END-SEMESTER PROJECT REPORT

Submitted in partial fulfillment
of the requirements for the course of
Design Project (ECE F376)

IMAGE PROCESSING ON FPGA

by

Abdultaiyeb Vasanwala
2019AAPS0279G

Under the Supervision of
DR. KIZHEPPATT VIPIN

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE,
K.K. BIRLA GOA CAMPUS

October 2021

Acknowledgments

I’d like to express our sincere gratitude to Dr. Kizheppatt Vipin for his constant support and guidance throughout this project.

Table of contents

Introduction

In the computer vision world, objects can be viewed through images. And classifying, tagging, segmenting, and annotating these are images are important to make the objects of interest perceivable to machines. If we want to extract or define something from the rest of the image, e.g., detecting an object from a background, we can break the image up into segments on which we can do more processing. This is typically called Segmentation. Segmentation techniques are used to isolate the desired object from the image in order to perform an analysis of the object. For example, a tumor, cancer, or a block in the blood flow can be easily isolated from its background with the help of the image segmentation technique.

Field programmable gate arrays (FPGAs) are increasingly being used for the implementation of image processing applications. Especially for real-time embedded applications, where latency and power are important considerations. An FPGA embedded in a smart camera is able to perform much of the image processing directly as the image is streamed from the sensor, with the camera providing a processed output data stream rather than a sequence of images. The parallelism of hardware is able to exploit the spatial (data level) and temporal (task level) parallelism implicit within many image processing tasks. Unfortunately, simply porting a software algorithm onto an FPGA often gives disappointing results because many image processing algorithms have been optimized for a serial processor. It is usually necessary to transform the algorithm to efficiently exploit the parallelism and resources available on an FPGA. This can lead to novel algorithms and hardware computational architectures, both at the image processing operation level and also at the application level.

The aim of this project is to learn different image processing algorithms and their need and efficiently implement Image segmentation algorithms on the ZedBoard Zynq Evaluation and Development Kit using Verilog hardware description language.

MATLAB

The aim of implementing algorithms on MATLAB is to get an understanding of how the algorithm works and its need. **Lena** is used for all the experiments which is a standard test image widely used in the field of image processing.

Following algorithms were implemented on MATLAB.

1. Gaussian Blur

Gaussian blur is used to remove the noise from the image before applying other image processing algorithms to get better results. Gaussian blur is a mask processing technique(2D convolution). Mask/Kernal of the filter is a 2D Gaussian Function with mean zero and variance one.

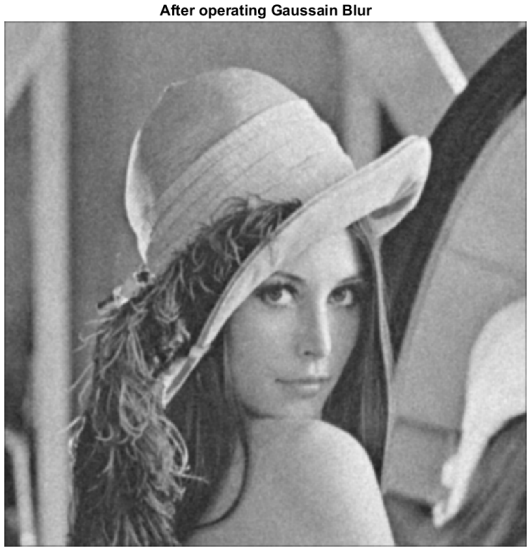
$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \qquad \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

It is a weighted average of the pixels in the neighborhood pixels, with the target pixel having the highest contribution and decreasing contribution as we move away from the pixel. It preserves the edges in an image slightly better than a mean blur filter.

The output of MATLAB implemented Gaussian Blur algorithms:

Lenna with Gaussian Noise

After operating Gaussain Blur



2. Sobel and Canny Edge Detection

Sobel filter is used to detect edges in an image. It is a mask processing technique. It uses two kernels that are convolved with the original image to get approximate derivates along with the horizontal and vertical directions.

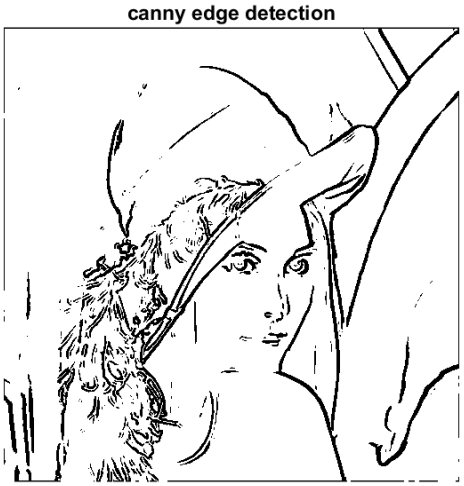
$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

Combining information from both the outputs, we can net gradient and its direction.

$$\mathbf{G} = \sqrt{{\mathbf{G}_x}^2 + {\mathbf{G}_y}^2} \qquad \mathbf{\Theta} = \operatorname{atan2}(\mathbf{G}_y, \mathbf{G}_x)$$

Sobel filter detects minor changes in the gradients, and hence there can be some edge pixels that are caused by noise and color variation. In order to account for this edge with pixels with a weak gradient should be eliminated, and only high gradient pixels must be preserved. This can be done by selecting high and low threshold values. This, combined with Sobel, is called **Canny Filter**.

The output of MATLAB implemented Sobel and Canny edge detection algorithms:



3. Otsu's Thresholding

Otsu's thresholding method iterates through all the possible threshold values and calculates a measure of spread for the pixel levels on each side of the threshold, i.e. the pixels that fall in foreground or background. The aim is to find the

threshold value where the sum of foreground and background spreads is minimum. This final value is used to convert appropriately grayscale image to B&W image for various operations.

The threshold value for Lenna from otsu's method is 117, and applying the thresholding; we get the following result:

4. Dilation and Erosion

In a binary image, **Dilation** expands the connected sets of ones. This can be used to grow features to approximate them to standard shapes and fill holes and gaps. In contrast, **Erosion** shrinks the connected sets of ones in a binary, which is used to shrink features and remove bridges, branches, and protrusions to approximate them to standard shapes.

The output of MATLAB implemented Dilation and Erosion edge detection algorithms:

FPGA/Vivado/iVerilog implementation

FPGAs (Field Programmable Gate Arrays) are semiconductor devices that consist of a matrix of configurable logic blocks (CLBs) linked by programmable interconnects. After production, FPGAs can be reprogrammed to meet specific application or feature specifications.

Vivado Design Suite is software by Xilinx for the synthesis and analysis of hardware implementations and can be used to simulate HDL codes and program FPGAs. In the project, Vivado is used extensively for hardware simulations using Verilog HDL.

1. Mask Processing IP

The implementation has the following modules **Line Buffer**, **Convolution**, **Control Logic,** and **Top module. Line Buffer** takes input as and stores a row of the image and outputs 3 pixels (as the kernel is 3x3) and goes to the next group of 3 pixels till the end of the row. 4 line buffers are used to implement the **Control Logic** module. Forth line buffer is used to load the next row into it while the other 3 are in and as soon as the next operation starts, the line buffer with the first row gets loaded with the next row of the image. Control logic unit outputs 9 pixels to be operated. The **convolution** module takes 9 pixels (3x3) from 3 line buffers, and outputs convolved 1 pixel. The **top module** connects the control logic module and convolution module. A FIFO output buffer is connected as there is a mismatch between input and output due to pipelining. An appropriate testbench was written to open grayscale Lenna pass it as input to the IP and store the blurred image into a file.

2. RGB to Grayscale IP

rgb2gray module takes 1 colored pixel (3 bytes) and converts it to the corresponding gray pixel (1 byte) by using a digital approximation of the formula `gray level =0.2989*R + 0.5870*G + 0.1140*B`. An appropriate testbench was written to take colored Lenna as input and store grayscale, Lenna.

All Verilog, Simulation, and MATLAB files are uploaded on google drive and the link is provided in the Bibliography section.

3. Fast Image Processing IP

The implementation has taken from [6] and has the following modules **Data Register, Line Buffer**, **Image Buffer, Convolution module**, and **Tb module. Data Register** is a basic register with one input parameter as the size of the register in bits which would be taken equal to the data width of the image. **Line Buffer** takes input and makes a row of the image available at the output (512*8 bits in case of test image), using Data registers. **Image buffer** uses 3 Data Register and 2 line buffers to sequentially output kernel of the data on which operation has to performed, taking sequential input of center pixel of the kernel from the test image. The **convolution** module takes 9 pixels (3x3) from Line Buffer, and outputs convolved 1 pixel. An appropriate testbench was written connecting image Buffer and convolution module and to open grayscale Lena pass it as input to the IP and store the blurred image into a file.

Erosion and **Dilation** module added to perform the operations by the same name with passing parameter data width as one through the image buffer. Binary test image Lena was passed as input and eroded and dilated images were obtained.

Updates

Faster image processing IP was implemented which continuously takes in data and outputs operated data (Fast Image Processing IP). hardware implementation of the dilation and erosion were added. concepts related to opening and closing were implemented on Matlab for understanding along with image segmentation concepts.

Future Timeline

Then hardware implementation of edge detection algorithms has to be done. Combining all these implementations a hardware implementation of image segmentation has to be achieved. Further, the next step would be to test it on various images and optimize it.

Bibliography

[1] Rayan Potter. Why Image Segmentation is Needed - medium.com
URL: https://becominghuman.ai/why-image-segmentation-is-needed-image-segmentation-techniques-ee52b92e651a

[2] ZedBoard's Specs and details
URL: https://www.element14.com/community/docs/DOC-95651

[3] Wikipedia contributors. various pages - Wikipedia, the free encyclopedia, 2021.
URL:https://en.wikipedia.org/wiki/Sobel_operator
URL:https://en.wikipedia.org/wiki/Canny_edge_detector

[4] Various authors. Image Processing Using FPGAs - Journal of Imaging.

[5] Various authors. MATLAB documentation - Mathworks.

[6] A survey of FPGA-based accelerators for convolutional neural networks
https://www.researchgate.net/publication/327931012_A_Survey_of_FPGA-based_Accelerators_for_Convolutional_Neural_Networks

[7] Link to Academic only Google drive :
URL:https://drive.google.com/drive/folders/1Nlr-I6lLCwNwX8w2zWfbTyT6-5uRE3ZM?usp=sharing