COL215 HW Assignment-3

IMAGE FILTER IMPLEMENTATION

Jahnabi Roy : 2022CS11094
Anamika : 2022CS11098

7th November, 2023

1 AIM

The objective of the assignment is: implementation of a 3x3 image filtering operation, which is an extension of the previous hardware assignment. The components involved are memory elements (RAM, ROM and registers), compute unit (filter operation), and VGA controller

2 MODULE IMPLEMENTED

Clock Cycle = 10ns.

2.1 KERNEL IMPLEMENTATION

Designed a model that read the filter.coe and stored the filter values accordingly.
Here, we are basically reading the ROM addresses and storing them all in 9 integers predefined. Now with this, we can now calculate the image filter in the next module.

2.2 FILTER IMPLEMENTATION

Designed a module that read the kernel filter from the distributed ROM and calculate the image filter as being described below.
With the 9 values stored, we know the values will correspond to the values in the matrix. The first 3 values will correspond to the first row of the matrix, the next 3 correspond to the next 3 values and the last 3 values correspond to the last row of the matrix. An 8-bit grayscale image is represented as a 2-D matrix, each element representing a pixel value between 0-255, as shown in Figure 2. Our objective is to perform an image filtering operation using a 3x3 filter. The output image pixel value is the sum of element-wise multiplications between input image pixels and the corresponding kernel. This is what we accomplish by MAC. The

steps of filtering are as follows :

The output pixel value $O(i, j)$ at location (i, j) is computed as the sum of element-wise multiplications between kernel value and input image pixel, as shown in the equation below. The input image pixel at location (i, j) is denoted $I(i, j)$.

$$O(i,j) = a * I(i-1, j-1) + b * I(i-1, j) + c * I(i-1, j+1) + d * I(i, j-1) + e * I(i, j) + f * I(i, j+1) + g * I(i+1, j-1) + h * I(i+1, j) + i * I(i+1, j+1)$$

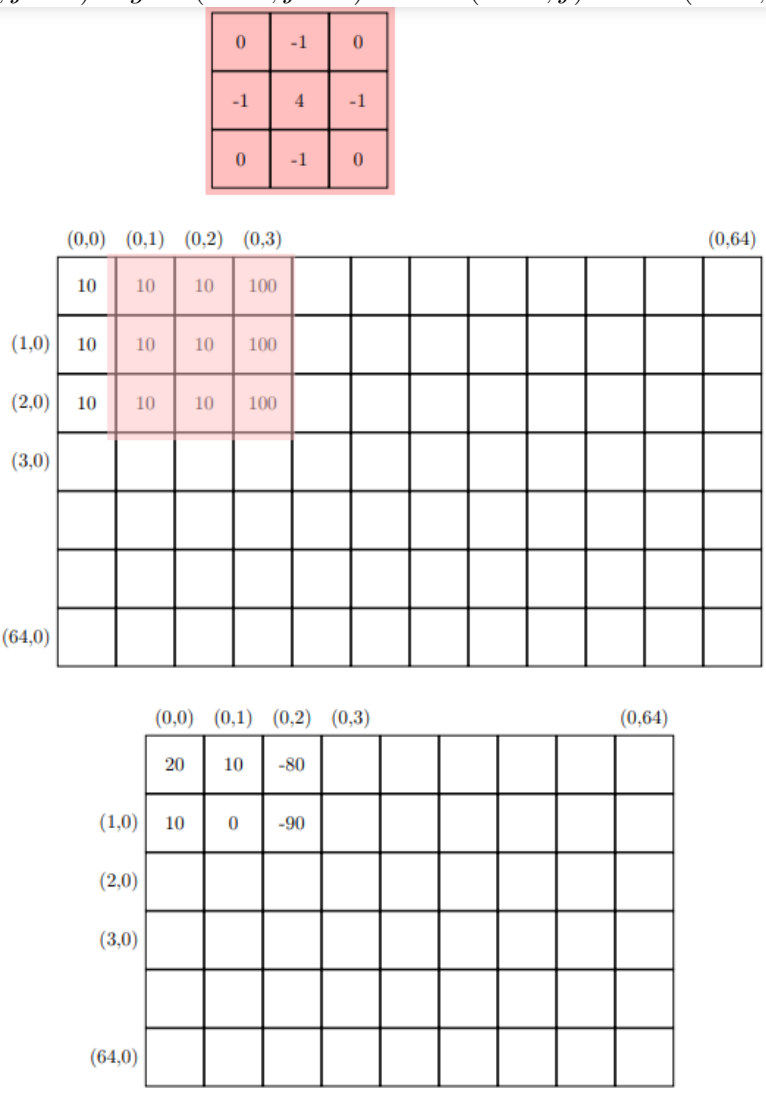


Figure 2: Illustration to perform 1D operation on grayscale image

Now to maintain the output pixel value to be within the range of 0 to 255, we will use image normalization for this purpose.

$$New\_I(i,j) = (I(i,j) - min) * (\frac{255-0}{max-min}) + 0$$

This ensures that the output pixels are now bounded in the region of 0 to 255.

2.2.1 OUR APPROACH

- We stored the input image in a read-only memory (ROM), kept the filter values of the 3x3 pixels and write the output image into the RAM. (RAM).
- The image is just a 256x256 2-D matrix with each pixel of 8 bit. These were stored at address 0 to address 4096. We need to traverse the complete matrix to perform the gradient operation. We kept a check on the row in which we are by the index i and the column by the index j. The address of the pixel was then given by $64 * i + j$.
- We passed the address to the input port of ROM to retrieve the data that has been stored at that address in ROM and stored the data in variables.
- We used nine variables **p1**, **p2**, **p3**, **q1 q2**, **q3**, **r1**, **r2** and **r3** to store the pixel value at the nine variables needed to perform the gradient operation. And we passed the value of the p2 to p1 and p3 to p2 and read a new pixel in p3, q2 to q1 and q3 to q2 and read a new pixel in q3 and r2 to r1 and r3 to r2 and read a new pixel in r3. Thus, optimising the process of reading the pixels from ROM.
- We started off with i and j being 0 and each time a pixel is read we increase j by 1 if j is not equal to 63 because j=63 marks the last pixel corresponding to a line hence, when

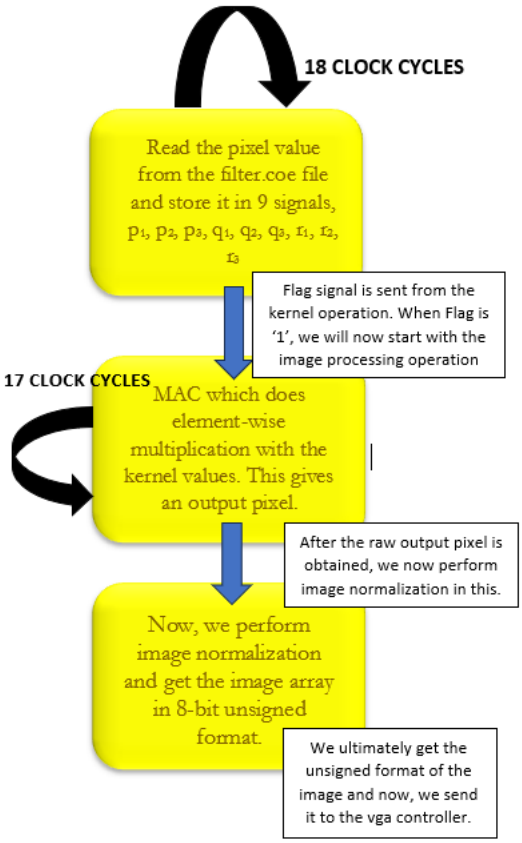
j=63 and that pixel is also read we set j=0 and increase i by 1 due to initialisation of new line.

- The total number of pixels that have been read and operated is stored in a signal count and the reading operation from the ROM is continued till the value of count is less than 4096.

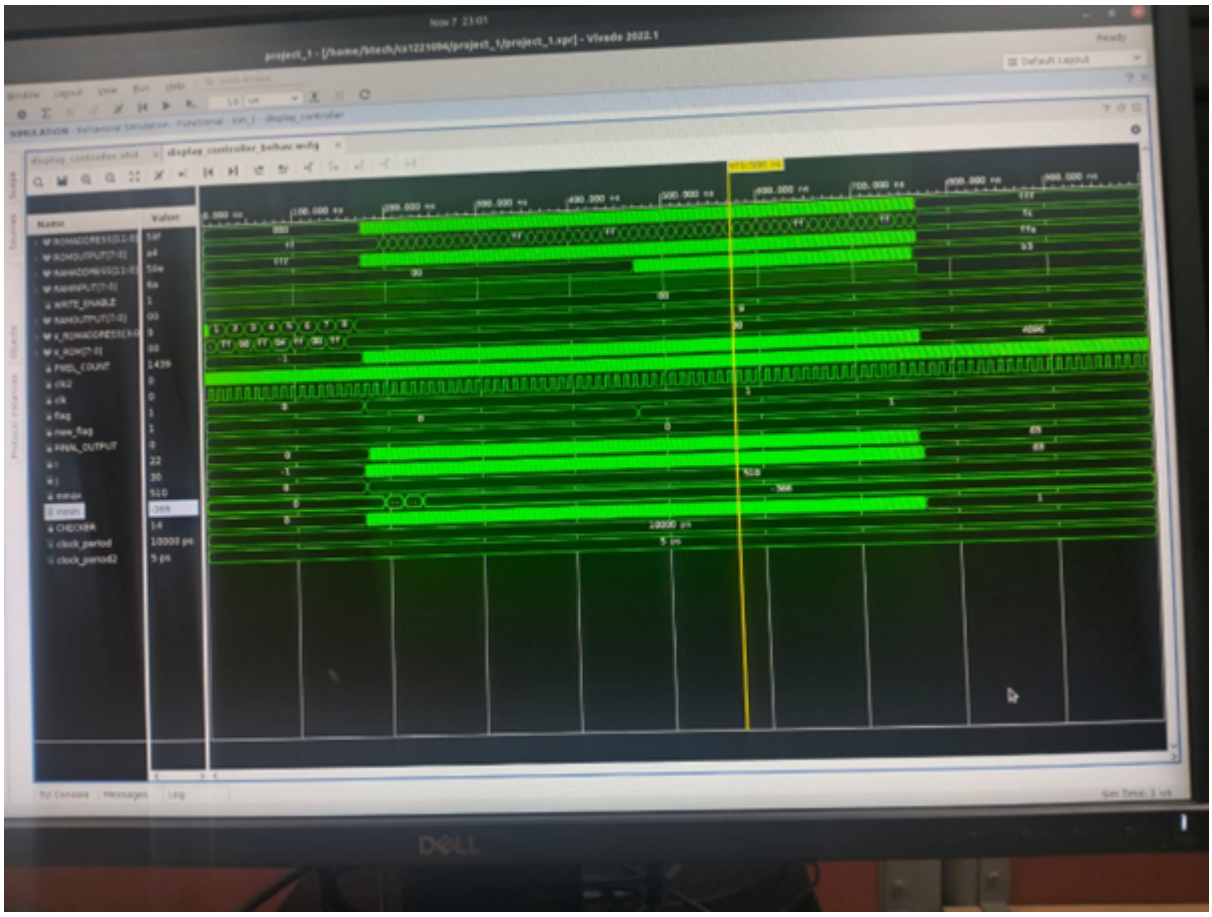
- The edge cases were handled carefully by image normalisation. For this process, we keep a track of minimum and maximum of the entire output array and then use the above formula for normalisation, maintaining the 8-bit unsigned format.

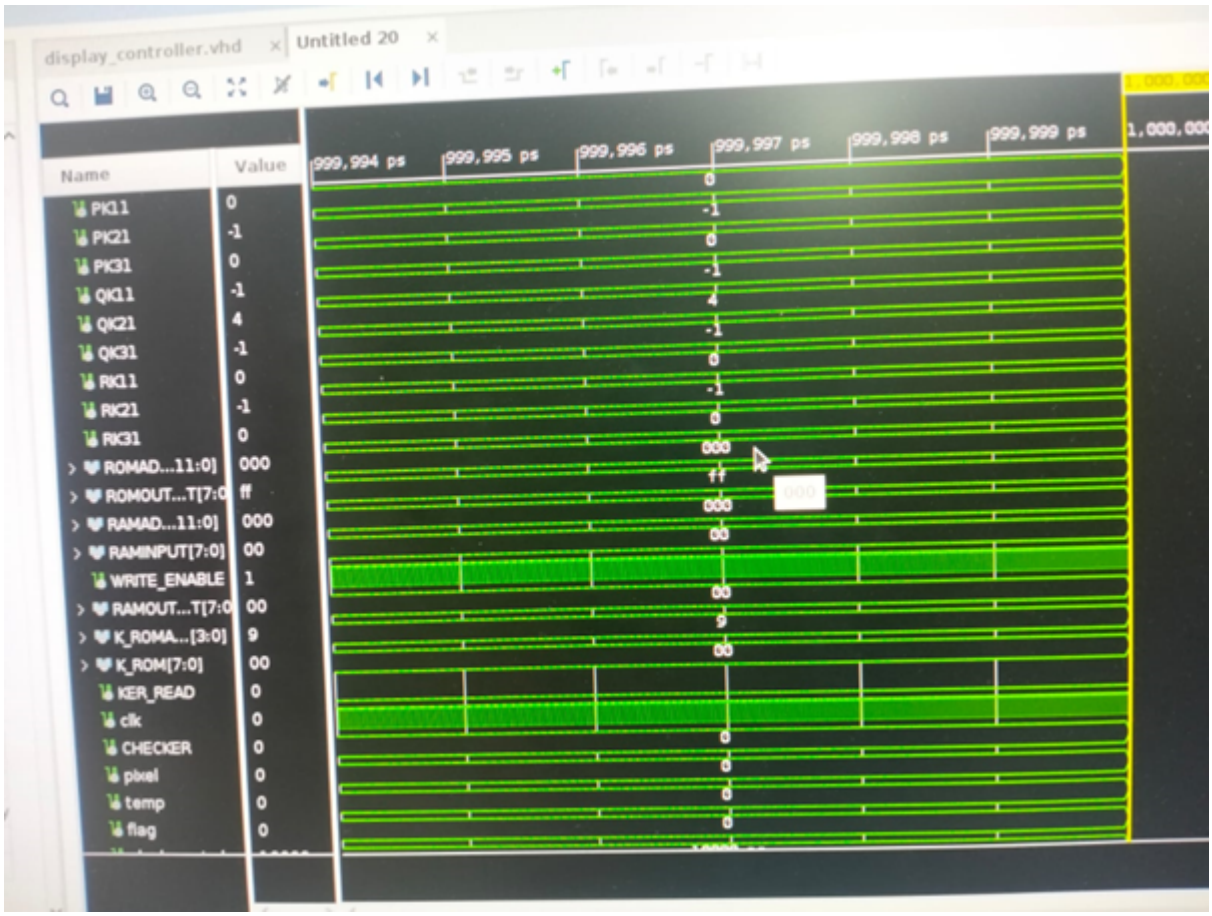
- We then passed the RAM address in one clock cycle and then stored the output pixel, and then followed the image normalisation process.

- To give a breakdown of the processes happening at each clock cycle, one can refer the block diagram below.


18 CLOCK CYCLES
Read the pixel value from the filter.coe file and store it in 9 signals, p1, p2, p3, q1, q2, q3, r1, r2, r3
Flag signal is sent from the kernel operation. When Flag is '1', we will now start with the image processing operation
17 CLOCK CYCLES
MAC which does element-wise multiplication with the kernel values. This gives an output pixel.
After the raw output pixel is obtained, we now perform image normalization in this.
Now, we perform image normalization and get the image array in 8-bit unsigned format.
We ultimately get the unsigned format of the image and now, we send it to the vga controller.

3 OUTPUT DISPLAY AND WAVEFORM





4 ATTACHMENTS AND REFERENCES

The submission of the assignment consists of 1 file - the VHDL code file `DisplayController_gate.vhd`.

TIMESTAMP : 12.42 AM, GROUP 2.

For reference, we used the references given to us in the assignment.