Basic Image Processing Toolbox

Anshul Yadav, Banoth Dinesh, Chitipolu Gowtham, Kaushal Modi
Indian Institute of Technology Gandhnagar

Introduction

In this project, we have implemented image processing operations (those involving convolutions) on a given image through FPGA Basys-3. We send a given image in binary form to the FPGA Block RAM and then perform some specific image processing applications depending user's choice in the FPGA itself and then display it through a VGA display. We use Verilog as the hardware description language and python for converting the given digital image into binary form.

Block Memory

To feed the image into verilog, we need to convert it binary (.coe file). We do that using python. The converted image is such that it has as many rows as the total number of pixel and each row having 24 bit (8X3). So a 160X115p image will have 18400 rows.

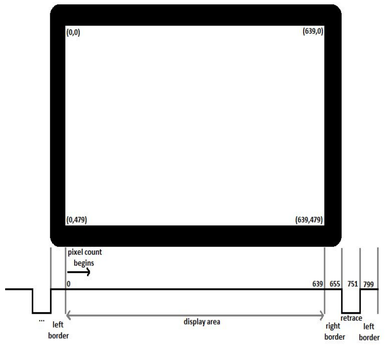
Then a block Memory Module is created in the project which has as many addresses as the number of rows and 24 data bits. So, for the above example, it will have 2^{15} address bits.

This Memory module, like other modules can be instantiated and used in the main module. The module has inputs as clock, address,datain and read-write command and the dataout as output. So, for a given address, it gives the daa at that address during that clock cycle. And thus can give only one data set at a time (here one pixel).

For **convolutions**, we need multiple pixels at a time to use the kernels, that we access by adding and subtracting values to the address.

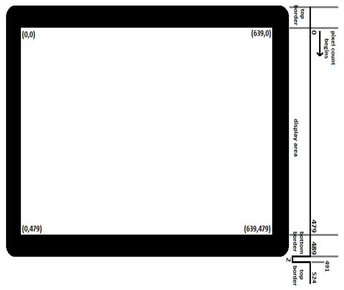
VGA Interface

We wrote the code for 480p display set at 60Hz refresh rate. Refresh rate is the number of times the screen refreshes in a second. For each refresh, each pixel (480X640) is refreshed one after another. For this, the counter starts from the origin (0,0) and travels to (0, 799) and so on till (524, 799). These include the non display area as well along with the retrace as shown on the figures on the left. These are fed to the screen using hsync and vsync.



The hsync signal becomes "0" after the counter reaches the end of the right border which initiates the retrace. After the retrace, the the signal once, once again becomes "1", initiating the left border and tracing on the display area.

Also, for the screen, we only start the display from some point and end at the number of pixels added to that in both the directions.



Original Image (0111)

Each Image consists of three values for each pixel, red, green and blue ranging from 0 to 255. This image is converted into a .coe format using python code and loaded in the bram of fpga. We can do necessary functions using *sel_module* input from fpga.

Grayscale Image(0000)

RGB image can be converted into grayscale by converting the different RGB values of each pixel into a single common value for RGB of each pixel. The formula used here for doing this is
Gray = (Red * 0.3 + Green * 0.59 + Blue * 0.11)

Increase Brightness(0001/0010)

Brightness can be adjusted by adding or subtracting some constant value to the RGB values for each pixel of the original image.

Colour Inversion(0011)

Color Inversion can be achieved by subtracting each RGB value from the maximum possible value, i.e., 255 for each pixel of the original image.

Color filters(0100/0101/0110)

In red, green and blue filters, the value of corresponding RGB value is simply made zero for each pixel or subtracted with a value to adjust the intensity of filter. In this way, that color is filtered out from the original image.

Blurred image(1000/1111)

Image blurring is a technique to reduce the noise in an image by averaging the pixel values of all the adjacent pixels. In a way it smoothes the image and reduces the details of the image.

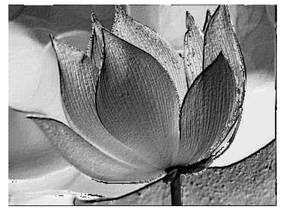


Image Embossing(1100)

An emboss filter gives a 3D shadow effect to the image. This implementation has some very useful applications such as making a bump-map of the image.

Kernel= $\begin{pmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix}$

Motion Blur(1011/1110)

It is simply image blurring (averaging of pixel values) performed in one direction. By doing so, the original image gets an effect as if it were in motion.

Kernel = $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Outline (1010)

To create an outline of the image.

Kernel = $\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$

Image Sharpening(1101)

To sharpen the image(emphasizing the edges)

Kernel = $\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$

Sobel Edge Detection(1001)

An edge detection algorithm to create an image emphasising edges. First we apply sobel operator in x direction and then in y direction and take the rms value of both the values to create the final image.

References:
https://www.wikiwand.com/en/Sobel_operator
https://reference.digilentinc.com/basys3/refmanual
http://setosa.io/ev/image-kernels/