Real Time Imaging & Control
Real Time Filtering

Masters in Computer Vision

UNIVERSITE DE BOURGOGNE

Centre Universitaire Condorcet - UB, Le Creusot

11 January 2018

Submitted By:
Mohit Kumar Ahuja
Meng Di

Submitted To:
Dr. Julien Dubois

Index

1. Introduction

Image processing is considered to be one of the most rapidly evolving areas of information technology, with growing applications in all fields of knowledge. It constitutes a core area of research within the computer science and engineering disciplines given the interest of potential applications ranging from image enhancing, to automatic image understanding, robotics and computer vision. The performance requirements of image processing applications have continuously increased the demands on computing power, especially when there are real time constraints. Image processing applications may consist of several low level algorithms applied in a processing chain to a stream of input images. In order to accelerate image processing, there are different alternatives ranging from parallel computers to specialized Application Specific Integrated Circuits (ASIC) architectures. The computing paradigm using reconfigurable architectures based on Field Programmable Gate Arrays (FPGAs) promises an intermediate trade-off between flexibility and performance.

Various techniques have been developed in Image Processing during the last four to five decades. Most of the techniques are developed for enhancing images obtained from unmanned spacecraft's, space probes and military reconnaissance flights. Image Processing systems are becoming popular due to easy availability of powerful personnel computers, large size memory devices, graphics software etc.

Filtering is fundamental operation in image processing, it can be used for image enhancement, noise reduction, edge detection, and sharpening, the concept of filtering has been applied in the frequency domain, where it rejects some frequency components while accepting others. In the spatial domain, filtering is a pixel neighborhood operation. Commonly used spatial filtering techniques include: (median filtering, average filtering, Gaussian filtering, etc.). The filtering function sometimes is called filter mask, or filter kernel. They can be broadly classified into two different categories: linear filtering and order-statistic filters. The common elements of a filter are the neighborhood including the pixel itself. Typically the neighborhood is a rectangular of different size, for example 3×3, 5×5… and smaller than the image itself.

Neighborhood of pixels is also called windowing operators that are use a window to calculate their output. For example, windowing operator may perform an operation like finding the average of all pixels in the neighborhood of a pixel. The pixel around which the window is found is called the origin. Figure 1, shows a 3×3 pixel window and the corresponding origin.

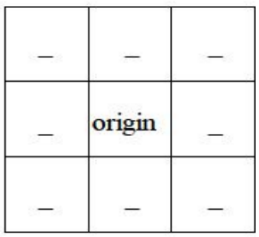

Figure 1: 3×3 Pixel window and origin

Programmable logic is emerging as an attractive solution for many digital image processing applications. As image sizes and bit depths grow larger, software has become less useful in the image processing, Field Programmable Gate Array (FPGA) technology has become a viable

target for the implementation of algorithms suited to image processing applications, the unique architecture of the FPGA has allowed the technology to be used in many such applications encompassing all aspects of image processing. Image smoothing is one of image processing applications, it often done to reduce the effect of pixel noise in images.

As we are also supposed to implement 4 of image filters:

a) Averaging filters with kernel $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$.

b) Averaging filters with kernel $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$.

c) Sobel Filter – Horizontal

d) Sobel Filter - Vertical

e) Gaussian Filter.

Image smoothing algorithms are particularly suitable for implementation on FPGA, due to the parallelisms that may be exploited. Due to use of microcontroller or microprocessor instruction level parallelism is achieved. Research has been conducted to improve speed by designing system block by block.

2. Problem Statement

The goal is to process the input data flow (corresponding to lena image) using a 2D filter. Two main tasks are expected:

- The design and the validation of a customizable 2D filter (filter IP)
- The implementation on a Nexys4 evaluation board of the 2D filter. The filter IP implementation should be included in a reference design (furnished by teacher) to ease the integration.

The filter IP could be split into two main parts: the memory cache which aims to be temporarily stored the data flow before filtering and the processing part.
The cache memory designed for simultaneous pixel accesses enables a 3x3 pixel neighbourhood to be accessible in one clock cycle. The structure is based on flip-flop registers and First-In-First-Out (FIFO) memory. The structure is represented in the following:

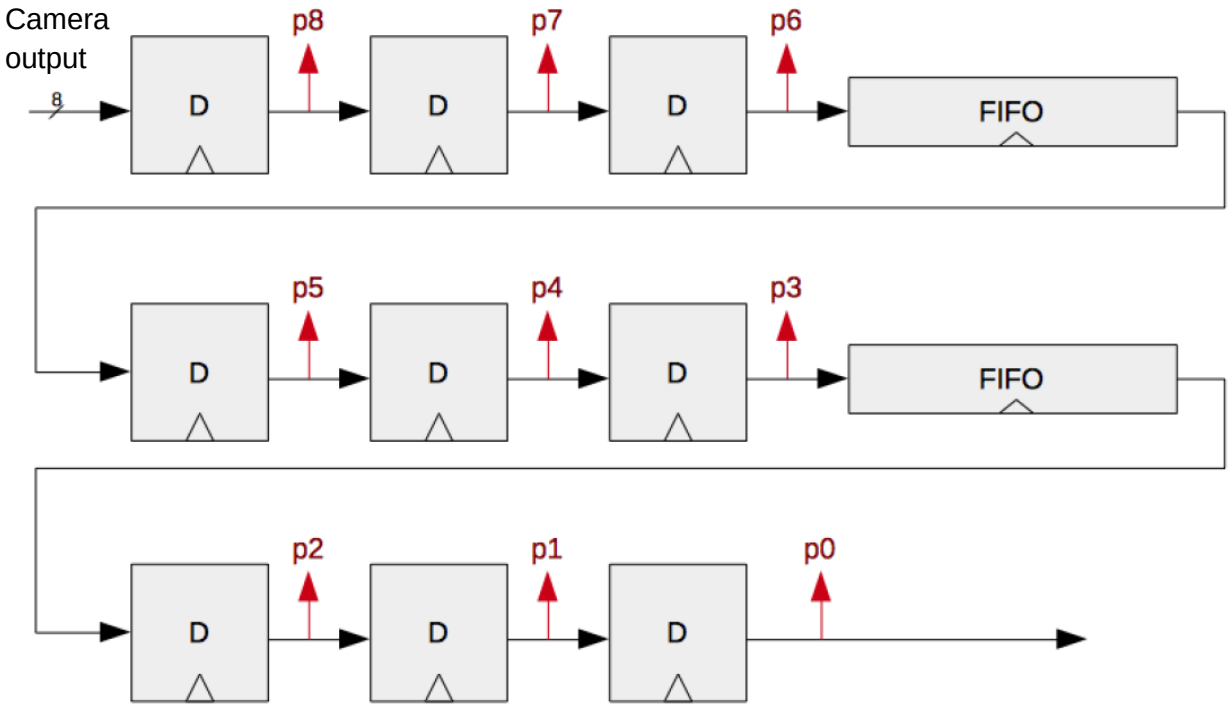

Figure 2: Structural Diagram

The processing part is in charge of filtering to be performed. The nine pixels extracted at each cycle in the memory cache are used in a pipelined architecture (to be designed). An example of a simple average filter based on pipelined structure is represented in the following figure:

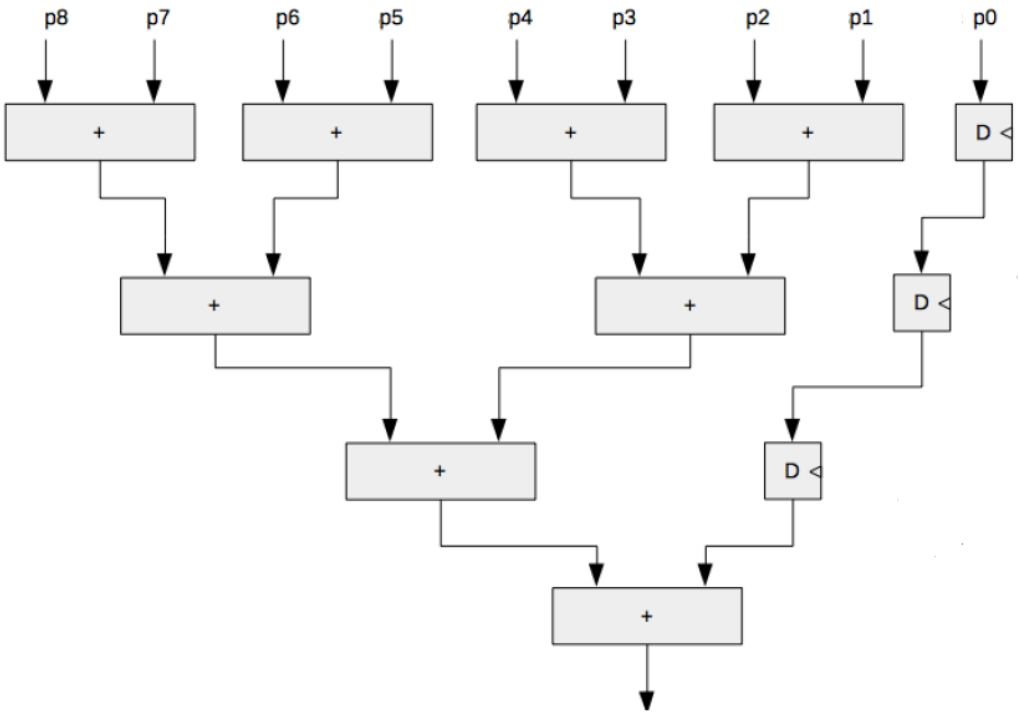

Figure 3: Structure to Implement

A stage of multipliers has to be included to perform more complex filtering. The filter coefficients must be tuneable (configuration mode to be done using evaluation board switches). The student should proposed different filter:

a) Average (blur effect) using the following coefficients $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ and then $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$.

b) A Sobel filter (horizontal and vertical filter to be tested)

c) Gaussian filter

2.1 Solution Proposed

We implemented this problem in VHDL using Xilinx software in 3 steps:

1. Cache Memory – To hold the Value of Pixels
2. Processing Unit – To apply different kernels for different Images
3. Test Bench – To integrate both and to get an output.

{Our Code Is well commented for better understanding of Viewer}

3. Cache Memory

In order to realize the real time image processing, Cache memory is programmed and used in this project. It aims to be temporarily stored the data flow before the data going to the processing unit. The cache memory is customizing designed to access the image by 3x3 neighborhood in one clock cycle. The structure is based on flip-flop registers and First-in-First-out (FIFO) memory which is represented in the following figure 4.

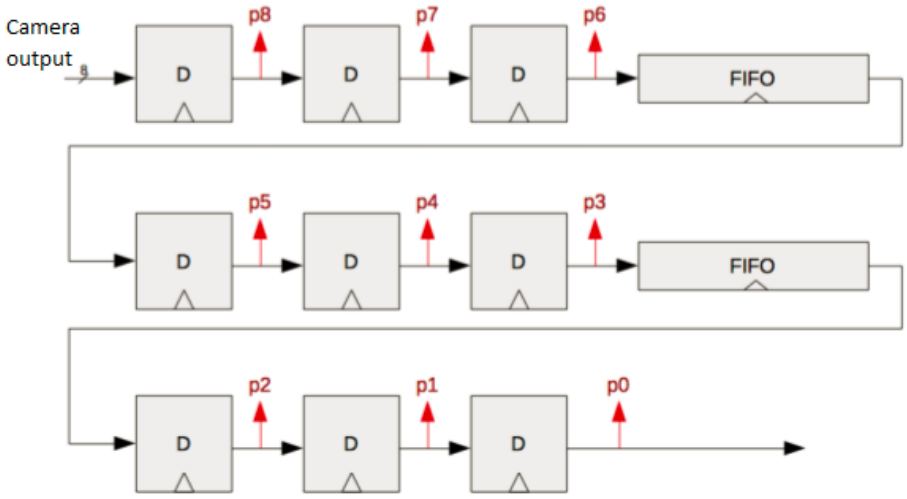

Figure 4: Schematic of the Plan

3.1 Flip Flop

A generic D flip-flop is generated. The D flip-flop captures the value of the D-input at a definite portion of the clock cycle (such as the rising edge of the clock). That captured value becomes the Q output. At other times, the output Q does not change. So it usually works as a memory cell. In this project, the input and output data is 8 bits. The register outputs one data every clock cycle.

Though we didn't used Flip-Flop for temporary storage of 9 pixels, but it is being used for temporary storage of only the 9th pixel while rest 8 of them will be in addition process as shown in figure 5.

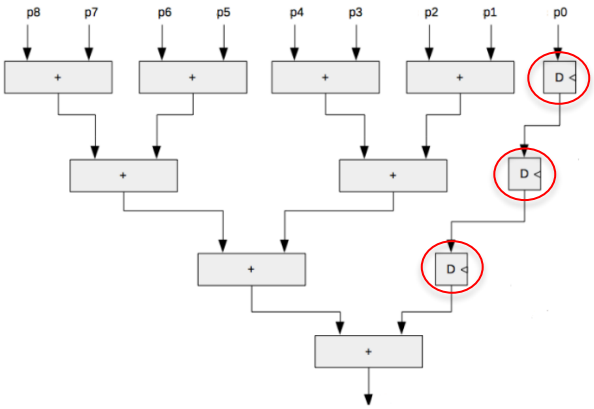

Figure 5: D-FLIP-FLOP usage

3.2 FIFO

FIFO is an essential memory buffer used to temporarily store the data until another process is ready to read it. It has the structure that the first data going in comes out firstly. FIFO can be defined to have a specific amount of storage. Besides, FIFO has read and write enables to control the reading and writing process. It would start writing data out when the storage is full. In addition, FIFO has a key flag "prog_full" which is a customized limitation of storage. A threshold can be set to define the storage that how much data can be buffered. The flag would be set to one and the FIFO cannot read anymore when the data is accelerated reaching the threshold.

We generated 1 FIFO and declared it twice in the code so it acted as 2 FIFO'S. We can generate any generic component and can use as many times as we can as shown in figure 6:

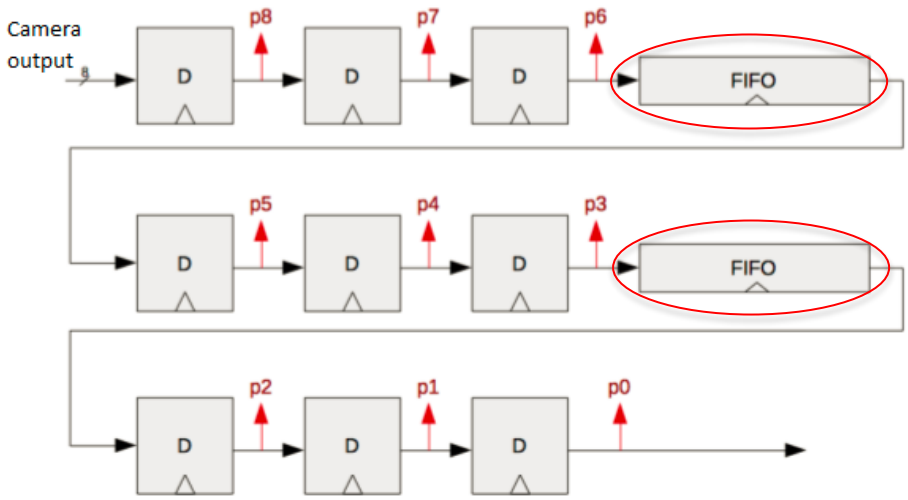

Figure 6: FIFO Usage

3.3 Structure

In our project, we are dealing with a 128x128 size image and 3x3 neighborhood to convolve. Despite the nine pixels are accessable each clock cycle, the rest of the data are buffered in the FIFO memory. The threshold for each FIFO is set to 123 in this case.

A testbench is created to test the cache memory and the Processing Unit. Reading the Lena image as a sequence of data, the cache memory outputs groups of nine pixels which are 3x3 neighborhood along clock cycling. The cache memory starts to write data out after inputing (128x2+3) datas.

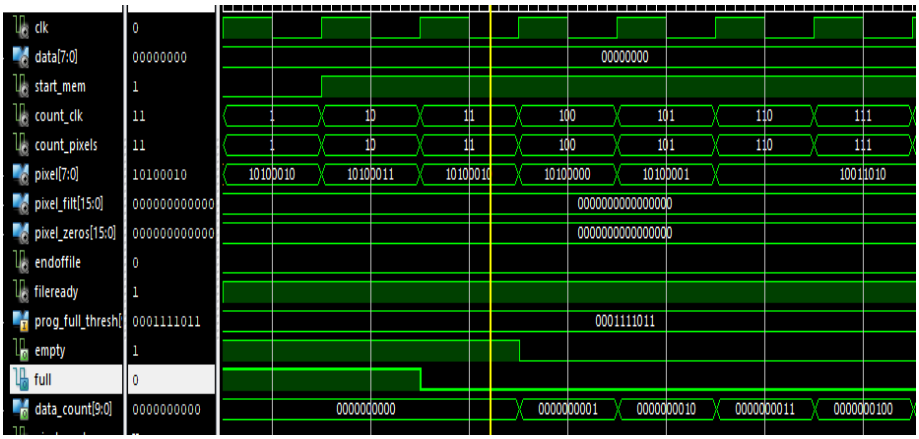

Figure 7: Structural Diagram showing Output of the Filter

4. Processing Unit

We solved the problem in two different ways and both of them will be explained in detail. The two methods used are:

1. By using NUMERIC_STD library.
2. By defining Components and making it more generalized.

4.1 Using Library NUMERIC_STD

I was searching for different ways to solve the problem and on "Google" it showed me that we can also use "Numeric" library to solve our problem. So, I started writing a code in VHDL. This process is commented in file "FILTER_PROCESS.vhd", you can uncomment the other process and uncomment this one to see the output of this filter.

In this process, we need not to define special components like we did in the second process. We just need to use arithmetic symbols for every operation which we want to perform. For example: we want to add two pixel:

```
Sum0 <= PIXEL_1 + PIXEL_2;
```

So, this will return us the sum of Pixel 1 and pixel 2.

4.1.1 Results

We got quite promising results using this process. We used this process to be used as an Averaging Filter and the output of this Process is:

Figure 8: Output of Averaging Filter Using Process-1

4.2 Process-2 (Declaring Components)

In the second process, we declared components like multiplexer, adder, D-FlipFlop and divider. This process is completely generic and can be used for any kind of

filtering. We just need to change the kernel of the filter and it will give us thee output for that kernel. The components used in this process are explained below:

4.2.1 Multiplexer

A multiplexer allows digital signals from several sources to be routed onto a single bus or line. A 'select' input to the multiplexer allows the source of the signal to be chosen. It is another word for a selector. It acts much like a railroad switch. This picture shows two possible source tracks that can be connected to a single destination track. The railroad switch controls via some external control which train gets to connect to the destination track. This exact same concept is used with a 2-1 Mux. Two inputs can connect to a single output.

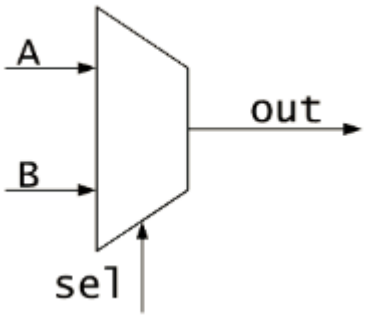

Figure 9: 2:1 Multiplexer

We generated one multiplexer and used it 9 times to multiply the kernel with the Pixel values. And the output of the Multiplexers are stored in temporary variables.

4.2.2 Adder

An adder is used to perform addition of numbers. In many computers and other kinds of processors adders are used in the arithmetic logic units or ALU. They are also utilized in other parts of the processor, where they are used to calculate addresses, table indices, increment and decrement operators, and similar operations.

Although adders can be constructed for many number representations, such as binary-coded decimal or excess-3, the most common adders operate on binary numbers. In cases where two's complement or ones' complement is being used to represent negative numbers, it is trivial to modify an adder into an adder–subtractor. Other signed number representations require more logic around the basic adder.

In our program, we generated an adder and then we used that adder 8 times as shown in figure 10.

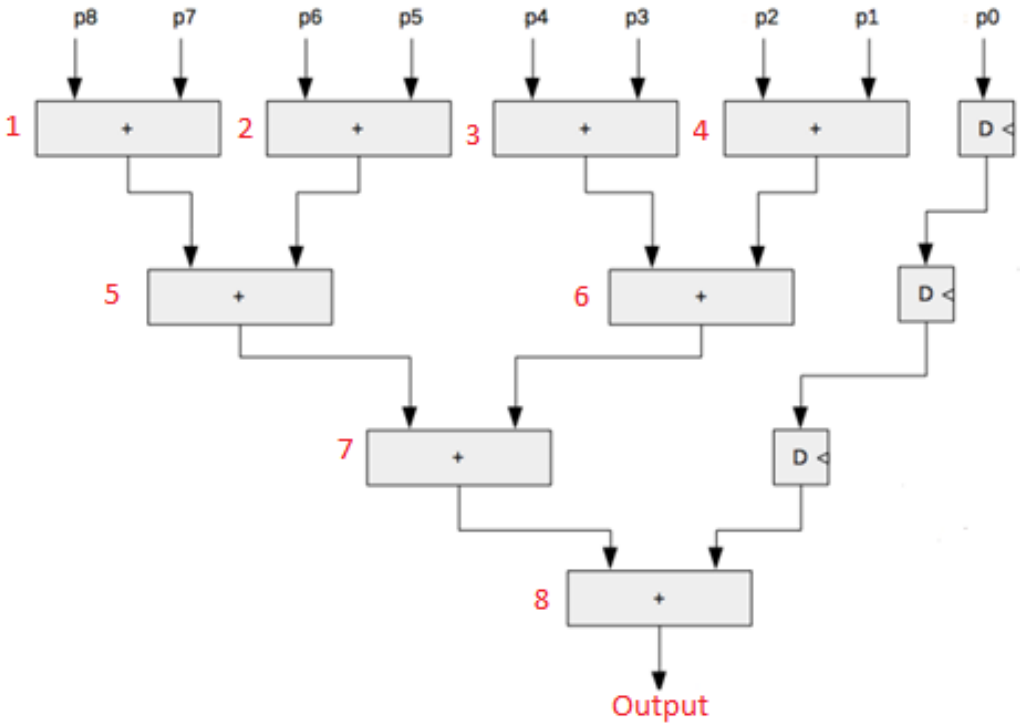

Figure 10: Adders Used

4.2.3 Divider

Dividend, divisor, and quotient are assumed to be n-bit quantities. The initial partial remainder is n 0s followed by the dividend. The basic division algorithm recursively applies the following divide step to find successive bits of the quotient beginning with the MSB. Shift the partial remainder one bit to the left. Subtract the divisor from the partial remainder. If the new partial remainder is positive, then set the new quotient bit to 1 and shift the new partial remainder and the quotient one bit to the left. If the new partial remainder is negative, set the new quotient bit to 0, and replace the new partial remainder with the original one and shift it and the quotient one bit to the left. Repeat this divide step until the all bits of the quotient have been generated to produce the quotient and remainder.

In our code, we generated one 16-Bit Divider and we used only once to divide the output of the filter. For Example: If it's an average filter with 3*3 kernal then the output will be divided by 9. So as we used it to divide the output of adders and to give the final output.

4.3 TASK - Average Filter

We had to implement two average filters, the first average filter we used during implementation is mean filter, which is to obtain the mean of nine pixel values in 3x3 neighborhood and assign the value to the center pixel. The kernel mask is showing in the following:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Figure 11: Kernel-1

The second Kernel for averaging filter we used is with kernel:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 0 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Matrix 1: Kernel-2

4.3.1 Problems Faced

We faced some problems while implementing this kernel. As it will be clearer from the result, we tried a lot of things to improve the result but we were not able to resolve the issue. The good thing was that the output of this filter is blurry as it should be but the problem we faced was that the image at the output has different gray scale than the original one.

4.3.2 Results – Kernel 1

While using Kernel 1 shown in figure 12, we got results as explained before;

Figure 12: Output of Average Filter using kernel-1

4.3.3 Results – Kernel 2

While using Kernel 2 shown in figure 13, we got blurry image but with different color as shown below;

Figure 13: Output of Average Filter using kernel-2

4.4 TASK - Sobel Filter

The sobel filter using in image processing is usually particularly within edge detection algorithms where it creates an image emphasising edges. By applying the sobel filter to the image, the gradients of the image can be obtained so that the edges can be observed and detected. The kernel function is showing as follows:

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Figure 14: Sobel Filter Kernels – For Horizontal detection – For Vertical Detection

4.4.1 Problems Faced

We applied first the vertical kernel and then the horizontal kernel we got two outputs. But at first we got very bad results as shown in Figure 15:

Figure 15: Output of Sobel Filter at Starting

So, to solve this problem, we just added “typecast” to the function in matlab before conversion. And the output was improved, the same can be seen in the results.

4.4.2 Results

We applied the vertical kernel to detect the vertical edges of the image and we got the output as shown in figure 16:

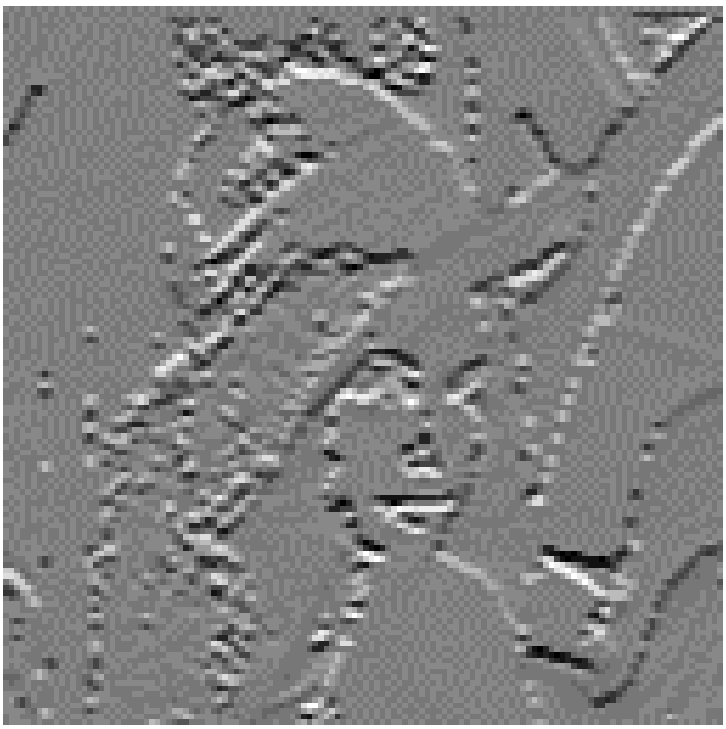

Figure 16: Vertical edge detection

Then we applied horizontal kernel to detect the horizontal edges of the image and we got the output as shown in figure 17:

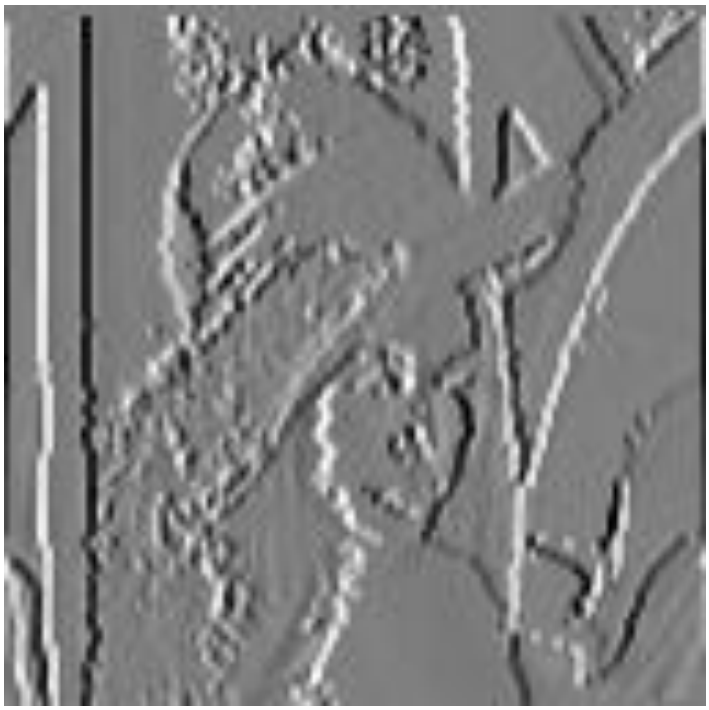

Figure 17: Horizontal Edge Detection

Then we added both the images to combine the result which is shown below:

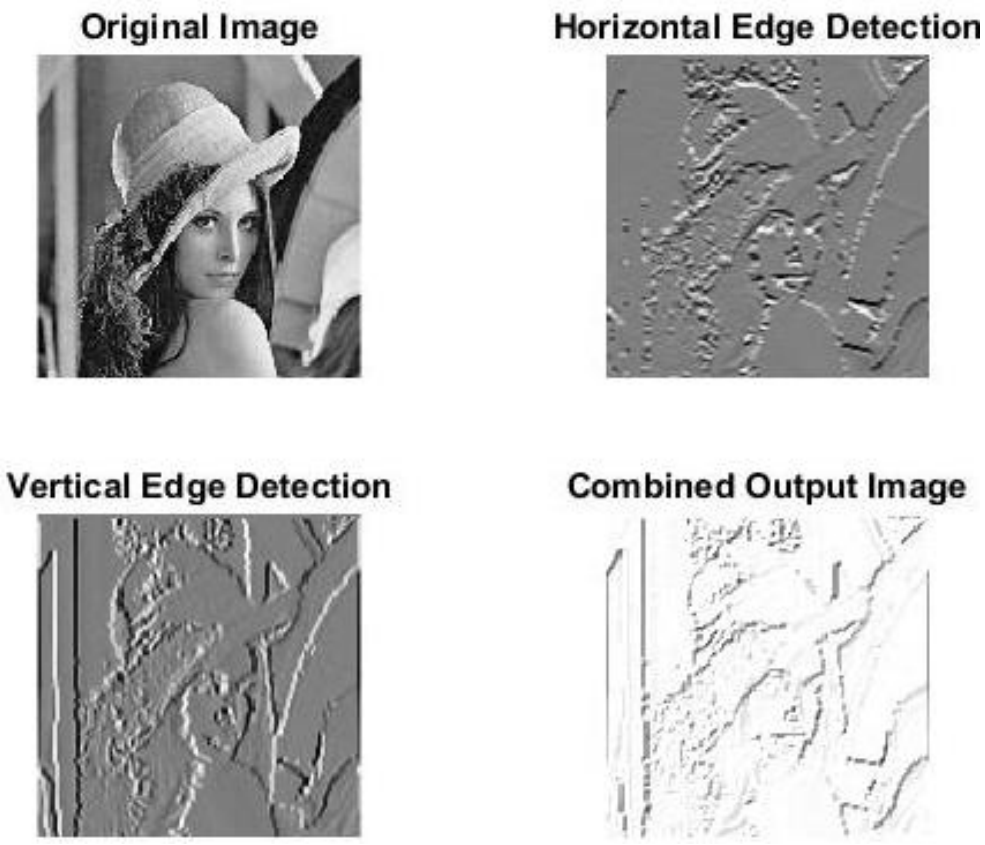

Figure 18: Combined Result of Vertical as well as Horizontal Image

4.5 TASK - Gaussian Filter

In image processing, an image can be smoothed by applying the Gaussian filter. Gaussian filter is a widely used effect in graphics software, typically to reduce image noise and reduce detail. It has the effect of reducing the image's high-frequency components. Gaussian filter is thus a low pass filter.

In our code, we used a kernel of Gaussian filter which is shown below:

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

Matrix 2: Gaussian filter kernel

4.5.1 Problems faced

Though the Gaussian filter is a LPF, which means its output should be similar to Average filter, but the problem faced in average filter result repeated here too. Maybe there is a problem in the divider or somewhere in computation, we couldn't find it. But the result we got is a blurry image which means that the Gaussian filter has been applied but the colours are not the same as it can be seen in figure 19.

4.5.2 Results

The result of Gaussian filter is shown in figure 19. Though the colours are not the same but it can be clearly seen in the Image that there is a blurry effect which means Gaussian filter has been applied to it.

Figure 19: Gaussian filter output

5. Test Bench

After we finish our cache memory and processing unit. The design need to be tested through testbench. The testbench is working like a black box testing and used for making sure it works as perfect as specified functionality. Using a testbench, we can pass inputs of our choice to the design to be tested. The outputs coming out of our design can be viewed on a simulation waveform or text file.

Firstly the components of cache memory and processing unit are instantiated. Then in the main process, the image is read as a signal from local text file. The signal is passed into cache memory and the output is taken to the processing function directly. The result from processing unit is transferred from type signal to variable and then written to a new local text file which is representing the filtered image data.

The process done in test bench is:

1. We are merging both the files; Cache_Memory.vhd as well as FILTER_PROCESS.vhd inside the test bench.

2. We are Reading the Lena Image in test bench.

3. We are calling the filter part as well as the cache memory part to filter out the image.

4. We are writing the output of the filter onto a new image named “Output.dat”.

   And as soon as the filtering is done, it will stop the processing.

6. Results

All of the results have been shown in each result section, so here I would like to conclude the results of all the methods we applied to solve the problem statement. As explained before, the result of average filter is good by applying process-1 but if generically seen, process-2 can be used to apply any filter to the image and we can get the output. We just need to find the problem in the code.

Result of Process-1 for average filter:

Figure 20: Result of process 1 in average filter

Result of Process - : Generalized Process

Figure 21: Result of process 2 in average filter with Kernel - 1

Figure 22: Result of process 2 in average filter with Kernel – 2

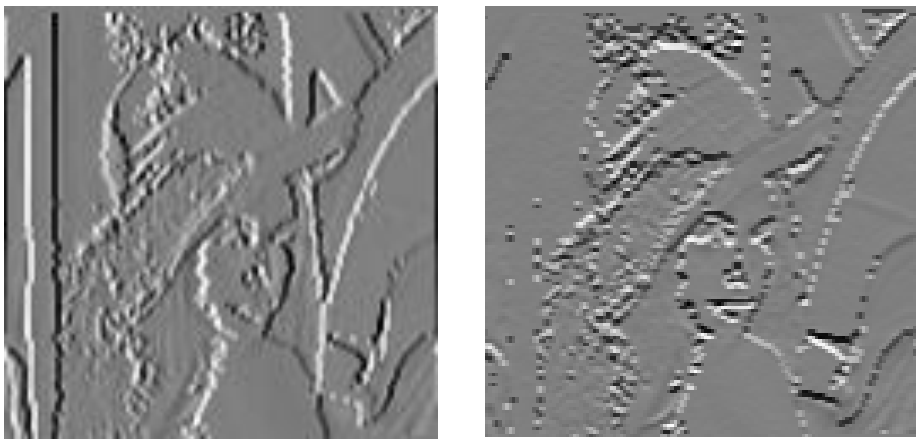

Figure 21: Result of process 2 in Sobel filter with Horizontal and vertical edge detection

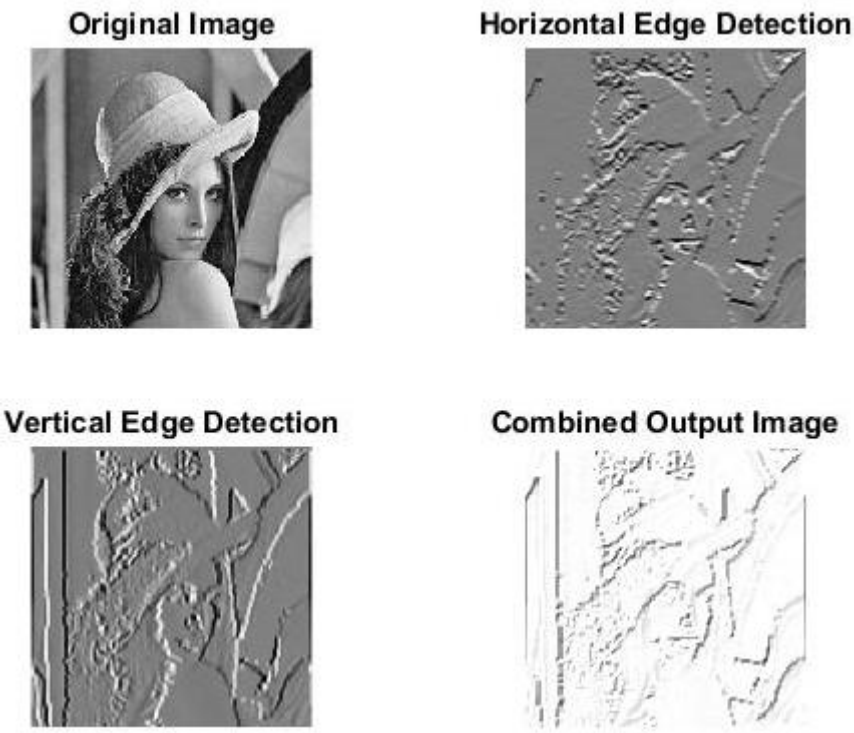

Figure 22: Result of process 2 in Sobel filter with Horizontal and vertical edge detection

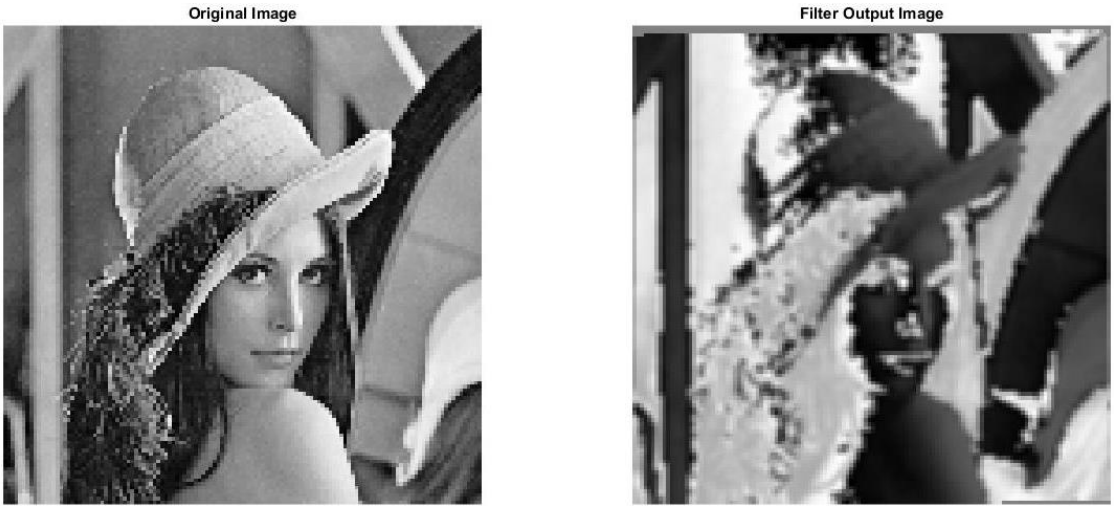

Figure 23: Result of process 2 in Gaussian filter

7. Conclusion

In this Project, we learned how to use FPGA boards and how to code in VHDL so as various operations like filtering can be done onboard. We performed filtering of image using VHDL language. We designed three different type of filters; averaging filter with two different kernels, Sobel edge detection filter and Gaussian filter. So, we can work on FPGA's in future to reduce the computing power required by computers to process a lot of data.

As the performance required for image processing applications is continuously increasing the demands on computing power, especially when there are real time constraints. Image processing applications may consist of several low level algorithms applied in a processing chain to a stream of input images. In order to accelerate image processing, there are different alternatives ranging from parallel computers to specialized Application Specific Integrated Circuits (ASIC) architectures.

The computing paradigm using reconfigurable architectures based on Field Programmable Gate Arrays (FPGAs) promises an intermediate trade-off between flexibility and performance and can be used for many purposes like we have explained in this report too. It can be used for filtering images etc. etc.