변환하는 함수, 어셈블리로 구성된 *ColourToGrey\_Pixel*이 포함되어 있습니다.

그림 2는 해당 프로젝트의 main 기능을 보여줍니다. 먼저 입력 이미지 데이터로 N x M 행렬을 만드는 *initColourImage* 함수를 호출합니다. 그런 다음 컬러 이미지를 그레이 스케일 이미지로 (function *ColourToGrey*) 변환합니다. 마지막으로 이 함수는 메시지를 인쇄하고 무한 루프 (while (1);)에 들어갑니다.

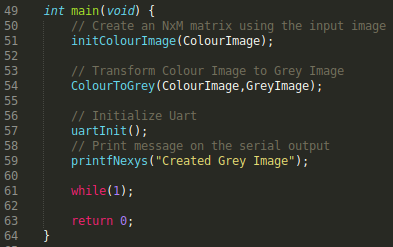
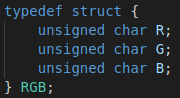


그림 2. 이미지 처리 프로젝트의 *main* function

1. **RGB 및 그레이 스케일 이미지**

이미지는 픽셀 행렬로 구성되며, 행렬의 각 요소는 특정 배율로 픽셀 값을 나타냅니다. RGB에서 각 픽셀은 빨간색 (**R**), 녹색 (**G**) 및 파란색 (**B**) 구성 요소의 광도에 해당하는 세 가지 값으로 구성됩니다. 따라서 컬러 이미지의 각 픽셀은 3 성분 벡터가 됩니다. 이 프로젝트에서는 RGB 픽셀 유형에 대해 다음 정의를 사용합니다.



이 코드는 *RGB*라는 구조를 정의합니다. C에서 struct 데이터 유형은 단일 이름으로 지정된 다양한 유형의 변수 모음입니다. 이 구조에는 *R*, *G* 및 *B*라는 이름의 동일한 유형 (부호 없는 문자, unsigned char)의 3 필드가 포함되어 있습니다. 따라서 각 색상 채널은 8 비트로 표시되므로 각 색상 채널에서 256 개의 서로 다른 강도 레벨을 구별할 수 있습니다. 픽셀 당 총 24 비트 (24bpp)이며, 이것은 현재 디지털 이미지 처리에서 일반적인 형식입니다.

그레이 스케일 이미지를 표현하기 위해 0에서 255 사이의 단일 값 (단일 채널)은 각 픽셀의 밝기를 나타냅니다. 이 ImageProcessing 프로젝트에서는 2 차원 문자 배열을 사용하여 그레이 스케일 이미지를 나타냅니다.



1. **컬러 이미지를 그레이 스케일 이미지로 변환**

2가지 색상 공간 (RGB 및 그레이 스케일) 간의 변환은 다음 가중치 합계를 사용하여 수행됩니다.

grey = 0.299\*R + 0.587\*G + 0.114\*B

이 방정식은 다음에 설명된 알고리즘을 기반으로 합니다. <https://www.mathworks.com/help/matlab/ref/rgb2gray.html>

각 픽셀에 대해 각 색상 채널 방정식에 주어진 가중치를 곱하여 그레이 스케일 값을 계산합니다. 가중치의 합 (0.299 + 0.587 + 0.114)은 1이므로 결과 그레이 스케일 값은 0-255 범위 내에 있으므로 단일 바이트로 표현할 수 있습니다.

방정식에 주어진 가중치를 사용하려면 실수로 연산해야 하지만 **SweRV EH1** 프로세서에는 부동 소수점 지원이 포함되어 있지 않습니다. 한 가지 접근 방식은 시작 가이드의 섹션 5.H에 표시된 DotProduct 프로그램에서처럼 부동 소수점 에뮬레이션을 사용하는 것 이지만, 이 LAB에서는 정수 산술을 기반으로 한 접근 방식을 사용합니다. 가중치는 정수로 변환되고 합계는 2의 거듭 제곱입니다 (이 경우 210). 가중치를 정수로 변환하려면 각 부동 소수점 가중치에 210을 곱하고 가장 가까운 정수로 반올림합니다.

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

물론 최종 그레이 스케일 값을 0-255 범위로 줄이려면 합계를 210으로 나누어야 합니다. 이는 값을 오른쪽으로 10 비트 이동하여 쉽게 완료 할 수 있습니다. 따라서 최종 변환은 다음 공식을 사용하여 얻습니다:

grey = (306\*R + 601\*G + 117\*B) >> 10

상수의 합 (306 + 601 + 117)이 1024 인 경우 결과 그레이 스케일 값은 여전히 0-255 범위 내에 있습니다.

그림 3은 *ColourToGrey*가 호출하는 *ColourToGrey* 함수 (왼쪽) 및 *ColourToGrey\_Pixel* 서브 루틴 (오른쪽)에 대한 코드를 보여줍니다.

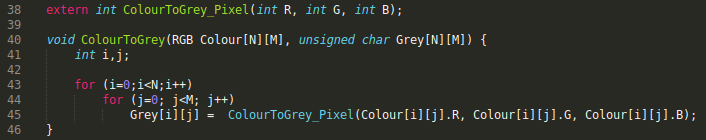
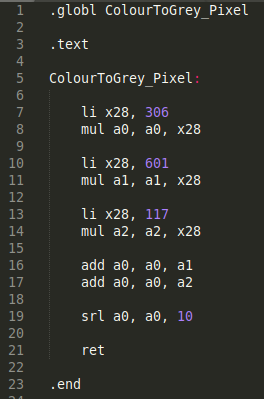
 

그림 3. *ColourToGrey* 함수 (파일 main.c에서 구현 됨) 및 *ColourToGrey\_Pixel* 서브 루틴 (파일 *assemblySubroutines.S*에서 구현 됨).

어셈블리 언어에서 심볼 (변수 및 함수 / 서브 루틴)은 기본적으로 로컬이며, 다른 파일에는 표시되지 않습니다. 이러한 로컬 심볼을 전역(글로벌) 심볼로 변환하려면, .globl 어셈블러 지시문을 사용하여 내보내야 합니다. 그림 3의 오른쪽에서 첫 번째 줄 (.globl ColourToGrey\_Pixel)은 *ColourToGrey\_Pixel* 함수를 생성하여 다른 파일 (*main.c*)에 있는 *ColourToGrey* 함수에서 사용하도록 합니다. 그림 3의 왼쪽에서 첫 번째 줄 (extern int ColourToGrey\_Pixel(int R, int G, int B))은 *ColourToGrey\_Pixel* 함수를 이 파일에 대한 외부 함수로 선언합니다.

1. **프로그램 실행 및 결과 시각화**

그레이 코드 변환이 완료된 후 프로그램 실행이 끝나기 전에 일부 메모리 영역의 내용을 파일로 덤프할 수 있습니다. 이를 위해 GDB 디버거의 dump 명령을 사용합니다. 프로젝트 코드를 실행하고 이미지 결과를 얻으려면 다음 단계를 따르십시오.

1. VSCode 및 PlatformIO를 엽니다.
2. 상단 메뉴 표시 줄에서 *파일 → 폴더 열기*를 클릭하고 *[RVfpgaPath]/RVfpga/Labs/Lab5* 디렉토리로 이동합니다. *ImageProcessing* 디렉토리를 선택하고 (열지 말고 선택만 하면 됩니다) 창 상단에서 확인을 클릭합니다. PlatformIO는 이제 프로젝트를 엽니다.
3. platformio.ini를 열고 board\_build.bitstream\_file의 주석 처리를 제거하고 비트 파일의 디렉터리 위치를 입력합니다. 예를 들어 LAB 1에서 만든 비트 파일을 사용합니다.

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

1. src 디렉터리 (*main.c, assemblySubroutines.S*)에 있는 모든 소스 파일을 열고 분석하여 프로그램 작동 방식을 명확하게 이해합니다.
2. 왼쪽 메뉴 리본에서 PlatformIO 아이콘을 클릭한 다음 프로젝트 작업 → env: swervolf\_nexys → 플랫폼을 확장하고 비트 스트림 업로드를 클릭하여 Nexys A7 보드에 RVfpgaNexys를 다운로드 합니다. Verilator와 Whisper 를 사용하여 시뮬레이션 환경에서 이러한 프로그램을 수행할 수 있습니다.
3. PlatformIO에서 프로그램을 실행합니다. 보드에서 (이 경우 먼저 이전 단계에서 수행 한대로 Nexys A7에 RVfpgaNexys를 업로드해야 함) 또는 Whisper 시뮬레이터 (RVfpga 시작 안내서에 설명된 대로)를 사용하여 수행할 수 있습니다. 어쨌든 PlatformIO의 왼쪽에 있는 "실행"버튼을 클릭한 다음 재생 버튼을 클릭하여 디버거를 시작합니다.

주 기능의 시작 부분에서 실행이 중지되므로 "계속"버튼을 클릭하여 다시 시작하십시오.

잠시 후 (약 1 초), 프로그램은 위에서 설명한 그레이 스케일 이미지 변환을 완료하고 끝에서 무한 루프문 (while(1);)에 도달합니다 (그림 2 참조). 일시 중지 버튼을 클릭하여 실행을 일시 중지합니다.

1. 디버그 콘솔에서 다음 명령을 실행하여 그레이 이미지 (GreyImage)를 내 보냅니다 (이 두 명령의 실행을 보여주는 그림 4 참조).

cd AdditionalFiles

dump value GreyImage.dat GreyImage

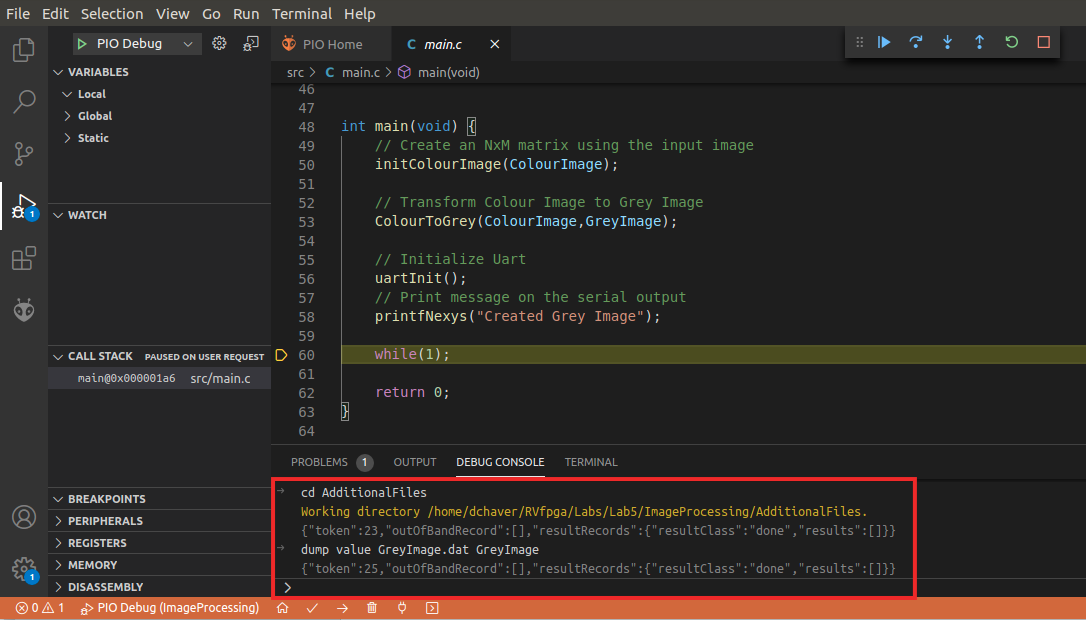


그림 4. 그레이 스케일 이미지를 파일로 내보내기

1. .dat 파일을 시스템에서 볼 수 있는 .ppm 파일로 변환합니다.

**LINUX:** 터미널을 열고 다음 명령을 입력하면 됩니다 (그림 5 참조).

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

gcc -o dump2ppm dump2ppm.c

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

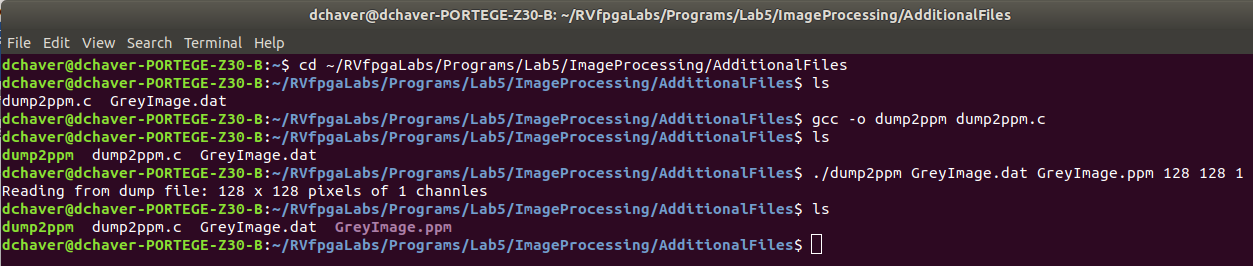


그림 5. 이미지를 *.ppm* 형식으로 변환

**WINDOWS: 다음 중 하나를 수행하십시오.**

1. *[RVfpgaPath]\RVfpga\Labs\Lab5\ImageProcessing\AdditionalFiles*. 에 제공된 dump2ppm.exe 실행 파일을 사용합니다. 명령 셀을 열고 해당 폴더로 이동한 다음 위와 동일한 인수를 사용하여 실행 파일을 실행합니다.

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

또는

2. Cygwin (RVfpga 시작 안내서에 설명된 대로 설치 한 경우)을 사용하여 dump2ppm.c 프로그램을 컴파일합니다. 그런 다음 Cygwin 터미널 또는 위의 옵션 1에서와 같이 명령 셸에서 프로그램 (dump2ppm.exe)을 실행합니다.

1. GNU 이미지 조작 프로그램인 GIMP 를 사용하여 .ppm 파일을 엽니다. 해당 프로그램이 아직 설치하지 않은 경우 다음 웹 사이트로 이동하여 설치 프로그램을 다운로드 하십시오:

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

그레이 스케일 이미지는 그림 1의 오른쪽에 표시된 것과 비슷합니다.

(또한, 사용자는 입력 컬러 이미지를 그림 1의 왼쪽 부분인 *[RVfpgaPath]/RVfpga/Labs/Lab5/ImageProcessing/AdditionalFiles/VanGogh\_128.ppm* 에서 액세스 할 수 있습니다.).

# 연습

**연습 1.** 다른 입력 이미지에서 프로그램을 실행합니다. *[RVfpgaPath]/RVfpga/Labs/Lab5/ImageProcessing/src/TheScream\_256.c*에서 제공된 이미지를 사용할 수 있습니다. (해당 .ppm 이미지는 *[RVfpgaPath]/RVfpga/Labs/Lab5/ImageProcessing/AdditionalFiles/TheScream\_256.ppm* 에서 볼 수 있습니다. 또한 앞서 설명한대로 dat2ppm 프로그램을 실행하여 이미지를 생성할 수도 있습니다.)

**연습 2.** *VanGogh* 그레이 스케일 이미지에서 흰색에 가까운 (> 235) 요소와 검은 색에 가까운 (<20) 요소의 수를 세는 C 함수를 만듭니다. LAB 2의 섹션 3에 설명된 대로 Western Digital의 PSP 및 BSP 라이브러리를 사용하여 직렬 콘솔에 두 번호를 인쇄합니다.

**연습 3. ColourToGrey\_Pixel** 어셈블리 서브 루틴을 C 함수로 변환하고 C 함수 **ColourToGrey**를 **ColourToGrey\_Pixel** C 함수를 호출하는 어셈블리 서브 루틴으로 변환합니다.

* + C에서 모든 함수와 전역 변수는 기본적으로 전역 심볼로 생성하여 *ColourToGrey* 서브 루틴에서 *ColourToGrey\_Pixel* 함수를 사용할 수 있습니다.
  + 어셈블리 언어로 행렬에 액세스하기 위해서는 배열의 시작 주소가 주어지면 요소 (*i*,*j*)의 주소를 계산해야 합니다. ANSI C 표준에 따르면 2 차원 배열은 행 별로 메모리에 저장됩니다. 따라서 *i*행과 *j* 열의 픽셀 주소는 배열의 시작 주소와 오프셋 *i\*M + j)\*B* 를 더하여 얻습니다. 여기서 *M*은 열 수이고 *B*는 바이트 수입니다. 각 픽셀이 차지하는 공간: RGB 3 바이트, 그레이 스케일 1 바이트.

**연습 4.** VanGogh 컬러 이미지에 **블러(Blur) 필터**를 적용합니다 (온라인에서 많은 정보를 찾을 수 있으며, 다음 사이트에서 제공되는 정보를 사용할 수 있습니다: [https://lodev.org/cgtutor/filtering.html#Find\_Edges](https://lodev.org/cgtutor/filtering.html#Find_Edges_)).

.dat 이미지를 .ppm이미지로 변환하려면 1 개가 아닌 3 개 채널을 고려하도록 dump2ppm 명령 호출을 약간 수정해야 합니다.

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

또한 필터링된 이미지를 *[RVfpgaLabsPath]/RVfpgaLabs/Programs/Lab5/ImageProcessing/AdditionalFiles/VanGogh\_128.ppm*에서 사용할 수 있는 원본 이미지와 비교할 수 있습니다.