

EMBEDDED SYSTEMS

BASED ON CORTEX-M4 AND THE RENESAS
SYNERGY PLATFORM

2020

PROF. DOUGLAS RENAUX, PHD
PROF. ROBSON LINHARES, DR.
UTFPR / ESYSTECH

RENESAS ELECTRONICS CORPORATION

LAB3 – ASSEMBLY PROGRAMMING AND ATPCS

Objectives:

- Develop an assembly routine that is callable from a C program. The assembly routine must follow the ATPCS standard.
- The function to be implemented in assembly generates the histogram of an 8-bit grayscale image.

LAB3 – ASSEMBLY PROGRAMMING AND ATPCS

Learning Objectives:

- Apply the embedded software development process presented in Lab 2
- Define the interface between the C program (caller) and the Assembly function (callee)
- Plan the data structures to be used
- Devise an algorithm to generate a histogram
- Implement, Test, Debug

LAB3 – ASSEMBLY PROGRAMMING AND ATPCS

Activities:

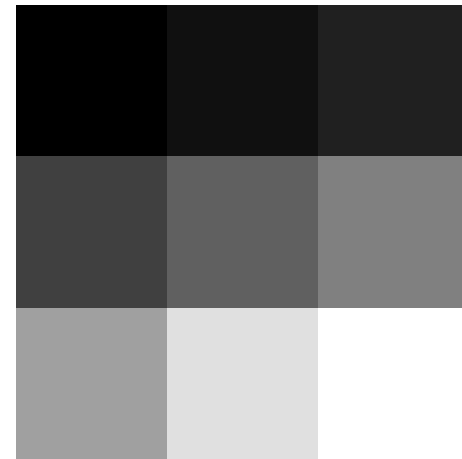
1. Understanding the Problem Domain
2. Problem definition
3. Designing the Data Structures
4. Parameter passing and return of the result
5. Algorithm
6. Implementation
7. Test cases

LAB3 – ACTIVITY 1

Activity 1 – Understanding the Problem Domain

- A raster image or bitmap is composed of dots, or pixels, lay out as a matrix. On a grayscale image, each pixel is represented by a number indicating the level of lighting of that pixel.
- On an 8-bit grayscale image, each pixel is represented by an 8-bit value. Hence, there are 256 levels of gray, ranging from 0 (black) to 255 (white).
- Shown below is a 3 x 3 8-bit grayscale image (9 pixels in total) and the corresponding 9-pixel image.

0	16	32
64	96	128
160	224	255



LAB3 – ACTIVITY 1

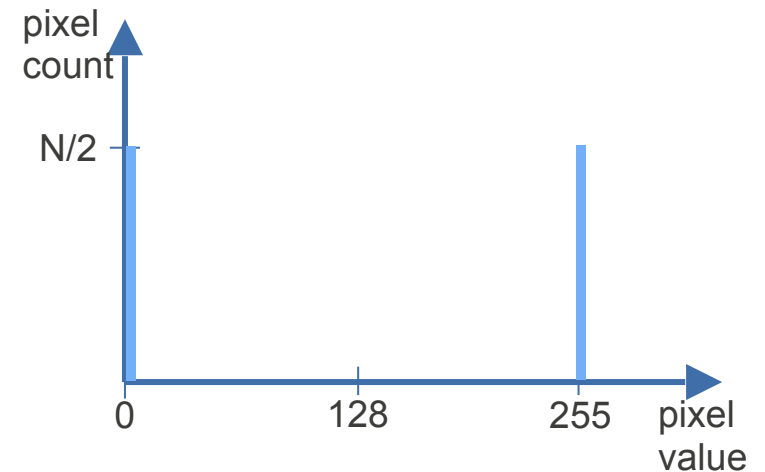
Activity 1 – Understanding the Problem Domain

A histogram is a graphical representation of the tonal distribution of an image. On the horizontal axis there are the possible values that a pixel can have (0-255 in this example) and the vertical axis presents the quantities of pixels with a given luminosity level.

An image with N pixels were half of them are white and half of them are black would have a histogram like this:

Histograms are very useful in digital image processing, to determine thresholds, to adjust brightness and contrast, to identify problems, and many more.

To construct a histogram, all pixels of an image have to be processed, hence, it is desirable to have efficient algorithms and implementations for better performance.



LAB3 – ACTIVITY 2

Activity 2 – Problem Definition 1/2

Develop a function, to be implemented in assembly, that constructs the histogram of an 8-bit grayscale bitmap image.

Input parameters:

- Image width - number of pixels across the image.
- Image height - height of image in pixels
- Starting address - address of the first pixel in memory. Each pixel occupies one byte. The image is represented by a matrix where `image[0][0]` is the upper left pixel of the image. The matrix is stored by rows, hence, the next address holds `image[0][1]` (next pixel on the upper row).
- Histogram - address of a 256-position vector holding 16-bit unsigned integers that hold the pixel counts. The histogram has invalid data when the function is called.

Output: 16-bit unsigned integer indicating the total number of pixels processed.

LAB3 – ACTIVITY 2

Activity 2 – Problem Definition 2/2

Restrictions:

The total number of pixels in the image (i.e. width x height) must be less than 64K (65,536)

Error codes:

Function returns 0 to indicate an error (e.g. image too large)

Function prototype

```
uint16_t EightBitHistogram(uint16_t width, uint16_t height, uint8_t * p_image, uint16_t *  
p_histogram) ;
```


LAB3 – ACTIVITY 3

Activity 3 – Designing the Data Structures

The two important data structures for this problem are the matrix that holds the bitmap and the vector that stores the histogram.

The bitmap matrix has its number of columns equal to the width of the image and its number of rows equal to the height of the image. Each element stored in the matrix is an 8-bit unsigned integer (`uint8_t`) that represents the gray level of the corresponding pixel, 0 being black and 255 being white.

The histogram vector has size 256, hence, its indexes run from 0 to 255. The i_{th} element of the vector stores the count of pixels at value i .

`histogram[i]` = number of pixels whose value is i .

Hence, adding up all elements of the vector must result in a number equal to width x height.

For a 2-pixel high by 3-pixel wide image the matrix would be:

```
uint8_t bitmap[2][3] = {  
    {64, 96, 128},  
    {160, 224, 255}};
```



LAB3 – ACTIVITY 4

Activity 4 – Parameter passing and return of the result

Considering that the function prototype is:

```
uint16_t EightBitHistogram(uint16_t width, uint16_t height, uint16_t * p_image, uint8_t *  
p_histogram) ;
```

By ATPCS the parameters are in registers R0 to R3:

`width` - in R0 (upper half of R0 is 0)

`height` - in R1 (upper half of R1 is 0)

`p_image` - in R2

`p_histogram` - in R3

The result is passed by R0 (upper half is 0)

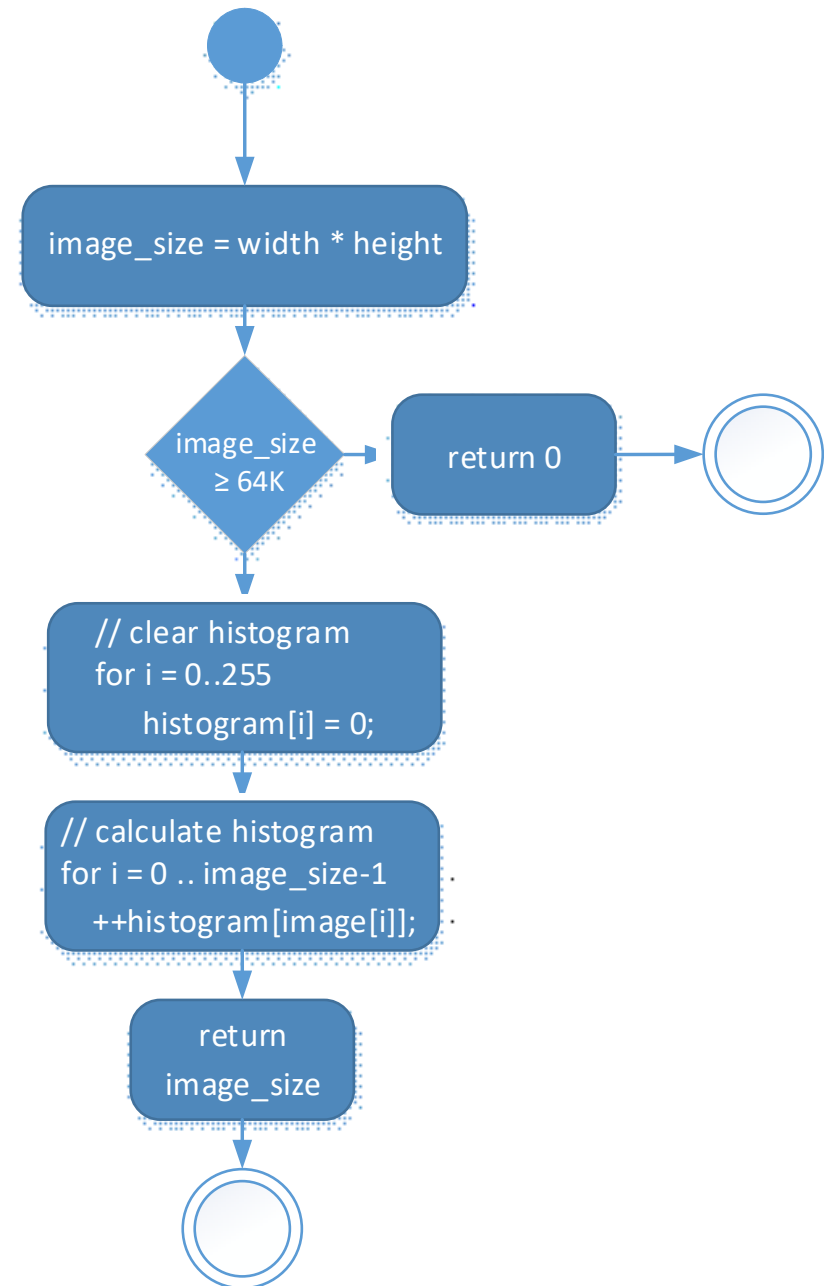
LAB3 – ACTIVITY 5

Activity 5 – Algorithm

One possible solution design is presented here using the UML 2.5 Activity diagram notation.

If the implementation requires registers other than R0..R3 and R12 then there is the need to push these registers at the start and restore them at the end.

Note that the histogram is constructed in a very efficient way by simply using the value of each pixel as an index to the position in the histogram that must be incremented.

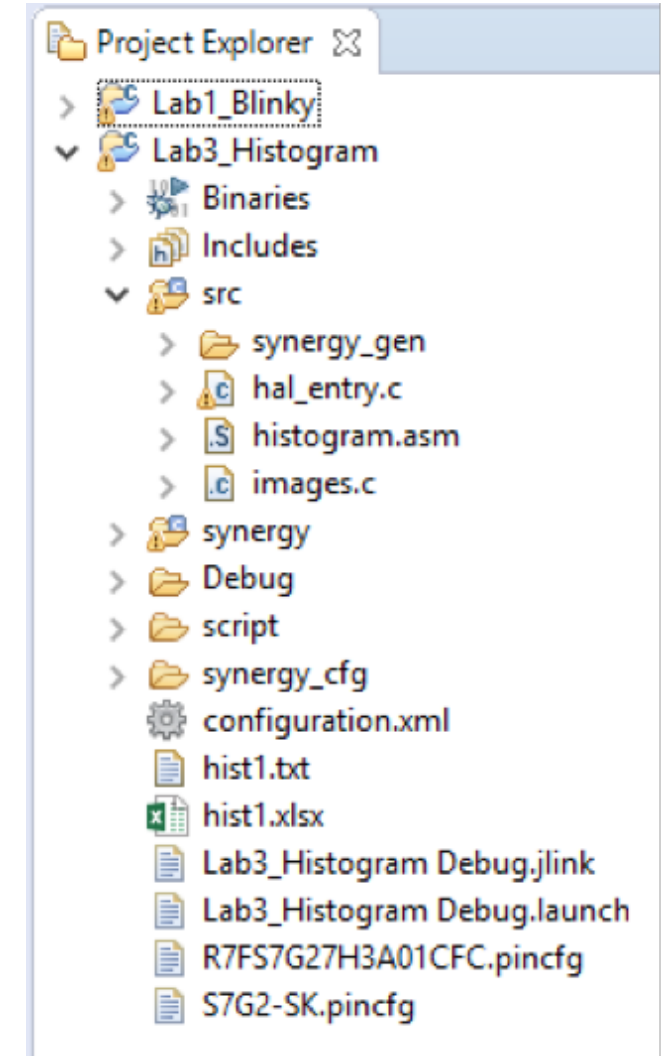


LAB3 – ACTIVITY 6

Activity 6 – Implementation

A possible organization of the source files is:

- `hal_entry.c` this is the C program that calls the assembly function `hal_entry` is executed after initialization; it calls `EightBitHistogram` then presents the results on the virtual console.
- `histogram.asm` this is the assembly source file where `EightBitHistogram` is defined.
- `images.c` holds the matrices with the test images.

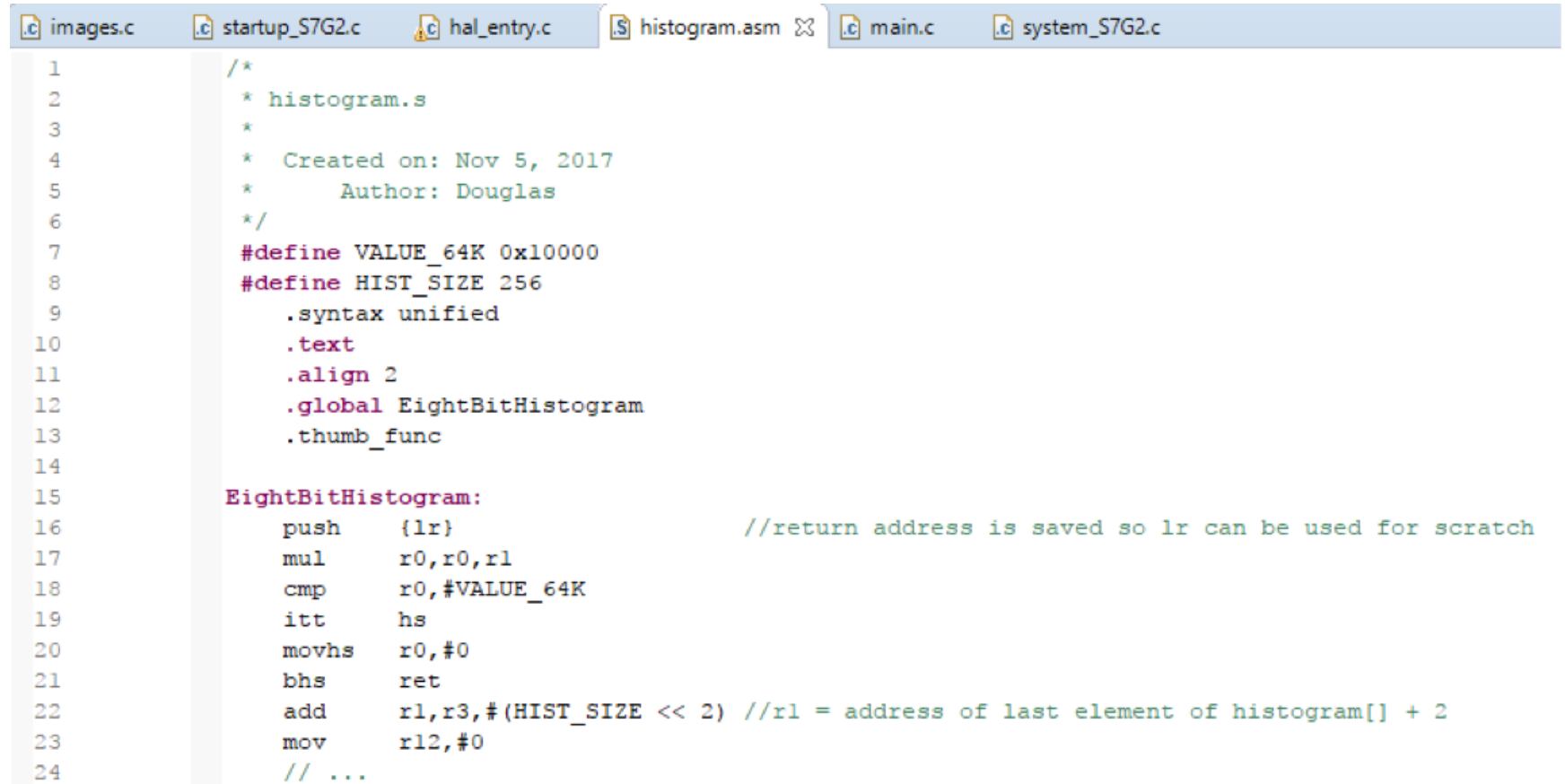


Tips on how to use the Renesas Virtual Console are in slide: [LAB 5 - Activity 3](#)

LAB3 – ACTIVITY 6

Activity 6 – Implementation

The assembly source file requires assembler directives at the start, as shown here.



```
1      /*
2      * histogram.s
3      *
4      * Created on: Nov 5, 2017
5      * Author: Douglas
6      */
7      #define VALUE_64K 0x10000
8      #define HIST_SIZE 256
9      .syntax unified
10     .text
11     .align 2
12     .global EightBitHistogram
13     .thumb_func
14
15     EightBitHistogram:
16     push    {lr}                //return address is saved so lr can be used for scratch
17     mul    r0,r0,r1
18     cmp    r0,#VALUE_64K
19     itt    hs
20     movhs  r0,#0
21     bhs    ret
22     add    r1,r3,#(HIST_SIZE << 2) //r1 = address of last element of histogram[] + 2
23     mov    r12,#0
24     // ...
```

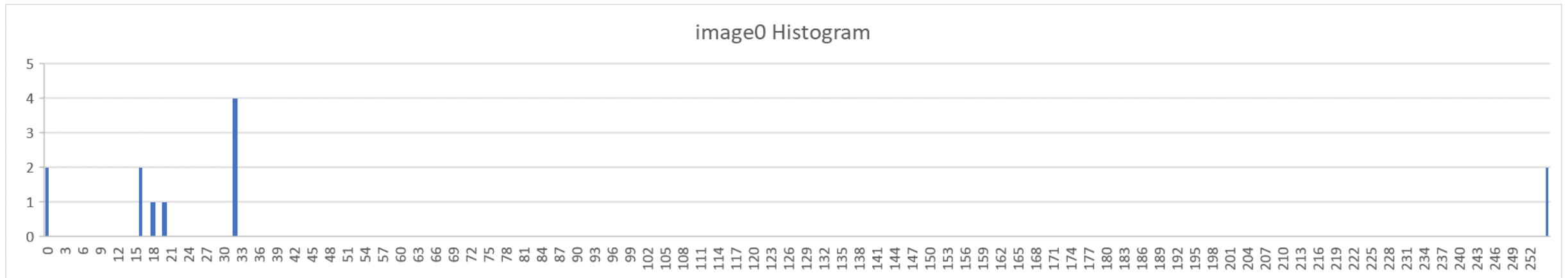
LAB3 – ACTIVITY 7

Activity 7 – Test Cases

Two test cases are provided. The first is matrix image0 that has only 3 lines and 4 columns. Such a small test case is important to debug the implementation on a step-by-step execution.

Shown here is the contents of image0 and the corresponding histogram.

```
#define WIDTH0 4
#define HEIGHT0 3
const uint8_t image0[HEIGHT0][WIDTH0] = {
    { 20, 16, 16, 18},
    {255, 255, 0, 0},
    { 32, 32, 32, 32}
};
```

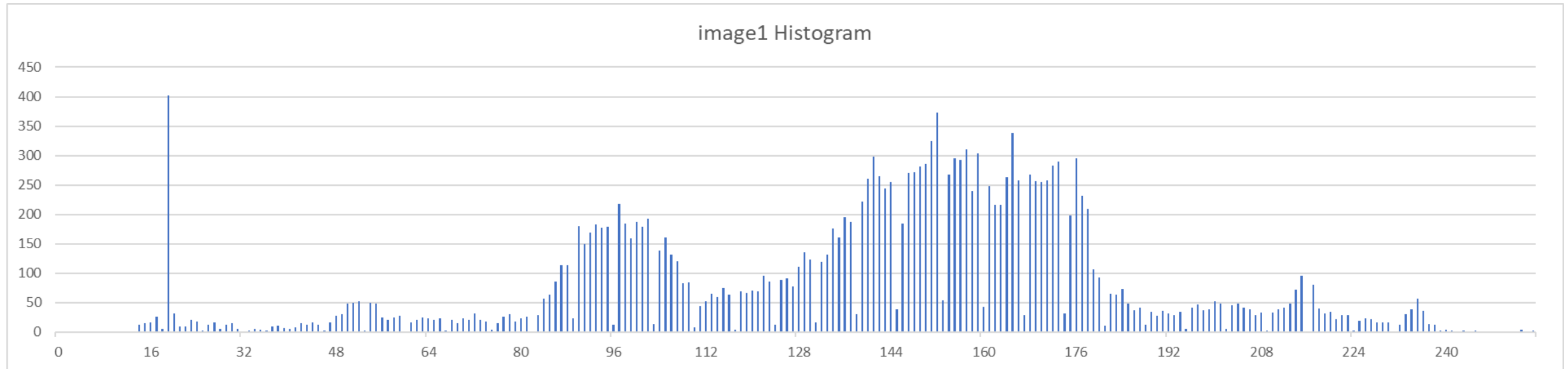


LAB3 – ACTIVITY 7

Activity 7 – Test Cases

The second test case is the test image presented here. Its pixels are encoded in the matrix image1. Its size is 160 x 120 pixels.

The corresponding histogram is presented below:



[Renesas.com](https://www.renesas.com)