

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 Xilinx 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 spacial 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 GPUs 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:

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (1)$$

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:

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

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 vertically edge detection and a horizontal sobel kernel for horizontally edge detection [7]. The following are the vertical and horizontal Sobel kernels used:

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (3)$$

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:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad (4)$$

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:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad (5)$$

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 off 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.

2.6.1. 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)

2.7. Evaluasi Kinerja

Pada penelitian ini peneliti menggunakan waktu komputasi, *frame rate* (FPS), penggunaan CPU, penggunaan memory, penggunaan resident memory (RES), shared memory (SHR), dan virtual memory

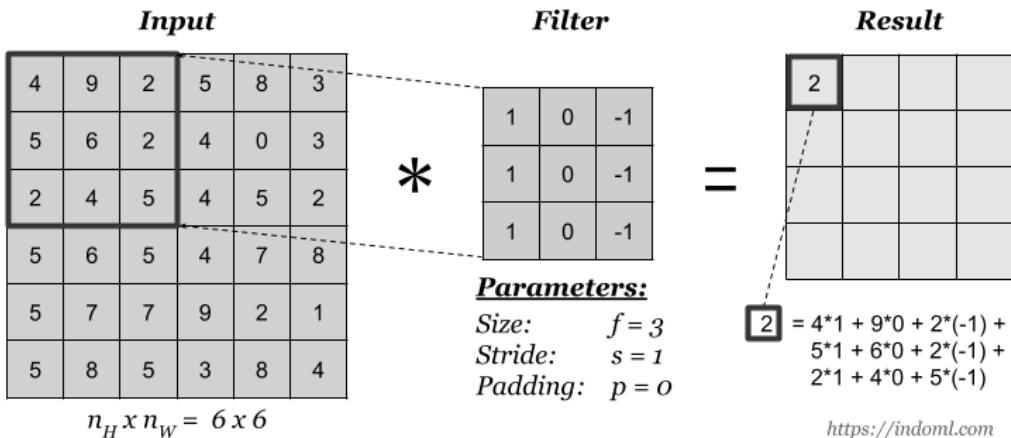


Figure 1: Convolution in digital image.



Figure 2: FPGA Board Xilinx PYNQ-Z2.

(VIRT) untuk mengukur kinerja pada penerapan filter spasial linear dengan prosesor ARM dan FPGA.

2.7.1. Waktu Komputasi

Waktu komputasi yang dimaksud oleh peneliti adalah durasi yang dibutuhkan sebuah kernel untuk melakukan filter spasial linear terhadap beberapa *frame* input. Waktu komputasi ini diperoleh dengan cara menghitung selisih waktu selesai dengan waktu dimulai penerapan filter spasial linear pada *frame* input.

$$\text{waktu komputasi} = \text{waktu selesai} - \text{waktu mulai} \quad (9)$$

2.7.2. Frame Rate (FPS)

Frame rate atau *frame per second* (fps) adalah banyaknya *frame* yang ditampilkan per detik pada video ataupun video *stream*. Semakin tinggi fps sebuah video maka semakin halus pula gerakan yang dihasilkan. Sebaliknya video dengan fps rendah akan menghasilkan gerakan yang kurang baik. *Frame rate* atau fps dapat

dihitung dengan cara membagi jumlah *frame* dengan waktu komputasinya seperti pada persamaan 10 [8].

$$fps = \frac{\text{jumlah frame}}{\text{waktu komputasi}} \quad (10)$$

2.7.3. Penggunaan CPU

Pada penelitian ini peneliti menggunakan fitur yang tersedia pada sistem operasi Linux yang berjalan di FPGA Development Board untuk melihat persentase penggunaan CPU pada proses penerapan filter spasial linear. Program ini menampilkan informasi tentang proses-proses yang berjalan pada sistem operasi seperti ID sebuah proses, user yang menjalankan proses tersebut, *memory* yang digunakan, status sebuah proses, persentase CPU yang digunakan dan lainnya.

2.7.4. Penggunaan Memory

Pada sistem operasi linux *memory* dibagi menjadi tiga jenis [6]. Pertama yaitu *memory* fisik, sumber daya terbatas di mana kode dan data harus berada saat dijalankan atau direferensikan. Berikutnya adalah *memory swap*, yaitu *memory* yang berguna untuk membantu kerja *memory* fisik, data dari *memory* fisik akan disimpan pada *swap* dan kemudian diambil kembali jika terlalu banyak permintaan pada *memory* fisik. Ketiga yaitu virtual *memory*, sumber daya yang hampir tidak terbatas yang digunakan untuk tujuan berikut [11]:

- *abstraction*, bebas dari alamat / batas *memory* fisik
- *isolation*, setiap proses dalam ruang alamat terpisah
- *sharing*, pemetaan tunggal dapat memenuhi banyak kebutuhan
- *flexibility*, menetapkan alamat virtual ke data

2.7.5. Virtual Memory (VIRT)

Virtual *memory* menggunakan disk sebagai perpanjangan dari RAM sehingga ukuran efektif *memory* yang dapat digunakan bertambah secara bersamaan. Kernel akan menulis konten dari blok *memory* yang saat ini tidak digunakan ke hard disk sehingga *memory* dapat digunakan untuk tujuan lain. Ketika konten asli dibutuhkan lagi, mereka dibaca kembali ke dalam *memory*. Ini semua dibuat transparan sepenuhnya bagi pengguna. Program yang berjalan di Linux hanya melihat jumlah *memory* yang tersedia lebih besar dan tidak memperhatikan bahwa sebagian dari program tersebut berada di disk dari waktu ke waktu. Tentu saja, membaca dan menulis hard disk lebih lambat daripada menggunakan *memory* fisik, sehingga program tidak berjalan secepat itu. Bagian dari hard disk yang digunakan sebagai *memory* virtual disebut ruang swap [10].

2.7.6. Resident Memory (RES)

Resident memory adalah bagian dari ruang alamat virtual (VIRT) yang mewakili *memory* fisik yang tidak ditukar yang sedang digunakan tugas. Resident *memory* ini juga merupakan penjumlahan dari RSan, RSfd dan Bidang RSsh. Ini dapat mencakup private anonymous *pages*, halaman pribadi yang dipetakan ke file (termasuk *program images* dan *shared libraries*) ditambah shared anonymous *pages*. Semua *memory* tersebut didukung oleh file *swap* yang direpresentasikan secara terpisah pada SWAP. Resident *memory* ini juga dapat menyertakan *pages* yang didukung *shared file-backed* yang apabila dimodifikasi, maka akan bertindak sebagai file *swap* khusus dan karenanya tidak akan pernah memengaruhi SWAP [6].

2.7.7. Shared Memory (SHR)

Shared memory adalah bagian dari resident *memory* (RES) yang dapat digunakan oleh proses lain. Termasuk *anonymous pages* dan *shared file-backed pages*. Ini juga termasuk private *pages* dipetakan ke file yang mewakili *program images* dan *shared libraries* [6].

3. TOOLS AND METHODS

3.1. Rancangan Sistem

Video *stream* dari *source* disalurkan melalui port HDMI Input pada FPGA Development Board, kemudian video *stream* tersebut akan diolah dengan menerapkan filter spasial linear pada setiap framenya. Setiap

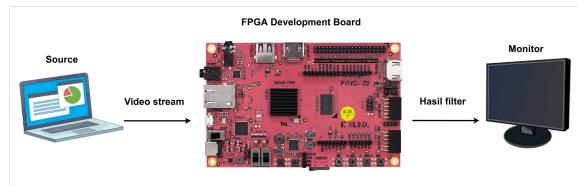


Figure 3: Rancangan sistem.

frame yang telah diterapkan filter spasial akan dialirkan ke monitor untuk kemudian ditampilkan. Selanjutnya dilakukan analisis kinerja pada FPGA. FPGA Development Board yang digunakan dalam penelitian ini dapat diakses dengan *ssh* pada port 22 atau dengan *Jupyter Notebook* melalui *web browser*.

3.2. Instrumen Penelitian

1. Kebutuhan perangkat lunak:
 - a. Linux Ubuntu 18, sebagai OS pada FPGA Development Board.
 - b. Python 3.6, dengan library OpenCV, Numpy, Pynq 5.2, dan Xilinx xfOpenCV.
 - c. Jupyter Notebook pada FPGA Development Board.
 - d. Web Browser untuk mengakses Jupyter Notebook pada FPGA Development Board.
2. Kebutuhan perangkat keras:
 - a. FPGA Development Board.
 - b. Micro SD Card 16 GB, sebagai media penyimpanan OS pada FPGA Development Board.
 - c. Monitor Eksternal, untuk menampilkan hasil penerapan filter spasial pada FPGA Development Board.
 - d. Laptop Lenovo Ideapad 320 (sebagai *source* video *stream*).

Spesifikasi FPGA Development Board yang digunakan:

- 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. Penerapan Filter Spasial

Proses filter spasial dilakukan dengan operasi konvolusi pada setiap matriks dengan kernel yang telah ditentukan sebelumnya. Operasi konvolusi ini



Figure 5: Hasil filter Average Blur.

menghasilkan matrix baru dengan ukuran 1280x720. Matriks hasil tersebut selanjutnya direpresentasikan kembali sebagai citra digital yang selanjutnya disebut sebagai hasil filter. Hasil filter dari setiap *frame* ini ditampilkan ke monitor melalui HDMI Output pada FPGA Development Board secara berkesinambungan sehingga tampak seperti video.



Figure 4: Contoh Frame Grayscale.

3.3.1. Average Blur

Penerapan filter spasial pada *frame grayscale* yang berukuran 1280x720 pixel dengan kernel *average blur* (1) yang berukuran 3x3 menghasilkan citra *blur* yang berukuran 1280x720. Hasil filter *average blur* dapat dilihat pada gambar 5. Filter seperti ini dapat digunakan untuk mengurangi derau pada citra.

3.3.2. Gaussian Blur

Penerapan filter spasial dengan kernel *gaussian blur* (2) yang berukuran 3x3 menghasilkan citra *blur* yang secara kasat mata mirip dengan filter *average blur*. Namun apabila diperhatikan nilai masing-masing pixel pada gambar 6 akan terlihat berbeda dengan nilai masing-masing pixel pada gambar 5. Hal ini disebabkan oleh nilai bobot pada kernel *gaussian blur* yang berbeda dengan kernel *average blur* sehingga hasil konvolusinya juga berbeda.



Figure 6: Hasil filter Gaussian Blur.

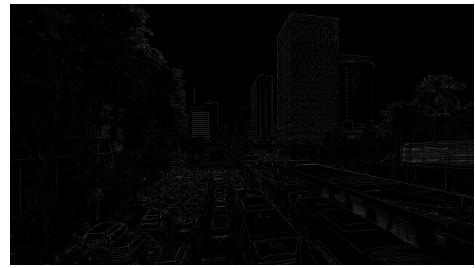


Figure 7: Hasil filter Laplacian.

3.3.3. Laplacian

Penerapan filter spasial dengan kernel *laplacian* (4) menghasilkan citra *biner* yang hanya direpresentasikan dengan warna hitam dan putih saja, dapat dilihat pada gambar 7. Filter seperti ini dapat digunakan pada metode deteksi tepi dalam proses pengolahan citra digital.

3.3.4. Sharpening

Penerapan filter spasial dengan kernel *sharpening* (5) dapat meningkatkan detail (seperti garis) pada citra, namun dapat juga dapat menimbulkan derau pada citra apabila bobot kernelnya tidak sesuai. Filter seperti ini lebih tepat digunakan untuk memperbaiki kualitas citra (dengan nilai kernel yang sesuai). Hasil filter *sharpening* ini dapat dilihat pada gambar 8.



Figure 8: Hasil filter Sharpening.

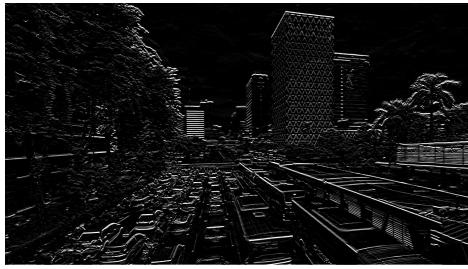


Figure 9: Hasil filter Sobel Horizontal.

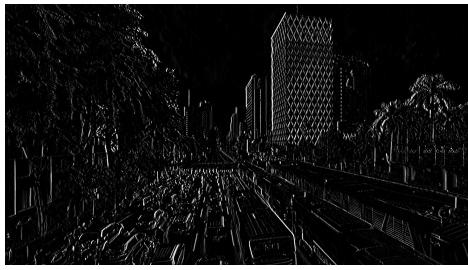


Figure 10: Hasil filter Sobel Vertical.

3.3.5. Sobel Horizontal

Penerapan filter spasial dengan kernel *sobel horizontal* (3) menghasilkan citra *biner*, dapat dilihat pada gambar 9. Filter seperti lebih tepat digunakan pada metode deteksi tepi dengan citra yang banyak mengandung garis horizontal.

3.3.6. Sobel Vertical

Penerapan filter spasial dengan kernel *sobel vertical* (3) menghasilkan citra *biner*, dapat dilihat pada gambar 10. Sama halnya dengan filter *sobel horizontal*, filter *sobel vertical* juga dapat digunakan untuk metode deteksi tepi, terutama pada citra yang banyak mengandung garis vertikal.

4. RESULTS AND DISCUSSIONS

4.1. Waktu Komputasi

Data waktu komputasi dengan menggunakan 50 frame pada masing-masing kernel dapat dilihat pada tabel 1 dan grafik pada gambar 11. Secara umum waktu komputasi dengan menggunakan prosesor ARM lebih lambat daripada waktu komputasi dengan menggunakan FPGA. Rata-rata waktu komputasi dengan prosesor ARM menggunakan 50 frame adalah 7,26 detik, sedangkan rata-rata waktu komputasi dengan FPGA menggunakan 50 frame hanya 0,82 detik.

Table 1: Tabel perbandingan waktu komputasi dengan menggunakan 50 frame.

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

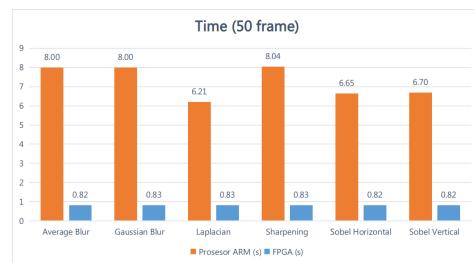


Figure 11: Grafik perbandingan waktu komputasi dengan 50 frame dan 200 frame

Data waktu komputasi menggunakan 200 frame pada masing-masing kernel dapat dilihat pada tabel 2 dan grafik pada gambar 12. Rata-rata waktu komputasi dengan prosesor ARM menggunakan 200 frame adalah 29,06 detik, sedangkan rata-rata waktu komputasi dengan FPGA menggunakan 200 frame hanya 3,32 detik.

Table 2: Tabel perbandingan waktu komputasi dengan menggunakan 200 frame.

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

Waktu komputasi tercepat dengan menggunakan prosesor ARM terdapat pada filter *laplacian* yaitu 6,21 detik dengan 50 frame dan 24,85 detik dengan 200 frame. Sedangkan waktu komputasi paling lambat ketika menggunakan prosesor ARM terdapat pada filter *sharpening* yaitu 8,04 detik dengan 50 frame dan 32,24

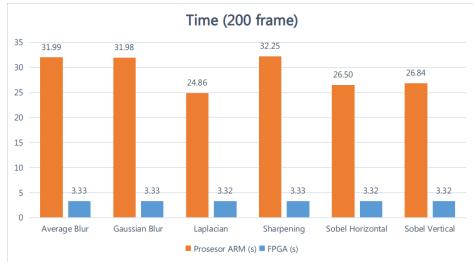


Figure 12: Grafik perbandingan waktu komputasi dengan 50 frame dan 200 frame

detik dengan 200 frame.

Untuk menghitung efisiensi waktu komputasi yang dimiliki FPGA dibandingkan dengan prosesor ARM, digunakan rumus:

$$\begin{aligned}
 &= 100\% - \left(\frac{\text{waktu komputasi FPGA}}{\text{waktu komputasi ARM Prosesor}} \times 100\% \right) \\
 &= 100\% - \left(\frac{3,32}{29,06} \times 100\% \right) \\
 &= 100\% - 11.42\% \\
 &= 88.58\%
 \end{aligned}$$

sehingga diperoleh efisiensi waktu komputasi FPGA dibandingkan dengan prosesor ARM adalah sebesar 88.85%.

4.2. Frame Rate (FPS)

Dengan mengetahui waktu komputasi dan jumlah frame maka frame rate atau FPS dapat dihitung menggunakan persamaan 10. Data FPS dari masing-masing kernel dengan prosesor ARM dan FPGA dapat dilihat pada tabel 3 dan grafik pada gambar 13.

Table 3: Tabel perbandingan FPS dengan menggunakan prosesor ARM dan 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

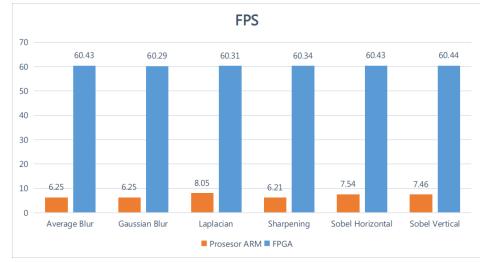


Figure 13: Grafik perbandingan FPS dengan menggunakan prosesor ARM dan FPGA.

Pada tabel 3 terlihat dengan menggunakan prosesor ARM diperoleh rata-rata 6.95 frame per detik (FPS), sedangkan ketika menggunakan FPGA diperoleh rata-rata 60.37 frame per detik. Terlihat pada grafik 13 nilai FPS dengan FPGA jauh lebih tinggi daripada dengan prosesor ARM.

Untuk menghitung efisiensi FPS yang dimiliki FPGA dibandingkan dengan prosesor ARM, digunakan rumus:

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

sehingga diperoleh efisiensi FPS dari FPGA dibandingkan dengan prosesor ARM adalah sebesar 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.

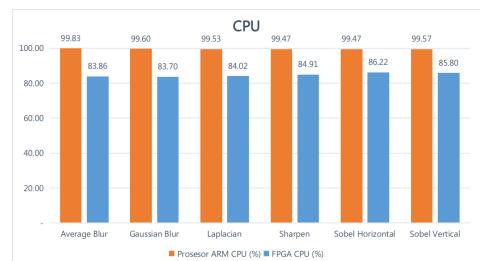


Figure 14: Grafik perbandingan penggunaan CPU dengan menggunakan prosesor ARM dan FPGA.

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

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:

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

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

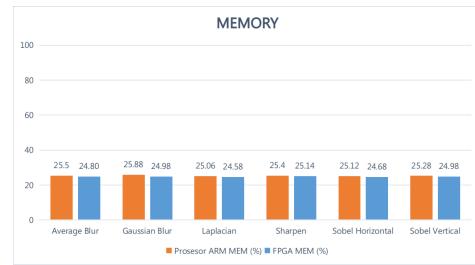


Figure 15: Grafik perbandingan penggunaan memory dengan menggunakan prosesor ARM dan FPGA.

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:

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

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:

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

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

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

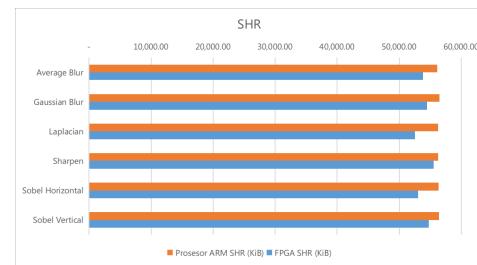


Figure 17: Grafik perbandingan penggunaan shared memory (SHR) dengan menggunakan prosesor ARM dan FPGA.

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.

Efisiensi penggunaan *shared memory* yang dimiliki FPGA dibandingkan dengan prosesor ARM, dapat

dihitung dengan persamaan berikut:

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

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:

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

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

5. CONCLUSION

Berdasarkan hasil penerapan filter spasial linear pada FPGA Development Board dengan menggunakan 6 kernel, peneliti dapat menarik beberapa kesimpulan sebagai berikut:

1. Proses implementasi filter spasial linear pada video *stream* dengan FPGA Development Board dilakukan dengan *library OpenCV python* dan *library xfOpenCV Xilinx*. Setiap *frame* dari *source video stream* direpresentasikan sebagai citra digital kemudian dilakukan filter spasial linear, selanjutnya hasil filter ini ditampilkan secara berkesinambungan sehingga tampak seperti video.
2. Waktu komputasi dengan FPGA 88.85% lebih baik dibandingkan dengan ARM prosesor. Video hasil filter dengan ARM prosesor memperoleh rata-rata 6.95 fps sedangkan dengan FPGA rata-rata 60.37 fps. FPS dengan FPGA 88.49% lebih baik lebih baik dibandingkan dengan ARM prosesor. Penggunaan CPU pada FPGA 14.89% lebih baik, penggunaan *memory* pada FPGA 2.02% lebih baik, penggunaan *resident memory* 2.07% lebih baik, dan penggunaan *shared memory* 4.08% lebih baik dibandingkan dengan ARM prosesor. Sedangkan penggunaan *virtual memory* pada ARM prosesor 0.03% lebih baik dibandingkan 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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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 Administrator’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](https://doi.org/10.1016/j.ijleo.2013.09.070).
- [16] Jin Zhao. “Video/Image Processing on FPGA”. Master thesis. Worcester Polytechnic Institute, Apr. 2015.