Implementation of Linear Spatial Filter in Video Stream Using FPGA Hardware Accelerator

SULAEMAN
Hasanuddin University
sulaeman16h@student.unhas.ac.id

Abstract

Various kinds of accelerators have been developed to improve performance and energy efficiency to handle heavy computations, one of which is FPGA. FPGA is capable of handling such a heavy computational load that it can be used for Digital Signal Processing, Image Processing, Neural Networks, etc. In this study, the authors tried to examine the performance of the ARM processor and the FPGA on the Xilin PYNQ Z2 FPGA Development Board in applying a linear spatial filter to the video stream. Kernel filters used in this study are the average blur, Gaussian blur, Laplacian, sharpen, Sobel horizontal, and Sobel vertical. The parameters used to measure the performance of ARM processors and FPGAs are runtime, frame rate (FPS), CPU usage, memory usage, resident memory (RES), shared memory (SHR), and virtual memory (VIRT). The average computation time required to apply linear spatial filters to 200 frames with an ARM processor is 29.06 seconds, while the average FPGA takes only 3.32 seconds. Compute time with FPGA is 88.85% better than ARM processor. The filtered video with the ARM processor gets an average of 6.95 fps while the FPGA average is 60.37 fps. FPS with FPGA is 88.49% better than ARM processor. CPU usage on FPGA is 14.89% better, memory usage on FPGA is 2.02% better, usage of resident memory is 2.07% better, and usage of shared memory is 4.08% better than ARM processor. While the use of virtual memory on ARM processors is 0.03% better than FPGA.

Keywords : *filter spasial linear, FPGA, ARM prosesor, video stream, video processing*

1. INTRODUCTION

To improve the performance and energy efficiency of a program, various types of accelerators have been developed, one of which is FPGA [3]. *Field Programmable Gate Arrays* or FPGA is a semiconductor device based on matrix configurable logic block (CLBs) that connected via programmable interconnects. FPGAs can be reprogrammed with the desired application or function after manufacturing. This feature distinguishes FPGAs from Application Specific Integrated Circuits (ASICs), which are specially made for a specific task [14].

FPGAs have demonstrated very high performance in many applications in image processing. However the latest CPUs and CPUs have high performance potential for these issues. The latest CPUs support multi-cores, where each core supports SIMD (Single Instruction, Multiple Data) which has been developed and runs up to 16 operations on 128 bits of data in one clock cycle. The latest GPUs support a large number of cores running in parallel, and their peak performance outperforms the CPU [1].

The parallelism in SIMD on the CPU is limited, but the operating frequency of the CPU is very high, and the CPU is expected to show high performance in applications where cache memory is running well. The cache memory size is large enough to store entire images in many image processing applications, and the CPU can run the same algorithms as the FPGA even though the required high memory bandwidth [1].

GPU operating frequency is faster than FPGA, but slightly slower than CPU. However, the GPU supports multiple cores running in parallel so that its peak performance outperforms the CPU. However the cores are grouped, and data transfer between groups is extremely slow. In addition, the size of the local memory provided by each group is very small. Due to this limitation, the GPU cannot run the same algorithms as the FPGA in some application issues [1].

Most FPGAs are assembled with a single processor board, often referred to as the FPGA Development Board. Xilinx PYNQ-Z2 is built from ARM Cortex-A9 processor, so it can run some software like python without having to design the circuit from scratch. However, the performance of the ARM processor on the FPGA Development Board is certainly different from the performance of the FPGA architecture function itself so that it can be studied more deeply regarding to the performance comparison.

2. Fundamental Concepts

2.1. Digital Image Processing

Digital image processing is the process of digitally processing the pixels of the image for a specific purpose. Based on the level of confidence, digital image processing is grouped into three categories, low-level, mid-level and high-level. low-level processing is performed with primitive operations such as image preprocessing to reduce noise (noise), improve image contrast and sharpen images (sharpening). The criterion of low-level is the correct output or result in the form of a digital image. Mid-level processing involves tasks such as segmentation (partitioning an image into sections or objects), description of objects to be performed by sequence, and classification of objects in digital images. The criterion of mid-level is the output or result in the form of attributes or features such as contours, edges, or objects contained in the image. High-level processing is an advanced process from the previous two processes, carried out to obtain more information contained in the image [4].

2.2. Spatial Filter

The concept of spatial filter in digital image processing comes from the application of the Fourier transform of signal processing in the frequency domain. The term spatial filter is used to distinguish this process from filters in the frequency domain. The filter process is done by shifting the kernel filter from point to point in the digital image. The terms mask, kernel, template, and window are the same terms and are often used in digital image processing [4]. The researchers used term kernel for the term.

2.3. Kernels

2.3.1. Average Blur

Average blur also known as box filter is one of the filters used to smooth the image and reduce noise. In simple terms, the value of a new pixel is the average value of the pixel value with its neighboring pixel value [7]. The following is the Average blur kernel used:

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

2.3.2. Gaussian Blur

It is also used to smooth the image and reduce noise. The idea is similar to Average blur, the new pixel value is constructed from the neighboring pixel values, but by giving a stronger weight to the pixel value itself followed by a lower weight on the top, bottom and side pixels [13]. The following is the kernel gaussian blur filter used:

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

2.3.3. Sobel

The Sobel filter is a high-pass filter which is commonly used for edge detection in images. Sobel has two kernels for edge detection namely a vertical sobel kernel for verticaly edge detection and a horizontal sobel kernel for horizontaly edge detection [7]. The following are the vertical and horizontal Sobel kernels used:

$$\begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix} \qquad \begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix} \tag{3}$$

2.3.4. Laplacian

This filter can be used for edge detection in images because of its sensitive with rapid changes in intensity [5]. Unlike Sobel which uses two kernels to detect edges vertically and horizontally, it is only used a kernel which can be used for both vertical and horizontal edge detection. The following are the Laplacian kernels used:

$$\begin{bmatrix}0 & 1 & 0\\ 1 & -4 & 1\\ 0 & 1 & 0\end{bmatrix} \tag{4}$$

2.3.5. Sharpening

Sharpening filters are used to clarify fine details in an image or to enhance details in an image that is blur, either by mistake or due to the effect of certain image acquisition methods [15]. The following is the kernel for the sharpening filter used:

$$\begin{bmatrix}-1 & -1 & -1\\ -1 & 9 & -1\\ -1 & -1 & -1\end{bmatrix} \tag{5}$$

2.4. Konvolusi

The concept of a linear spatial filter is similar to the concept of convolution in the frequency domain, for that reason a linear spatial filter is also called the convolution of a kernel with a digital image [4]. The convolutions of the functions *f (x)* and *g (x)* are defined in the equation 6.

$$h(x) = f(x) * g(x) = \int_{-\infty}^{\infty} f(a)g(x-a)da \quad (6)$$

where the * sign represents the convolution operator, and the variable a is the auxiliary variable. For discrete functions, convolution is defined in the equation 7.

$$h(x) = f(x) * g(x) = \sum_{a=-\infty}^{\infty} f(a)g(x-a) \quad (7)$$

In the convolution operation, *g (x)* is called a convolutional kernel or kernel filter. Kernel *g (x)* is operated shift in input signal *f (x)*. The sum of the multiplication of the two functions at each point is the result otf a convolution which is represented by the output *h (x)* [9].

Convolution with the 7 equation only applies to 1-dimensional input, while for 2-dimensional input such as images, the 8 equation can be used.

$$h(x,y) = (I * K)(x,y) = \sum_{m}\sum_{n} I(m,n)K(x-m, y-n) \quad (8)$$

Information:

h(x,y)	= function of the convolution
I	= input
K	= convolution kernel
x,y	= pixel input
m,n	= pixel kernel

In the figure (1) it illustrates how the convolution process in a digital image is represented in the form of a matrix. Convolution operations are performed on a 6x6 size input matrix with a 3x3 filter. The result of the convolution is displayed on the result matrix.

If the convolution results in a negative pixel value, then the value is set to 0, on the other hand, if the convolution results in a pixel value that exceeds the maximum gray value, then the value is turned into the maximum gray value of the image [12].

2.5. Video Stream

Video stream can be viewed as a series of consecutive digital images [16]. Unlike other video formats, this stream video is not stored on the storage media as a video file format but is directly transmitted each frame from the source to the receiver, in this case the FPGA. By assuming Video stream is a collection of digital images (frame), processing methods such as digital images can be carried out, including the application of spatial filters.

2.6. FPGA

Field Programmable Gate Arrays or FPGAs are semiconductor devices based on matrix configurable logic blocks (CLBs) connected via programmable interconnects. The FPGA can be reprogrammed to the desired application or function after the manufacturing. This feature is what differentiates FPGAs from the Application Specific Integrated Circuits (ASICs), which are specially made for a specific task [14].

A microprocessor receives instructions in the form of 1 or 0 code, these codes are then interpreted by the computer to execute the given command. The Microprocessor requires instructions in the form of code continuously to carry out its function. Whereas in FPGA, it only takes to configuration once every time it is powered on. Creating or downloading bitstream which defines logical functions is performed by logic elements (LEs), a circuit can be created by combining multiple LEs into a single unit. After bitstream is installed, the FPGA no longer needs to read instructions in the form of 1 and 0, in contrast to microprocessor which always requires the instruction [2]. Traditionally, to create an FPGA design, applications are described using a Hardware Description Language (HDL) such as Verilog or VHDL so as to produce a bitstream FPGA.

2.7. FPGA Development Board

FPGA Development Board or commonly known as FPGA Board, namely FPGA technology that is assembled in a board and is equipped with microprocessor and several interface IO for carry out certain tasks. Generally, FPGA Board is equipped with an interface for accessing and implementing the circuit design. Xilinx, Altera and Intel are well-known FPGA Board manufacturers. The FPGA Board used is the Xilinx PYNQ-Z2 with Jupyter Notebook as the interface to access and run the program in this study. Form of FPGA Board Xilinx PYNQ-Z2 can be seen in figure (2)

2.8. Performance Evaluatinon

The researchers used computation time, frame rate (FPS), CPU usage, memory usage, resident memory (RES), shared memory (SHR), and virtual memory

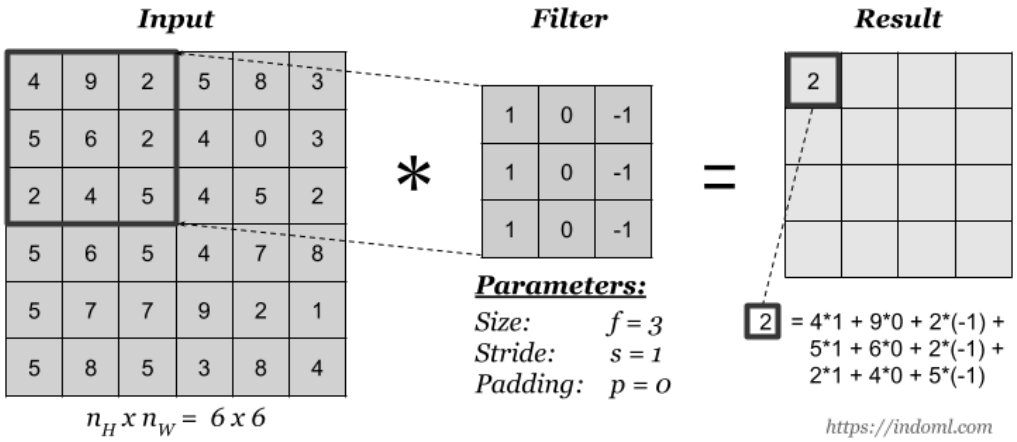

Figure 1: *Convolution in digital image.*

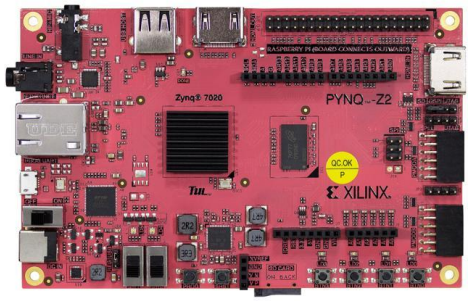

Figure 2: *FPGA Board Xilinx PYNQ-Z2.*

usage (VIRT) to measure the performance of applying linear spatial filters with ARM and FPGA processors.

2.8.1. Computation Time

The computation time is the duration it takes for a kernel to perform a linear spatial filter on multiple frame inputs. This computation time is obtained by calculating the difference between the finishing time and the starting time of applying linear spatial filters to the input frame.

$$computation\ time = finishing\ time - starting\ time \tag{9}$$

2.8.2. Frame Rate (FPS)

Frame rate or frames per second (fps) is the number of frames displayed per second on a video or video straem. The higher the fps of a video, the smoother the result. On the other hand, videos with low fps will result in poor motion. Frame rate or fps can be calculated by dividing the number of frames by the computation time as in the equation 10 [8].

$$fps = \frac{number\ of\ frame}{computation\ time} \tag{10}$$

2.8.3. CPU Usage

The researchers used the features available on the Linux operating system running on the FPGA Development Board to see the percentage of CPU usage in the process of applying linear spatial filters. This program displays information about processes running on the operating system such as the ID of a process, the user running the process, the memory used, the status of a process, the percentage of CPU used and others.

2.8.4. Memory Usage

On the linux operating system memory is divided into three types [6]. The first is the physical memory, a finite resource that code and data must reside in when executed or referenced. Next is memory swap, which is memory that useful for helping physical memory work, data from physical memory will be stored in swap and then retrieved return if too many requests to physical memory. The third is virtual memory, an almost unlimited resource used for the following purposes [11]:

- *abstraction*, free of physical memory addresses / boundaries
- *isolation*, each process in a separate address space
- *sharing*, a single map can meet many needs
- *flexibility*, assign virtual addresses to data

2.8.5. Virtual Memory (VIRT)

Virtual memory uses disk as an extension of RAM so that the effective size of the usable memory increases simultaneously. The kernel will write the contents of the currently unused memory block to the hard disk so that memory can be used for other purposes. When the original content is needed again, they are read back into memory. These are all made completely transparent to the user. Programs running on Linux only see the larger amount of memory available and do not notice that some of them are on disk over time. Of course, reading and writing a hard disk is slower than using physical memory, so programs don't run that fast. The part of the hard disk that is used as virtual memory is called swap space [10].

2.8.6. Resident Memory (RES)

Resident memory is the portion of the virtual address space (VIRT) that represents the unchanged physical memory that the task is currently running. Resident memory is also the sum of the RSan, RSfd and RSsh fields. This can include private anonymous pages, private pages mapped to files (including program images and shared libraries) plus shared anonymous pages. All such memory is supported by a swap file which is represented separately on SWAP. This resident memory can also include a supported pages shared file-backed which, when modified, acts as a custom swap file and therefore never affects SWAP [6].

2.8.7. Shared Memory (SHR)

Shared memory is a part of resident memory (RES) that can be used by other processes. Includes anonymous pages and shared file-backed pages. This also includes private pages mapped to files representing program images and shared libraries [6].

3. Tools and Methods

3.1. Sistem Design

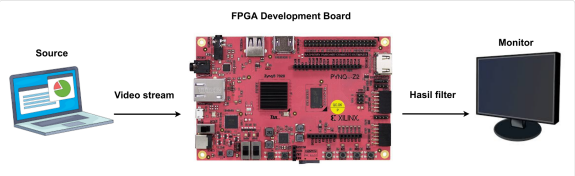

Figure 3: *Sistem Design.*

Video stream from the source is channeled through the HDMI Input port on the FPGA Development Board, then the video stream will be processed by applying a linear spatial filter to each frame. Each frame that has been applied the spatial filter will be streamed to the monitor to display. Furthermore, the FPGA performance analysis is carried out. The FPGA Development Board used can be accessed with ssh protocol on port 22 or by Jupyter Notebook via web browser.

3.2. Tools

1. Software requirements:
   a. Linux Ubuntu 18, Operation sistem on FPGA Development Board.
   b. Python 3.6, library OpenCV, Numpy, Pynq 5.2, and Xilinx xfOpenCV.
   c. Jupyter Notebook on FPGA Development Board.
   d. Web Browser.
2. Hardware requirements:
   a. FPGA Development Board.
   b. Micro SD Card 16 GB.
   c. Eksternal Monitor, to display the results of applying spatial filters on the FPGA Development Board.
   d. Lenovo Ideapad 320 as *source* of video stream).

   FPGA Development Board specification:
   - Model : Xilinx PYNQ-Z2.
   - Processor : Dual-Core ARM Cortex A9, 650 MHz
   - FPGA : 1,3M reconfigurable gates
   - Memory : 512 MB DDR3 / Flash
   - Storage : Micro SD card slot
   - Power : DC 7V-15V
   - Dimension : 3,44" x 5,39" (87mm x 137mm)

3.3. Methods

The spatial filter process is carried out by convolution operations on each matrix with a predetermined kernel. This convolutional operation produces a new matrix with a size of 1280x720. The resulting matrix is then represented as a digital image, hereinafter referred to as the filter result. The filter result of each frame is displayed to the monitor via HDMI Output on the FPGA Development Board continuously.

Figure 5: *Result of Average Blur filter.*

Figure 6: *Result of Gaussian Blur filter.*

Figure 4: *Grayscale Frame.*

3.3.1. Average Blur

Applying a spatial filter to grayscale frame which is 1280x720 pixels using a kernel average blur (1) measuring 3x3 produces a blur image that is 1280x720 too. The results of the average blur filter can be seen in the 5 image. Filters like this can be used to reduce noise in the image.

3.3.2. Gaussian Blur

The application of a spatial filter with a 3x3 kernel gaussian blur (2) produces a blur image which is visually similar to the average blur filter. However, if you pay attention to the value of each pixel in the figure 6 it will look different from the value of each pixel in the figure 5. This is because the weight value in the gaussian blur kernel is different from the average blur kernel, so the result of the convolution is also different.

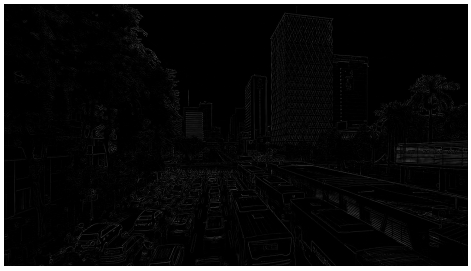

Figure 7: *Result of Laplacian filter.*

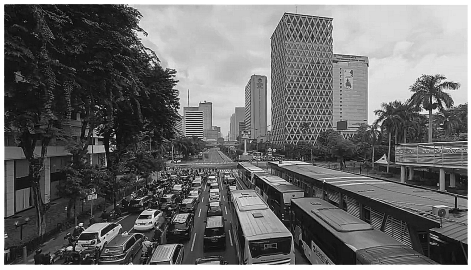

Figure 8: *Result of Sharpening filter.*

3.3.3. Laplacian

Application of spatial filter with kernel laplacian (4) produces binary image which is represented only in black and white, can be seen in figure 7. Filters like this can be used in edge detection methods in digital image processing.

3.3.4. Sharpening

Applying a spatial filter with kernel sharpening (5) can increase detail (such as lines) in the image, but can also cause noise in the image if the kernel weight is not appropriate. Filters like this are more appropriate to improve image quality (with the appropriate kernel values). The results of the sharpening filter can be seen in the 8 image.

3.3.5. Sobel Horizontal

Application of a spatial filter with kernel sobel horizontal (3) produces a binary image, seen in figure 9. Such filters are more appropriate for edge detection methods with images that contain lots of horizontal lines.

3.3.6. Sobel Vertical

Application of a spatial filter with kernel sobel vertical (3) produces a binary image, seen in figure 10. Similar to the sobel horizontal filter, the sobel vertical filter can

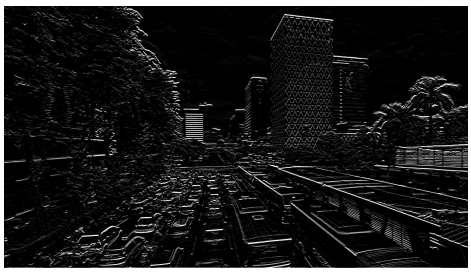

Figure 9: *Result of Sobel Horizontal filter.*

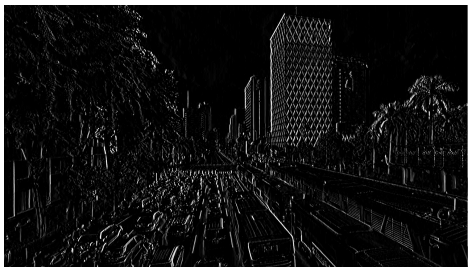

Figure 10: *Result of Sobel Vertical filter.*

also be used for edge detection methods, especially on images that contain lots of vertical lines.

4. Results and Discussions

4.1. Computation Time

The computation time using 50 frames in each kernel can be seen in the **??** table and the chart in figure 11. In general, the computation time using an ARM processor is slower than FPGA. The average computation time with an ARM processor using 50 frames was 7.26 seconds, while the average computation time with an FPGA using 50 frames was only 0.82 seconds.

Table 1: *Comparison table of Computing time using 50 frames.*

Filter	Prosesor ARM (s)	FPGA (s)
Average Blur	8.003612137	0.823560476
Gaussian Blur	7.998623228	0.827384996
Laplacian	6.210968781	0.827101517
Sharpening	8.041392469	0.825223923
Sobel Horizontal	6.646468353	0.823914003
Sobel Vertical	6.696593809	0.823559713
Rata-rata	7.266276463	0.825124105

The computation time data using 200 frames in each kernel can be seen in the table **??** and the chart in

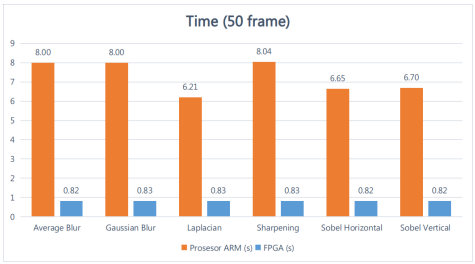

Figure 11: *Comparison chart of Computing time using 50 frames*

figure 12. The average computation time with an ARM processor using 200 frames was 29.06 seconds, while the average computation time with an FPGA using 200 frames was only 3.32 seconds.

Table 2: *Comparison table of Computing time using 200 frames.*

Filter	Prosesor ARM (s)	FPGA (s)
Average Blur	31.9899159	3.325525188
Gaussian Blur	31.97958055	3.325351763
Laplacian	24.85993662	3.323753452
Sharpening	32.24906039	3.328786802
Sobel Horizontal	26.49739237	3.32414484
Sobel Vertical	26.84196448	3.324060822
Rata-rata	29.06964172	3.325270478

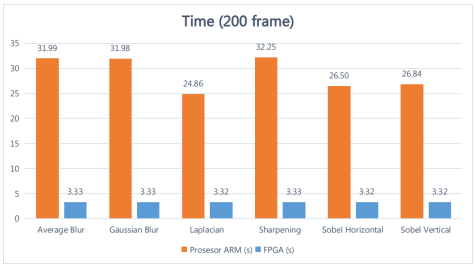

Figure 12: *Comparison chart of Computing time using 200 frames.*

The fastest computation time using an ARM processor is the laplacian filter, namely 6.21 seconds with 50 frames and 24.85 seconds with 200 frames. Meanwhile, the slowest computation time when using an ARM processor is the sharpening filter, namely 8.04 seconds with 50 frames and 32.24 seconds with 200 frames.

To calculate the computation time efficiency of an FPGA compared to an ARM processor, the formula is

used:

$$= 100\% - \left(\frac{computation\ time\ FPGA}{computation\ time\ ARM Prosesor} \times 100\% \right)$$
$$= 100\% - \left(\frac{3,32}{29,06} \times 100\% \right)$$
$$= 100\% - 11.42\%$$
$$= 88.58\%$$

the efficiency of FPGA computation time compared to ARM processors is 88.85%.

4.2. Frame Rate (FPS)

By the computation time and the number of frames, the frame rate or FPS can be calculated using the 10 equation. FPS data for each kernel with ARM and FPGA processors can be seen in the **??** table and the chart in the figure 13.

Table 3: *Comparison table of FPS using ARM prosesor and FPGA.*

Filter	Prosesor ARM	FPGA
Average Blur	6.249583533	60.42684593
Gaussian Blur	6.25253493	60.28874396
Laplacian	8.047839131	60.31306253
Sharpening	6.209782985	60.33633034
Sobel Horizontal	7.53538058	60.42633377
Sobel Vertical	7.459019618	60.44047449
Rata-rata	6.959023463	60.37196517

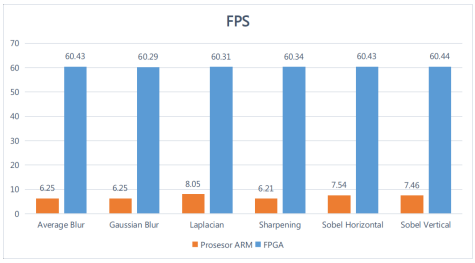

Figure 13: *Comparison chart of FPS using ARM prosesor and FPGA.*

In the table **??**, it can be seen that using an ARM processor the average is 6.95 frames per second (FPS) was obtained, while using an FPGA the average is 60.37 frames per second was obtained. As seen in the chart 13 the FPS value with FPGA is much higher than with an ARM processor.

To calculate the FPS efficiency that the FPGA has compared to an ARM processor, the formula is used:

$$= 100\% - \left(\frac{FPS\ ARM Prosesor}{FPS\ FPGA} \times 100\% \right)$$
$$= 100\% - \left(\frac{6.95}{60.37} \times 100\% \right)$$
$$= 100\% - 11.51\%$$
$$= 88.49\%$$

the FPS efficiency of the FPGA compared to the ARM processor is 88.49%.

4.3. Penggunaan CPU

Data perbandingan penggunaan CPU pada masing-masing kernel dengan prosesor ARM dan FPGA dapat dilihat pada tabel 4 dan grafik pada gambar 14. Rata-rata penggunaan CPU dengan prosesor ARM adalah 99.58% sedangkan dengan FPGA diperoleh 84.75%. Data ini menunjukkan bahwa penggunaan CPU dengan prosesor ARM sedikit lebih besar daripada dengan FPGA.

Table 4: *Tabel perbandingan penggunaan CPU dengan menggunakan prosesor ARM dan FPGA.*

Filter	Prosesor ARM (%)	FPGA (%)
Average Blur	99.83	83.86
Gaussian Blur	99.60	83.70
Laplacian	99.53	84.02
Sharpening	99.47	84.91
Sobel Horizontal	99.47	86.22
Sobel Vertical	99.57	85.80
Rata-rata	99.58	84.75

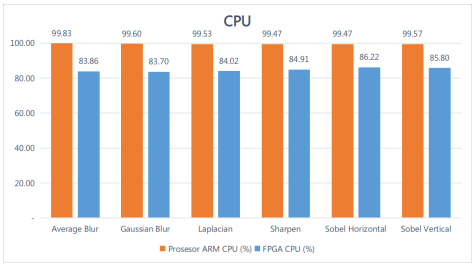

Figure 14: *Grafik perbandingan penggunaan CPU dengan menggunakan prosesor ARM dan FPGA.*

Penggunaan CPU terbesar dengan prosesor ARM yaitu pada kernel *average blur* (99,83%) dan dengan FPGA pada kernel *sobel horizontal* (86,22%).

Penggunaan CPU terkecil dengan prosesor ARM yaitu pada kernel *sharpening* dan *sobel horizontal* (99,47%) dan dengan FPGA pada kernel *gaussian blur* (83,70%).

Untuk menghitung efisiensi penggunaan CPU yang dimiliki FPGA dibandingkan dengan prosesor ARM, digunakan persamaan berikut:

$$
\begin{aligned}
&= 100\% - \left(\frac{CPU\ FPGA}{CPU\ ARM Prosesor} \times 100\%\right) \\
&= 100\% - \left(\frac{84.75}{99.58} \times 100\%\right) \\
&= 100\% - 85.11\% \\
&= 14.89\%
\end{aligned}
$$

sehingga diperoleh efisiensi penggunaan CPU FPGA dibandingkan dengan prosesor ARM adalah sebesar 14.89%.

4.4. Penggunaan Memory

Data penggunaan *memory* dengan prosesor ARM dan FPGA dapat dilihat pada tabel 5 dan grafik pada gambar 15. Data ini menunjukkan persentase *memory* yang digunakan pada masing-masing kernel. Rata-rata penggunaan *memory* dengan prosesor ARM adalah 25,37% dan 24,86% dengan FPGA.

Table 5: *Tabel perbandingan penggunaan memory dengan menggunakan prosesor ARM dan FPGA.*

Filter	Prosesor ARM (%)	FPGA (%)
Average Blur	25.50	24.80
Gaussian Blur	25.88	24.98
Laplacian	25.06	24.58
Sharpening	25.40	25.14
Sobel Horizontal	25.12	24.68
Sobel Vertical	25.28	24.98
Rata-rata	25.37	24.86

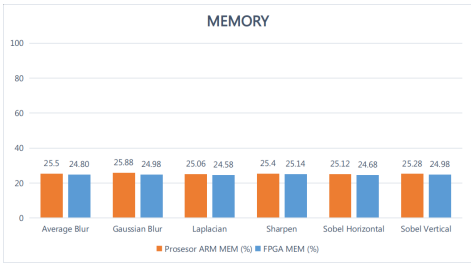

Figure 15: *Grafik perbandingan penggunaan memory dengan menggunakan prosesor ARM dan FPGA.*

Walaupun FPGA lebih baik daripada prosesor ARM pada segi waktu komputasi dan FPS namun penggunaan *memory* pada penerapan filter ini terlihat tidak jauh berbeda. Penggunaan *memory* FPGA ini hanya 0,51% lebih rendah dari penggunaan *memory* dengan prosesor ARM.

Efisiensi penggunaan *memory* yang dimiliki FPGA dibandingkan dengan prosesor ARM, dapat dihitung dengan persamaan berikut:

$$
\begin{aligned}
&= 100\% - \left(\frac{memory\ FPGA}{memory\ ARM Prosesor} \times 100\%\right) \\
&= 100\% - \left(\frac{24.86}{25.37} \times 100\%\right) \\
&= 100\% - 97.98\% \\
&= 2.02\%
\end{aligned}
$$

diperoleh efisiensi penggunaan *memory* FPGA dibandingkan dengan prosesor ARM adalah sebesar 2.02%.

4.5. Resident Memory (RES)

Data penggunaan *resident memory* atau RES dengan prosesor ARM dan FPGA dapat dilihat pada tabel 6 dan grafik pada gambar 16. Data ini menunjukkan banyaknya RES (dalam satuan *kilobyte*) yang digunakan pada saat penerapan filter spasial pada video *stream* dengan masing-masing kernel.

Table 6: *Tabel perbandingan penggunaan resident memory (RES) dengan menggunakan prosesor ARM dan FPGA.*

Filter	Prosesor ARM (KiB)	FPGA (KiB)
Average Blur	129794.80	126097.60
Gaussian Blur	131804.40	127135.20
Laplacian	127493.60	125005.60
Sharpening	129117.22	127931.20
Sobel Horizontal	127830.40	125420.00
Sobel Vertical	128611.20	127033.60
Rata-rata	129108.60	126437.20

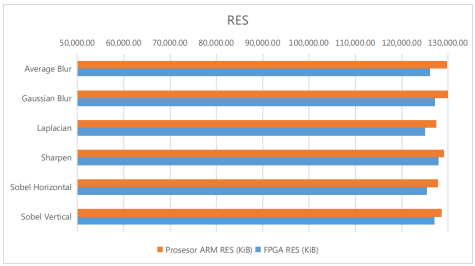

Figure 16: *Grafik perbandingan penggunaan resident memory (RES) dengan menggunakan prosesor ARM dan FPGA.*

Rata-rata RES yang digunakan pada prosesor ARM adalah 129108,60 KiB dan 126437,20 KiB pada FPGA. Terlihat bahwa penggunaan RES pada prosesor ARM dan FPGA juga tidak jauh berbeda. Penggunaan RES terbesar dengan prosesor ARM yaitu pada kernel *gaussian blur* 131804,40 KiB, sedangkan dengan FPGA yaitu pada kernel *sharpening* 127931,20 KiB.

Efisiensi penggunaan *resident memory* yang dimiliki FPGA dibandingkan dengan prosesor ARM, dapat dihitung dengan persamaan berikut:

$$= 100\% - \left(\frac{resident\ memory\ FPGA}{resident\ memory\ ARM Prosesor} \times 100\% \right)$$
$$= 100\% - \left(\frac{126437.20}{129108.60} \times 100\% \right)$$
$$= 100\% - 97.93\%$$
$$= 2.07\%$$

diperoleh efisiensi penggunaan *resident memory* FPGA dibandingkan dengan prosesor ARM adalah sebesar 2.07%.

4.6. Shared Memory (SHR)

Data penggunaan *shared memory* dengan prosesor ARM dan FPGA dapat dilihat pada tabel 7 dan grafik pada gambar 17. Data ini menunjukkan banyaknya *shared memory* (dalam satuan *kilobyte*) yang digunakan pada saat penerapan filter spasial pada video *stream* dengan masing-masing kernel. Rata-rata penggunaan *shared memory* pada prosesor ARM adalah 56325,57 KiB dan 54030,80 KiB pada FPGA. Terlihat bahwa penggunaan *shared memory* pada prosesor ARM sedikit lebih besar daripada FPGA. Penggunaan *shared memory* terbesar pada prosesor ARM yaitu pada kernel *gaussian blur* 56503,60 KiB dan kernel *laplacian* 55528,80 KiB pada FPGA. Penggunaan *shared memory* terkecil pada prosesor ARM yaitu pada kernel *average blur* 56157,40 KiB dan kernel *laplacian* 42568 KiB pada FPGA.

Table 7: *Tabel perbandingan penggunaan shared memory (SHR) dengan menggunakan prosesor ARM dan FPGA.*

Filter	Prosesor ARM (KiB)	FPGA (KiB)
Average Blur	56,157.40	53,840.00
Gaussian Blur	56,503.60	54,504.80
Laplacian	56,248.00	52,568.00
Sharpen	56,298.82	55,528.80
Sobel Horizontal	56,349.60	53,015.20
Sobel Vertical	56,396.00	54,728.00
Rata-rata	56,325.57	54,030.80

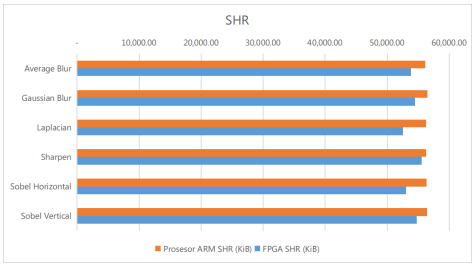

Figure 17: *Grafik perbandingan penggunaan shared memory (SHR) dengan menggunakan prosesor ARM dan FPGA.*

Efisiensi penggunaan *shared memory* yang dimiliki FPGA dibandingkan dengan prosesor ARM, dapat dihitung dengan persamaan berikut:

$$= 100\% - \left(\frac{shared\ memory\ FPGA}{shared\ memory\ ARM Prosesor} \times 100\% \right)$$
$$= 100\% - \left(\frac{54030.80}{56325.57} \times 100\% \right)$$
$$= 100\% - 95.92\%$$
$$= 4.08\%$$

diperoleh efisiensi penggunaan *shared memory* FPGA dibandingkan dengan prosesor ARM adalah sebesar 4.08%.

4.7. Virtual Memory (VIRT)

Data penggunaan *virtual memory* atau VIRT dapat dilihat pada tabel 8 dan grafik pada gambar 18. Data ini menunjukkan banyaknya *virtual memory* (dalam satuan *kilobyte*) yang digunakan pada saat penerapan filter spasial pada video *stream* dengan masing-masing kernel.

Table 8: *Tabel perbandingan penggunaan virtual memory (VIRT) dengan menggunakan prosesor ARM dan FPGA.*

Filter	Prosesor ARM (KiB)	FPGA (KiB)
Average Blur	396,474.64	395,396.00
Gaussian Blur	398,862.80	395,488.80
Laplacian	393,388.00	395,638.40
Sharpen	395,126.77	395,575.20
Sobel Horizontal	393,544.80	395,524.00
Sobel Vertical	395,048.80	395,385.60
Rata-rata	395,407.64	395,501.33

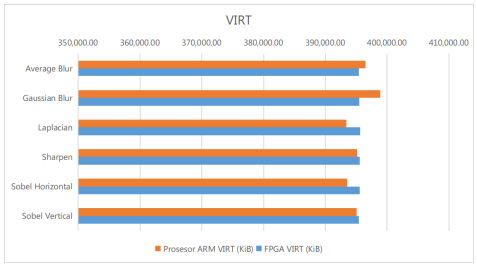

Figure 18: *Grafik perbandingan penggunaan virtual memory (VIRT) dengan menggunakan prosesor ARM dan FPGA.*

Rata-rata penggunaan VIRT pada prosesor ARM adalah 395407,64 KiB dan 395501,33 KiB pada FPGA. Rata-rata penggunaan VIRT pada FPGA sedikit lebih tinggi dari pada prosesor ARM. Penggunaan VIRT terbesar dengan prosesor ARM yaitu pada kernel *gaussian blur* 398862,80 KiB dan kernel *laplacian* 395638,40 KiB pada FPGA. Penggunaan VIRT terkecil dengan prosesor ARM yaitu pada kernel *laplacian* 393388,00 KiB dan kernel *sobel vertical* 395385,60 KiB pada FPGA.

Efisiensi penggunaan *virtual memory* yang dimiliki prosesor ARM dibandingkan dengan FPGA, dapat dihitung dengan persamaan berikut:

$$
\begin{aligned}
&= 100\% - \left(\frac{virtual\ memory\ ARM Prosessor}{virtual\ memory\ FPGA} \times 100\% \right) \\
&= 100\% - \left(\frac{395407.64}{395501.33} \times 100\% \right) \\
&= 100\% - 99.97\% \\
&= 0.03\%
\end{aligned}
$$

Ini menunjukkan bahwa penggunaan *virtual momory* pada prosesor ARM hanya 0.03% lebih baik dari penggunaan *virtual momory* pada FPGA.

5. Conclusion

Based on the results of implementation linear spatial filters on the FPGA Development Board using 6 kernels, researchers get the following conclusions:

1. The process of implementing a linear spatial filter on video stream with the FPGA Development Board is carried out with python's library OpenCV and Xilinx's library xfOpenCV. Each frame of video stream source is represented as a digital image then a linear spatial filter is applied, then the results of this filter are displayed continuously so that it looks like a video.
2. Compute time with FPGA 88.85% better than ARM processor. The filtered video with the ARM processor gets an average of 6.95 fps while the FPGA average is 60.37 fps. FPS with FPGA is 88.49% better than ARM processor. CPU usage on FPGA is 14.89% better, usage of memory on FPGA is 2.02% better, usage of resident memory is 2.07% better, and usage of shared memory is 4.08% better compared to ARM processors. Meanwhile, the use of virtual memory on ARM processors is 0.03% better than FPGA.

References

[1] Shuichi Asano, Tsutomu Maruyama, and Yoshiki Yamaguchi. "Performance comparison of FPGA, GPU and CPU in image processing". In: *2009 International Conference on Field Programmable Logic and Applications*. 2009, pp. 126–131. DOI: 10.1109/FPL.2009.5272532.

[2] Peter Cheung. "Introduction to FPGAs". http://www.ee.ic.ac.uk/pcheung/teaching/ee2_digital/Lecture2-IntroductiontoFPGAs.pdf. Accessed on 2020-04-19. 2019.

[3] Jason Cong dkk. "Understanding Performance Differences of FPGAs and GPUs". In: *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. FPGA '18. Monterey, CALIFORNIA, USA: Association for Computing Machinery, 2018, p. 288. ISBN: 9781450356145. DOI: 10.1145/3174243.3174970. URL: https://doi.org/10.1145/3174243.3174970.

[4] Rafael C. Gonzalez and Richard E. Woods. “Digital Image Processing”. 2nd. ISBN-13: 978-0201180756. Upper Saddle River, New Jersey 07458: Prentice Hall, 2001.

[5] Xu Jingbo dkk. “A New Method for Realizing LOG Filter in Image Edge Detection”. In: *The 6th International Forum on Strategic Technology* (Aug. 2011). DOI: 10.1109/IFOST.2011.6021127.

[6] Michael Kerrisk. “(Top) Linux Manual Page”. https://www.man7.org/linux/man-pages/man1/top.1.html. Accessed on 2021-02-2. 2020.

[7] Marcin Kowalczyk, Dominika Przewlocka, and Tomasz Krvjak. “Real-Time Implementation of Contextual Image Processing Operations for 4K Video Stream in Zynq UltraScale+ MPSoC”. In: *2018 Conference on Design and Architectures for Signal and Image Processing (DASIP)* (Oct. 2018). DOI: 10.1109/DASIP.2018.8597105.

[8] Pavan C. Madhusudana dkk. “Capturing Video Frame Rate Variations via Entropic Differencing”. In: *IEEE Signal Processing Letters* 27 (2020), pp. 1809–1813. DOI: 10.1109/LSP.2020.3028687.

[9] Munir Rinaldi. “Pengolahan Citra Digital dengan Pendekatan Algoritmik”. ISBN: 979-3338296. Bandung: Penerbit Informatika, 2004.

[10] Lars S dkk. “The Linux System Adminstrator’s Guide Chapter 6. Memory Management”. https://tldp.org/LDP/sag/html/vm-intro.html. Accessed on 2021-02-22. 2020.

[11] Avi Silbershatz, Peter Baer Galvin, and Greg Gagne. “Operationg System Concepts”. ISBN: 978-0-470-12872-5. John Wiley and Sons, Inc, 2009.

[12] T. Sutoyo dkk. “Teori Pengolahan Citra Digital”. ISBN-13: 978-979-29-0974-6. Jl. Beo 38-40, Yogyakarta 55281: Penerbit Andi, 2009.

[13] Dmitry I. Ustyukov, Alex I. Efimov, and Dmitry A. Kolchaev. “Features of Image Spatial Filters Implementation on FPGA”. In: *Mediterranean Conference On Embedded Computing (Meco)* (June 2019).

[14] Xilinx. “Field Programmable Gate Array (FPGA)”. https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html. Accessed on 2020-04-17. 2020.

[15] Ching-Chung Yang. “Finest Image Sharpening by Sse of the Modified Mask Filter Dealing with Highest Spatial Frequencies”. In: *OPTIK* (Sept. 2013). DOI: 10.1016/j.ijleo.2013.09.070.

[16] Jin Zhao. “Video/Image Processing on FPGA”. Master thesis. Worcester Polytechnic Institute, Apr. 2015.