

# IMPLEMENTASI FILTER SPASIAL LINEAR PADA VIDEO STREAM MENGGUNAKAN FPGA HARDWARE ACCELERATOR

SULAEMAN

Universitas Hasanuddin

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.*

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

## 1. PENDAHULUAN

Untuk meningkatkan kinerja dan efisiensi energi dari sebuah program, berbagai jenis akseleator telah dikembangkan, salah satu diantaranya yaitu FPGA [lb:cong]. *Field Programmable Gate Arrays* atau FPGA adalah perangkat semikonduktor yang berbasis *matriks configurable logic block* (CLBs) yang terhubung melalui interkoneksi yang dapat diprogram. FPGA dapat diprogram ulang dengan aplikasi atau fungsi yang diinginkan setelah *manufacturing*. Fitur ini yang membedakan FPGA dengan *Application Specific Integrated Circuits* (ASICs), yang dibuat khusus untuk tugas tertentu saja [XILINX].

FPGA telah menunjukkan kinerja yang sangat tinggi di dalam banyak aplikasi dalam pemrosesan citra. Namun CPU dan GPU terbaru memiliki potensi kinerja tinggi untuk masalah-masalah tersebut. CPU terbaru mendukung *multi-core*, dimana masing-masing *core* mendukung SIMD (*Single Instruction, Multiple Data*) yang telah dikembangkan dan dijalankan hingga 16 operasi pada 128 bit data dalam satu *clock cycle*. GPU terbaru mendukung sejumlah besar *core* yang berjalan secara paralel, dan kinerja puncaknya mampu mengungguli CPU [lb:asano].

Paralelisme dalam SIMD pada CPU terbatas, tetapi

frekuensi operasional CPU sangatlah tinggi, dan CPU diharapkan dapat menunjukkan kinerja yang tinggi dalam aplikasi yang dimana *cache memory* berjalan dengan baik. Ukuran *cache memory* cukup besar untuk menyimpan seluruh citra di banyak aplikasi pemrosesan citra, dan CPU dapat menjalankan algoritma yang sama dengan FPGA meskipun *bandwidth memory* yang dibutuhkan tinggi [lb:asano].

Frekuensi operasional GPU lebih cepat dibandingkan dengan FPGA, namun sedikit lebih lambat dibandingkan dengan CPU. Akan tetapi, GPU mendukung banyak *core* yang berjalan secara paralel sehingga kinerja puncaknya mengungguli CPU. Namun *core*-nya dikelompokkan, dan transfer data antara kelompok sangatlah lambat. Selain itu, ukuran *local memory* yang disediakan masing-masing kelompok sangat kecil. Karena keterbatasan ini, GPU tidak dapat menjalankan algoritma yang sama seperti FPGA dalam beberapa masalah aplikasi [lb:asano].

Sebagian besar FPGA sekarang telah dirangkai dengan prosesor dalam satu *board*, sering disebut sebagai *FPGA Development Board*. Xilinx PYNQ-Z2 dibangun dari prosesor ARM Cortex-A9, sehingga dapat menjalankan beberapa *software* seperti *python* tanpa harus merancang sirkuitnya dari awal. Akan tetapi, kinerja yang dimiliki oleh prosesor ARM pada FPGA

Development Board tentu berbeda dengan kinerja fungsi arsitektur FPGA itu sendiri sehingga dapat dikaji lebih dalam mengenai perbandingan kinerja dari keduanya.

## 2. LANDASAN TEORI

### 2.1. Pengolahan Citra Digital

Pengolahan citra digital merupakan proses mengolah piksel-piksel di dalam citra secara digital untuk tujuan tertentu. Berdasarkan tingkat pemrosesannya pengolahan citra digital dikelompokkan menjadi tiga kategori, yaitu: *low-level*, *mid-level* dan pemrosesan *high-level*. Pemrosesan *low-level* dilakukan dengan operasi primitif seperti *image preprocessing* untuk mengurangi derau (*noise*), memperbaiki kontras citra dan mempertajam citra (*sharpening*). Karakteristik dari pemrosesan *low-level* yaitu keluaran atau hasil dari pemrosesannya berupa citra digital. Pemrosesan *mid-level* melibatkan tugas-tugas seperti segmentasi (mempartisi gambar menjadi beberapa bagian atau objek), deskripsi objek untuk dilakukan pemrosesan lanjutan, dan klasifikasi objek dalam citra digital. Karakteristik dari pemrosesan *mid-level* yaitu keluaran atau hasilnya berupa atribut atau fitur seperti, kontur, tepi, atau objek yang terdapat dalam citra tersebut. Pemrosesan *high-level* merupakan proses tingkat lanjut dari dua proses sebelumnya, dilakukan untuk mendapat informasi lebih yang terkandung dalam citra [book:gonzalez].

### 2.2. Filter Spasial

Konsep filter spasial pada pengolahan citra digital berasal dari penerapan transformasi Fourier untuk pemrosesan sinyal pada domain frekuensi. Istilah filter spasial ini digunakan untuk membedakan proses ini dengan filter pada domain frekuensi. Proses filter dilakukan dengan cara menggeser filter kernel dari titik ke titik dalam citra digital. Istilah *mask*, *kernel*, *template*, dan *window* merupakan istilah yang sama dan sering digunakan dalam pengolahan citra digital [book:gonzalez]. Dalam penelitian ini peneliti menggunakan istilah kernel untuk istilah tersebut.

### 2.3. Kernel

#### 2.3.1. Average Blur

*Average blur* atau biasa juga disebut *box filter* adalah salah satu filter yang digunakan untuk menghaluskan citra dan mengurangi derau. Secara sederhana nilai sebuah piksel yang baru adalah nilai rata-rata dari

nilai piksel tersebut dengan nilai piksel tetangganya [pdf:marcin]. Berikut kernel *Average blur* yang digunakan dalam penelitian ini:

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

#### 2.3.2. Gaussian Blur

Filter ini juga digunakan untuk menghaluskan citra dan mengurangi derau. Idenya mirip seperti *Average blur*, nilai piksel yang baru dibentuk dari nilai piksel tetangganya, tetapi dengan memberikan bobot yang lebih kuat pada nilai pikselnya sendiri diikuti dengan bobot yang lebih rendah pada piksel atas, bawah dan sampingnya [soa:dmitry]. Berikut kernel gaussian blur filter yang digunakan dalam penelitian ini:

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

#### 2.3.3. Sobel

Filter Sobel termasuk *high-pass* filter yang umum digunakan untuk deteksi tepi pada citra. Sobel memiliki dua kernel untuk deteksi tepi yaitu kernel sobel vertikal untuk mendeteksi tepi secara vertikal dan kernel sobel horizontal yang mendeteksi tepi secara horizontal [pdf:marcin]. Berikut kernel Sobel vertikal dan horizontal yang digunakan dalam penelitian ini:

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

#### 2.3.4. Laplacian

Filter ini dapat digunakan untuk deteksi tepi pada citra karena sifatnya yang sensitif dengan perubahan intensitas yang cepat [pdf:jingbo]. Tidak seperti Sobel yang menggunakan dua kernel untuk mendeteksi tepi secara vertikal dan horizontal, disini hanya digunakan sebuah kernel yang dapat digunakan untuk deteksi tepi secara vertikal dan horizontal sekaligus. Berikut kernel Laplacian yang digunakan dalam penelitian ini:

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

### 2.3.5. Sharpening

Sharpening filter digunakan untuk memperjelas detail halus dalam citra atau untuk meningkatkan detail pada citra yang *blur*, baik karena kesalahan ataupun karena efek dari metode akuisisi citra tertentu [pdf:ching]. Berikut kernel untuk filter *sharpening* yang digunakan dalam penelitian ini:

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

### 2.4. Konvolusi

Konsep filter spasial linear mirip seperti konsep konvolusi pada domain frekuensi, dengan alasan tersebut filter spasial linear biasa disebut juga konvolusi sebuah kernel dengan citra digital [book:gonzalez]. Konvolusi pada fungsi  $f(x)$  dan  $g(x)$  didefinisikan pada persamaan ??.

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

dimana tanda \* menyatakan operator konvolusi, dan peubah a adalah peubah bantu. Untuk fungsi diskrit, konvolusi didefinisikan pada persamaan ??.

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

Pada operasi konvolusi diatas,  $g(x)$  disebut kernel konvolusi atau filter kernel. Kernel  $g(x)$  dioperasikan secara bergeser pada sinyal masukan  $f(x)$ . Jumlah perkalian kedua fungsi pada setiap titik merupakan hasil konvolusi yang dinyatakan dengan keluaran  $h(x)$  [book:munir].

Konvolusi dengan persamaan ?? hanya berlaku untuk input 1 dimensi, sedangkan untuk input 2 dimensi seperti citra dapat digunakan persamaan ??.

$$h(x, y) = (I * K)(x, y) = \sum_m \sum_n I(m, n)K(x - m, y - n) \quad (8)$$

Keterangan:

- $h(x,y)$  = fungsi hasil konvolusi
- $I$  = input
- $K$  = kernel konvolusi
- $x,y$  = piksel input
- $m,n$  = piksel kernel

Pada gambar (??) diilustrasikan bagaimana proses konvolusi pada citra digital yang direpresentasikan dalam bentuk matriks. Operasi konvolusi dilakukan pada matriks input berukuran  $6 \times 6$  dengan filter

berukuran  $3 \times 3$ . Hasil konvolusinya ditampilkan pada matriks *result*.

Jika hasil konvolusi menghasilkan nilai piksel negatif, maka nilai tersebut dijadikan 0, sebaliknya jika hasil konvolusi menghasilkan nilai piksel yang melebihi nilai keabuan maksimum, maka nilai tersebut dijadikan ke nilai keabuan maksimum pada citra tersebut [book:sutoyo].

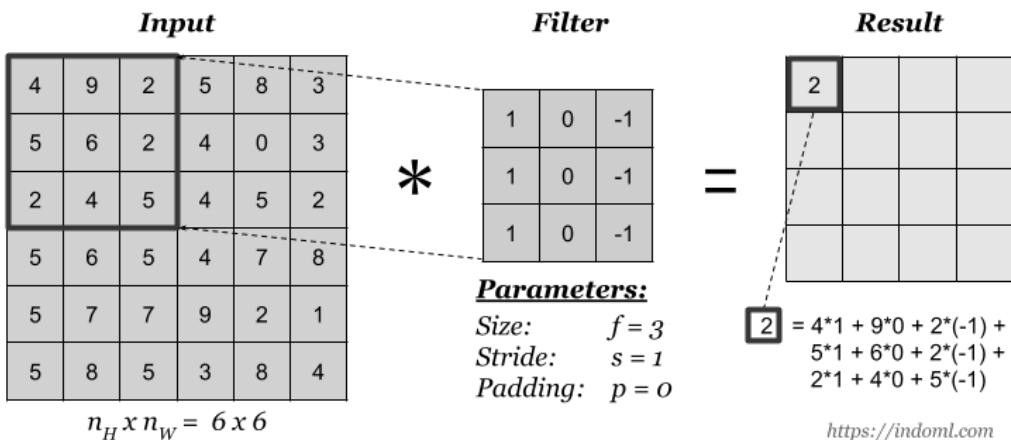
### 2.5. Video Stream

Video *stream* dapat dipandang sebagai serangkaian citra digital berturut-turut [thesis:jin]. Berbeda dengan format video lainnya, video *stream* ini tidak disimpan pada media penyimpanan sebagai file dengan format video melainkan langsung disalurkan setiap framenya dari sumber (*source*) ke penerima, dalam hal ini FPGA. Dengan menganggap Video *stream* adalah kumpulan citra digital (*frame*) maka dapat dilakukan metode pengolahan seperti pada citra digital, termasuk penerapan filter spasial.

### 2.6. FPGA

*Field Programmable Gate Arrays* atau FPGA adalah perangkat semikonduktor yang berbasis *matriks configurable logic block* (CLBs) yang terhubung melalui interkoneksi yang dapat diprogram. FPGA dapat diprogram ulang ke aplikasi atau fungsi yang diinginkan setelah *manufacturing*. Fitur ini yang membedakan FPGA dengan *Application Specific Integrated Circuits* (ASICs), yang dibuat khusus untuk tugas tertentu saja [XILINX].

Sebuah *microprocessor* menerima instruksi berupa kode 1 atau 0, kode-kode ini selanjutnya diinterpretasikan oleh komputer untuk menjalankan perintah yang diberikan. *Microprocessor* ini membutuhkan intruksi berupa kode secara terus menerus untuk menjalankan fungsinya. Sedangkan pada FPGA hanya dibutuhkan sekali konfigurasi *chip* setiap kali dinyalakan. Membuat atau mengunduh *bitstream* yang menentukan fungsi logika dilakukan oleh *logic elements* (LEs), sebuah sirkuit dapat dibuat dengan mengabungkan beberapa LEs menjadi satu kesatuan. Setelah *bitstream* dipasang, FPGA tidak perlu lagi membaca instruksi berupa 1 dan 0, berbeda dengan *microprocessor* yang selalu membutuhkan instruksi [pdf:cheung]. Secara tradisional, untuk membuat sebuah desain FPGA, aplikasi dideskripsikan menggunakan *Hardware Description Language* (HDL) seperti Verilog atau VHDL sehingga menghasilkan sebuah *bitstream* FPGA.



Gambar 1: Ilustrasi konvolusi pada citra.

### 2.6.1. FPGA Development Board

FPGA *Development Board* atau biasa disebut juga FPGA *Board* yaitu teknologi FPGA yang dirangkai dalam sebuah *board* dan dilengkapi dengan *microprocessor* dan beberapa *interface IO* untuk menjalankan tugas tertentu. Umumnya FPGA *Board* telah dilengkapi dengan interface untuk mengakses dan menerapkan desain sirkuitnya. Xilinx, Altera dan Intel adalah produsen FPGA *Board* yang terkenal. FPGA *Board* yang digunakan dalam penelitian ini yaitu Xilinx PYNQ-Z2 dengan Jupyter Notebook sebagai *interface* untuk mengakses dan menjalankan program pada penelitian ini. Bentuk FPGA *Board* Xilinx PYNQ-Z2 dapat dilihat pada gambar ??



Gambar 2: FPGA Board Xilinx PYNQ-Z2.

### 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

(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 ?? [pdf:pavan].

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

#### 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 [[manual:linux](#)]. 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 [[book:os](#)]:

- *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 [[site:ltdp](#)].

#### 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

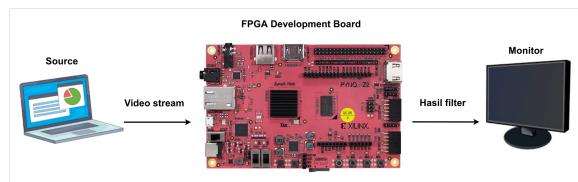
file (termasuk *program images* dan *shared libraries*) ditambah shared anonymous *pages*. Semua *memory* tersebut didukung oleh file *swap* yang direpresentasikan secara terpisah pada SWAP. Resident *memory* ini juga dapat menyertakan *pages* yang didukung *shared file-backed* yang apabila dimodifikasi, maka akan bertindak sebagai file *swap* khusus dan karenanya tidak akan pernah memengaruhi SWAP [[manual:linux](#)].

#### 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* [[manual:linux](#)].

### 3. METODE PENELITIAN

#### 3.1. Rancangan Sistem



Gambar 3: 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 *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 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.



Gambar 4: Contoh Frame Grayscale.

#### 3.3.1. Average Blur

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



Gambar 5: Hasil filter Average Blur.



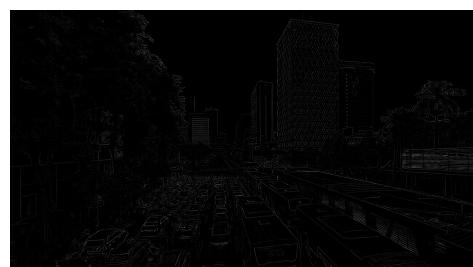
Gambar 6: Hasil filter Gaussian Blur.

#### 3.3.2. Gaussian Blur

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

#### 3.3.3. Laplacian

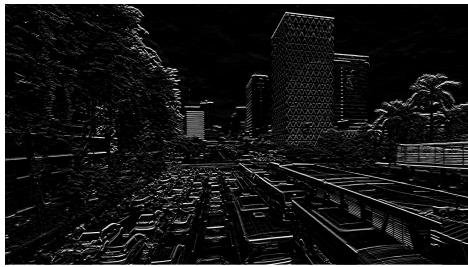
Penerapan filter spasial dengan kernel *laplacian* (??) menghasilkan citra *biner* yang hanya direpresentasikan dengan warna hitam dan putih saja, dapat dilihat pada gambar ???. Filter seperti ini dapat digunakan pada metode deteksi tepi dalam proses pengolahan citra digital.



Gambar 7: Hasil filter Laplacian.



Gambar 8: Hasil filter Sharpening.



Gambar 9: Hasil filter Sobel Horizontal.

### 3.3.4. Sharpening

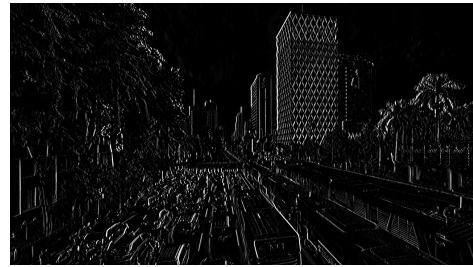
Penerapan filter spasial dengan kernel *sharpening* (??) dapat meningkatkan detail (seperti garis) pada citra, namun dapat juga dapat menimbulkan derau pada citra apabila bobot kernelnya tidak sesuai. Filter seperti ini lebih tepat digunakan untuk memperbaiki kualitas citra (dengan nilai kernel yang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

### 3.3.5. Sobel Horizontal

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



Gambar 10: Hasil filter Sobel Vertical.

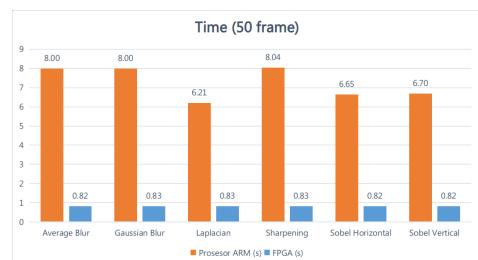
## 4. HASIL DAN PEMBAHASAN

### 4.1. Waktu Komputasi

Data waktu komputasi dengan menggunakan 50 frame pada masing-masing kernel dapat dilihat pada tabel ?? dan grafik pada gambar ???. 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.

Tabel 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



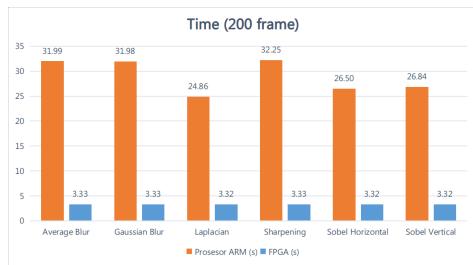
Gambar 11: Grafik perbandingan waktu komputasi dengan 50 frame dan 200 frame

Data waktu komputasi menggunakan 200 frame pada masing-masing kernel dapat dilihat pada tabel ?? dan

grafik pada gambar ???. 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.

**Tabel 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	3.324144484
Sobel Vertical	26.84196448	3.324060822
Rata-rata	29.06964172	3.325270478



**Gambar 12:** Grafik perbandingan waktu komputasi dengan 50 frame dan 200 frame

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 detik dengan 200 frame.

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

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

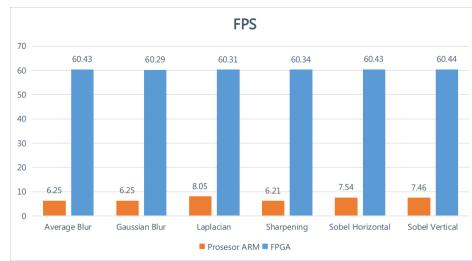
sehingga diperoleh efisiensi waktu komputasi FPGA dibandingkan dengan prosesor ARM adalah sebesar 88.85%.

## 4.2. Frame Rate (FPS)

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

**Tabel 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



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

Pada tabel ?? 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 ?? nilai FPS dengan FPGA jauh lebih tinggi daripada dengan prosesor ARM.

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

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

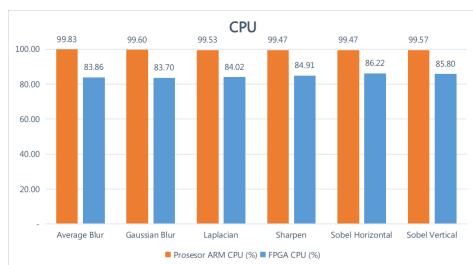
sehingga diperoleh efisiensi FPS dari FPGA dibandingkan dengan prosesor ARM adalah sebesar 88.49%.

### 4.3. Penggunaan CPU

Data perbandingan penggunaan CPU pada masing-masing kernel dengan prosesor ARM dan FPGA dapat dilihat pada tabel ?? dan grafik pada gambar ???. 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.

**Tabel 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



**Gambar 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:

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

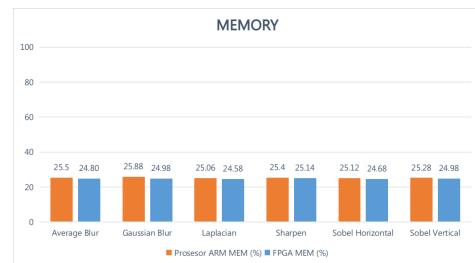
sehingga diperoleh efisiensi penggunaan CPU FPGA dibandingkan dengan prosesor ARM adalah sebesar 14,89%.

### 4.4. Penggunaan Memory

Data penggunaan *memory* dengan prosesor ARM dan FPGA dapat dilihat pada tabel ?? dan grafik pada gambar ???. 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.

**Tabel 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



**Gambar 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:

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

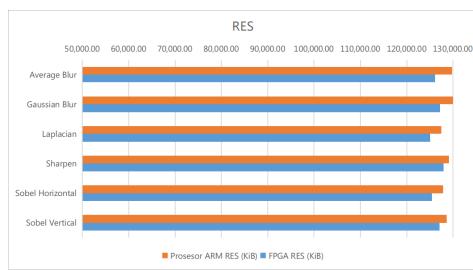
diperoleh efisiensi penggunaan *memory* FPGA dibandingkan dengan prosesor ARM adalah sebesar 2.02%.

#### 4.5. Resident Memory (RES)

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

**Tabel 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



**Gambar 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 ?? dan grafik pada gambar ???. 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

**Tabel 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

*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



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

dihitung dengan persamaan berikut:

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

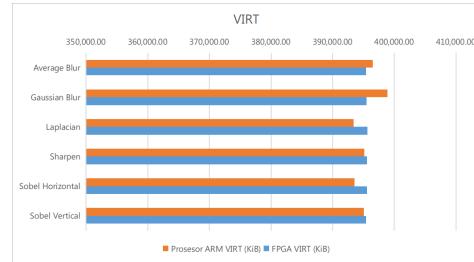
diperoleh efisiensi penggunaan *shared memory* FPGA dibandingkan dengan prosesor ARM adalah sebesar 4.08%.

#### 4.7. Virtual Memory (VIRT)

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

**Tabel 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



**Gambar 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:

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

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

## 5. KESIMPULAN

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.