



**FUNDAMENTAL OF DIGITAL SYSTEM FINAL PROJECT REPORT
DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITAS INDONESIA**

Image Convolution using VHDL

GROUP PA12

Bryan Herdianto	2306210885
Rowen Rodotua Harahap	2306250604
Filaga Tifira Muthi	2306208445
Calvin Kim	2306267095

PREFACE

Segala puji dan syukur kami panjatkan kepada Tuhan Yang Maha Esa atas rahmat dan karunia-Nya sehingga kami dapat menyelesaikan proyek akhir mata kuliah Perancangan Sistem Digital dengan tema *Image Convolution*. Berkat bimbingan-Nya, kami diberi kesehatan, kekuatan, dan hikmat untuk menuntaskan tugas ini sesuai dengan target yang telah direncanakan. Tanpa izin dan kasih karunia-Nya, proses yang penuh tantangan ini tidak mungkin dapat kami lalui dengan baik.

Kami juga ingin menyampaikan rasa terima kasih yang sebesar-besarnya kepada asisten laboratorium pendamping yang telah dengan sabar dan penuh dedikasi memberikan bimbingan, masukan, dan dukungan selama proses penggerjaan proyek ini. Kehadiran Anda tidak hanya membantu kami memahami aspek teknis dari sistem digital, tetapi juga memberikan semangat dan inspirasi dalam mengatasi berbagai kesulitan yang kami hadapi. Kesediaan Anda untuk meluangkan waktu di tengah kesibukan sangat kami apresiasi.

Akhir kata, kami berharap hasil dari proyek ini dapat bermanfaat tidak hanya bagi kami sebagai pembelajar, tetapi juga bagi pengembangan ilmu di bidang sistem digital. Kami menyadari bahwa proyek ini masih memiliki kekurangan, sehingga kami terbuka terhadap kritik dan saran yang membangun untuk meningkatkan kualitas karya kami di masa mendatang. Terima kasih atas segala dukungan dan doa dari berbagai pihak yang turut membantu keberhasilan proyek ini.

Depok, December 8, 2024

Group PA12

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

- 1.1 Background
- 1.2 Project Description
- 1.3 Objectives
- 1.4 Roles and Responsibilities

CHAPTER 2: IMPLEMENTATION

- 2.1 Equipment
- 2.2 Implementation

CHAPTER 3: TESTING AND ANALYSIS

- 3.1 Testing
- 3.2 Result

CHAPTER 4: CONCLUSION

REFERENCES

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Proyek akhir ini mendemonstrasikan implementasi konvolusi gambar menggunakan VHDL. Desain ini mengintegrasikan *Matrix Multiplier Unit* (MMU) dan *Arithmetic Logic Unit* (ALU) untuk mempercepat pemrosesan dengan berbagai kernel seperti *box blur*, *sharpening*, dan *edge detection*. Sistem ini mendukung pemilihan kernel dinamis berbasis file dan penanganan opcode khusus. Hasil pengujian validasi kemampuan sistem untuk menangani pemrosesan gambar dengan presisi dan efisiensi.

Proyek ini bertujuan untuk merancang dan mengimplementasikan sistem Konvolusi Gambar dalam VHDL, yang menyediakan arsitektur fleksibel dan modular untuk melakukan operasi konvolusi pada data gambar. Konvolusi gambar adalah operasi dasar dalam pemrosesan gambar dan visi komputer, yang digunakan secara luas dalam tugas-tugas seperti ekstraksi fitur dan penyaringan. Dengan memanfaatkan algoritma yang berbeda untuk unit pemrosesan tertentu, implementasi ini mengeksplorasi pendekatan baru untuk meningkatkan efisiensi dan modularitas konvolusi.

1.2 PROJECT DESCRIPTION

A. Image Processing

Image processing adalah teknik manipulasi dan analisis gambar digital dengan bantuan komputer. Operasi dasar melibatkan pengolahan data piksel gambar untuk meningkatkan kualitas, mengurangi noise, mendeteksi tepi, dan melakukan transformasi lainnya. Berikut ini adalah beberapa langkah pemrosesan dasar:

- a. Akuisisi gambar
- b. Representasi dan Deskripsi Gambar
- c. Analisis gambar
- d. Sintesis dan kompresi gambar
- e. Peningkatan Citra

- f. Segmentasi
- g. Ekstraksi Fitur
- h. Restorasi Citra
- i. Pengenalan Pola

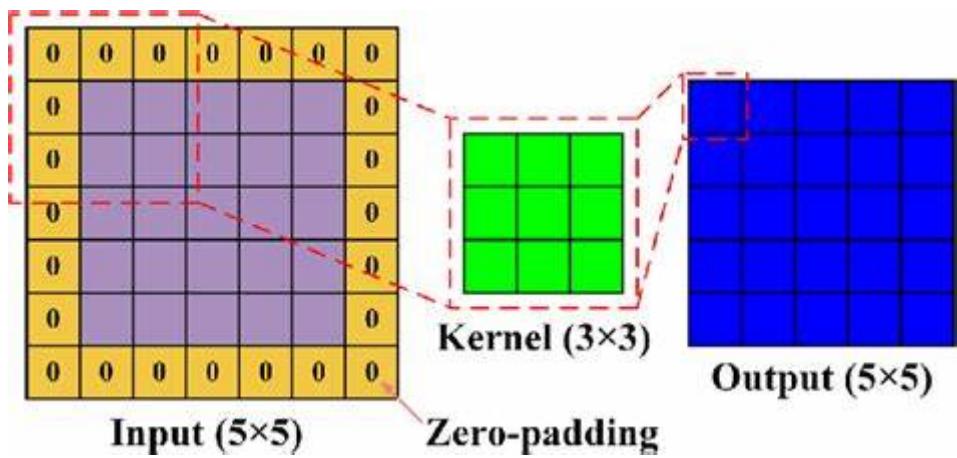
B. Convolution

Konvolusi adalah operasi matematika yang digunakan untuk melakukan filter atau memodifikasi citra dengan menerapkan kernel tertentu. Dalam konvolusi kernel adalah matriks kecil yang diaplikasikan pada setiap piksel gambar. Proses ini melibatkan perkalian elemen-wise antara kernel dan piksel gambar yang dilalui, diikuti dengan penjumlahan hasilnya. Rumus dasar konvolusi untuk setiap piksel output:

$$F(x, y) = \sum_i \sum_j f(x+i, y+j) h(i, j)$$

Berikut adalah penjelasan dari rumus konvolusi:

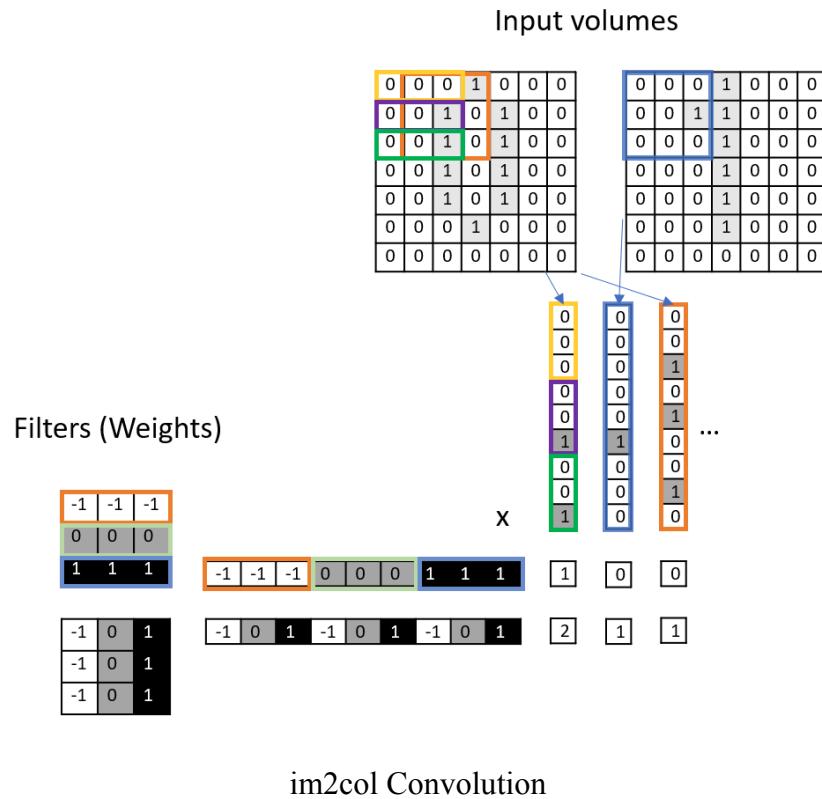
- $F(x,y)$ merupakan nilai piksel hasil konvolusi pada koordinat (x,y) di gambar output.
- $\sum_i \sum_j$ mengindikasikan bahwa operasi ini dilakukan sebagai penjumlahan dua kali. Hal ini berarti rumus ini akan menjumlahkan hasil dari semua piksel yang terlibat dalam operasi konvolusi pada area yang dilalui oleh kernel.
- $f(x+i,y+j)$ merupakan koordinat piksel pada gambar input yang sedang diproses. Indeks i dan j digunakan untuk menjelajahi nilai piksel dalam area lokal (biasanya ukuran kernel seperti 3×3 atau 5×5)
- $h(i,j)$ adalah fungsi kernel atau filter yang diaplikasikan pada gambar



Gambaran Umum Convolution

<table border="1"> <tbody> <tr><td>26</td><td>0</td><td>26</td><td>26</td><td>26</td><td>26</td><td>26</td></tr> <tr><td>0</td><td>26</td><td>0</td><td>26</td><td>26</td><td>26</td><td>26</td></tr> <tr><td>0</td><td>26</td><td>0</td><td>26</td><td>26</td><td>26</td><td>26</td></tr> </tbody> </table>	26	0	26	26	26	26	26	0	26	0	26	26	26	26	0	26	0	26	26	26	26	<table border="1"> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>26</td><td>0</td><td>26</td><td>26</td><td>26</td><td>26</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>26</td><td>0</td><td>26</td><td>26</td><td>26</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>26</td><td>0</td><td>26</td><td>26</td><td>26</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> </tbody> </table>	0	0	0	0	0	0	0	0	0	26	0	26	26	26	26	0	0	0	26	0	26	26	26	0	0	0	26	0	26	26	26	0	0	0	0	0	0	0	0	0	<table border="1"> <tbody> <tr><td>52</td><td>78</td><td>104</td><td>130</td><td>156</td><td>104</td></tr> <tr><td>78</td><td>104</td><td>156</td><td>182</td><td>234</td><td>156</td></tr> <tr><td>52</td><td>52</td><td>104</td><td>104</td><td>156</td><td>104</td></tr> </tbody> </table>	52	78	104	130	156	104	78	104	156	182	234	156	52	52	104	104	156	104
26	0	26	26	26	26	26																																																																											
0	26	0	26	26	26	26																																																																											
0	26	0	26	26	26	26																																																																											
0	0	0	0	0	0	0	0																																																																										
0	26	0	26	26	26	26	0																																																																										
0	0	26	0	26	26	26	0																																																																										
0	0	26	0	26	26	26	0																																																																										
0	0	0	0	0	0	0	0																																																																										
52	78	104	130	156	104																																																																												
78	104	156	182	234	156																																																																												
52	52	104	104	156	104																																																																												
<table border="1"> <tbody> <tr><td>26</td><td>26</td><td>26</td><td>26</td><td>0</td><td>26</td></tr> <tr><td>26</td><td>26</td><td>26</td><td>26</td><td>0</td><td>26</td></tr> <tr><td>26</td><td>26</td><td>26</td><td>0</td><td>0</td><td>0</td></tr> </tbody> </table>	26	26	26	26	0	26	26	26	26	26	0	26	26	26	26	0	0	0	<table border="1"> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>26</td><td>26</td><td>26</td><td>26</td><td>0</td><td>26</td><td>0</td></tr> <tr><td>0</td><td>26</td><td>26</td><td>26</td><td>26</td><td>0</td><td>26</td><td>0</td></tr> <tr><td>0</td><td>26</td><td>26</td><td>26</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> </tbody> </table>	0	0	0	0	0	0	0	0	0	26	26	26	26	0	26	0	0	26	26	26	26	0	26	0	0	26	26	26	0	0	0	0	0	0	0	0	0	0	0	0	<table border="1"> <tbody> <tr><td>104</td><td>156</td><td>156</td><td>104</td><td>104</td><td>52</td></tr> <tr><td>156</td><td>234</td><td>208</td><td>130</td><td>104</td><td>52</td></tr> <tr><td>104</td><td>156</td><td>130</td><td>78</td><td>52</td><td>26</td></tr> </tbody> </table>	104	156	156	104	104	52	156	234	208	130	104	52	104	156	130	78	52	26			
26	26	26	26	0	26																																																																												
26	26	26	26	0	26																																																																												
26	26	26	0	0	0																																																																												
0	0	0	0	0	0	0	0																																																																										
0	26	26	26	26	0	26	0																																																																										
0	26	26	26	26	0	26	0																																																																										
0	26	26	26	0	0	0	0																																																																										
0	0	0	0	0	0	0	0																																																																										
104	156	156	104	104	52																																																																												
156	234	208	130	104	52																																																																												
104	156	130	78	52	26																																																																												
<table border="1"> <tbody> <tr><td>0</td><td>26</td><td>0</td><td>0</td><td>26</td><td>0</td></tr> <tr><td>26</td><td>0</td><td>26</td><td>0</td><td>26</td><td>0</td></tr> <tr><td>26</td><td>0</td><td>26</td><td>26</td><td>26</td><td>26</td></tr> </tbody> </table>	0	26	0	0	26	0	26	0	26	0	26	0	26	0	26	26	26	26	<table border="1"> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>26</td><td>0</td><td>0</td><td>26</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>26</td><td>0</td><td>26</td><td>0</td><td>26</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>26</td><td>0</td><td>26</td><td>26</td><td>26</td><td>26</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> </tbody> </table>	0	0	0	0	0	0	0	0	0	0	26	0	0	26	0	0	0	26	0	26	0	26	0	0	0	26	0	26	26	26	26	0	0	0	0	0	0	0	0	0	<table border="1"> <tbody> <tr><td>52</td><td>78</td><td>52</td><td>78</td><td>52</td><td>52</td></tr> <tr><td>78</td><td>130</td><td>104</td><td>156</td><td>130</td><td>104</td></tr> <tr><td>52</td><td>104</td><td>78</td><td>130</td><td>104</td><td>78</td></tr> </tbody> </table>	52	78	52	78	52	52	78	130	104	156	130	104	52	104	78	130	104	78			
0	26	0	0	26	0																																																																												
26	0	26	0	26	0																																																																												
26	0	26	26	26	26																																																																												
0	0	0	0	0	0	0	0																																																																										
0	0	26	0	0	26	0	0																																																																										
0	26	0	26	0	26	0	0																																																																										
0	26	0	26	26	26	26	0																																																																										
0	0	0	0	0	0	0	0																																																																										
52	78	52	78	52	52																																																																												
78	130	104	156	130	104																																																																												
52	104	78	130	104	78																																																																												
	<table border="1"> <tbody> <tr><td>1</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	1	1	1	1	1	1	1	1	1																																																																							
1	1	1																																																																															
1	1	1																																																																															
1	1	1																																																																															

Direct Convolution



im2col Convolution

C. Kernel

Kernel berguna untuk memodifikasi gambar sehingga outputnya dapat diekstraksi pola-pola pentingnya. Beberapa kernel yang digunakan dalam proyek ini meliputi:

- **Box Blur:** Meratakan piksel dengan mengambil rata-rata nilai piksel sekitar.
- **Sharpening:** Menekankan detail gambar dengan memperkuat perbedaan antar piksel.
- **Edge Detection (Horizontal/Vertical):** Mengidentifikasi batas objek dalam gambar.
- **Sobel:** Deteksi tepi yang lebih sensitif, sering digunakan dalam *computer vision*.
- **Laplacian:** Deteksi tepi berbasis perbedaan nilai piksel kedua.

D. Matrix Multiplier Unit (MMU)

Matrix Multiplier Unit (MMU) dalam sistem ini menggunakan metode **im2col** untuk melakukan perkalian matriks dalam operasi konvolusi. Metode **im2col** (image to column) adalah teknik yang mengubah blok-blok kecil dari citra input menjadi baris-baris atau kolom

dalam bentuk matriks, sehingga memungkinkan proses konvolusi dilakukan sebagai operasi perkalian matriks konvensional.

E. Arithmetic Logic Unit (ALU)

ALU atau Arithmetic Logic Unit adalah komponen yang menjalankan operasi aritmatika dan logika. Dalam proyek ini:

- ALU digunakan untuk kernel sederhana yang tidak membutuhkan banyak komputasi (misalnya, Box Blur, Sharpening).
- Operasi ALU lebih ringan dibandingkan MMU, tetapi kurang efisien untuk kernel kompleks.

F. Decoder

Decoder adalah komponen yang menerjemahkan instruksi (opcode) ke dalam sinyal kontrol untuk menentukan:

- Jenis kernel yang digunakan.
- Unit eksekusi (ALU atau MMU).

G. Clamping

Clamping adalah teknik untuk memastikan nilai piksel hasil konvolusi berada dalam rentang valid (0-255). Operasi ini diperlukan karena hasil konvolusi sering kali melebihi rentang nilai warna yang valid.

H. Format Gambar BMP

BMP adalah format gambar yang sederhana, dengan header yang menyimpan informasi seperti dimensi gambar, jumlah bit per pixel, dan data pixel itu sendiri. Dalam proyek ini:

- File gambar BMP digunakan sebagai input dan output.
- Header diproses untuk membaca dimensi gambar dan data pixel nya.

1.3 OBJECTIVES

Berikut adalah tujuan dari proyek konvolusi gambar menggunakan VHDL yang telah dibuat dari kelompok kami:

1. Implementasi sistem konvolusi dengan dukungan pemilihan kernel dinamis melalui opcode
2. Pemrosesan gambar dalam format BMP
3. Mengoptimalkan kernel dengan algoritma im2col menggunakan MMU.

1.4 ROLES AND RESPONSIBILITIES

The roles and responsibilities assigned to the group members are as follows:

Roles	Responsibilities	Person
Ketua	Membuat kode ALU, MMU, CPU, dan Decoder beserta README	Bryan Herdianto
Anggota	Membuat laporan dan presentasi	Rowen Rodotua Harahap
Anggota	-	Filaga Tifira Muthi
Anggota	-	Calvin Kim

Table 1. Roles and Responsibilities

CHAPTER 2

IMPLEMENTATION

2.1 EQUIPMENT

The tools that are going to be used in this project are as follows:

- Visual Studio Code (coding VHDL)
- Modelsim (testing)

2.2 IMPLEMENTATION

A. Komponen

ALU (Arithmetic Logic Unit):

- Melakukan konvolusi dengan kernel sederhana (mis. Box Blur, Sharpening).
- Cocok untuk kernel yang tidak memerlukan komputasi intensif.

MMU (Matrix Multiplier Unit):

- Dioptimalkan untuk kernel kompleks yang memerlukan operasi perkalian matriks dengan presisi tinggi.
- Memanfaatkan logika **matrix multiplication** untuk akselerasi.

Decoder:

- Menerjemahkan **opcode** 4-bit menjadi:
 - **Opcode (3-bit):** Menentukan kernel yang digunakan.
 - **Address Flag (1-bit):** Menentukan penggunaan MMU atau ALU.

CPU:

- Mengontrol alur instruksi dari **fetch**, **decode**, hingga **execute**.
- Memutuskan apakah operasi akan dilakukan di ALU atau MMU berdasarkan **opcode**

B. Mapping Opcode

Instruksi	Deskripsi
0000	Kernel box Blur dengan ALU
0001	Kernel Horizontal edge detection dengan ALU
0010	Kervel Vertical Edge Detection dengan ALU
0011	Kernel Laplacian dengan ALUU
0100	Kernel Sharpening dengan ALU
0101	Kernel Sobel Horizontal dengan ALU
0110	Kernel Sobel Vertical dengan ALU
0111	Kernel Custom dengan ALU
1000	Kernel Box Blur dengan MMU
1001	Kernel Horizontal Edge Detection dengan MMU
1010	Kernel Vertical Edge Detection dengan MMU
1011	Kernel laplacian dengan MMU
1100	Kernel Sharpening dengan MMU
1101	Kernel Sobel Horizontal dengan MMU
1110	Kernel Sobel Vertical dengan MMU
1111	Kernel Custom dengan MMU

C. Implementasi

1. CPU:

CPU bertugas untuk mengontrol jalannya proses dari awal hingga selesai, termasuk mem-fetch instruksi, mendecode, dan mengeksekusi operasi berdasarkan opcode. CPU juga memutuskan apakah akan menggunakan ALU atau MMU berdasarkan instruksi yang diberikan.

Cara Kerja:

1. **Fetch:** Membaca instruksi dari input.
2. **Decode:** Menggunakan komponen Decoder untuk menerjemahkan instruksi 4-bit menjadi:
 - **Opcode (3-bit):** Menentukan kernel mana yang digunakan.
 - **Address Flag (1-bit):** Menentukan apakah menggunakan ALU atau MMU.
3. **Execute:** Mengaktifkan ALU atau MMU sesuai dengan hasil decoding.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY CPU IS
    PORT (
        -- Inputs
        CLK : IN STD_LOGIC;
        ENABLE : IN STD_LOGIC;
        RESET : IN STD_LOGIC;
        INSTRUCTION_RAW : IN STD_LOGIC_VECTOR(3 DOWNTO 0)
    );
END ENTITY CPU;

ARCHITECTURE RTL OF CPU IS

    -- Components
    COMPONENT DECODER IS
        PORT (
            CLK : IN STD_LOGIC;
```

```

        ENABLE          : IN STD_LOGIC;
        INSTRUCTION     : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        OPCODE          : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
        ADDRESS_FLAG    : OUT STD_LOGIC
    );
END COMPONENT DECODER;

COMPONENT ALU IS
PORT (
    CLK           : IN STD_LOGIC;
    ENABLE_ALU    : IN STD_LOGIC;
    OPCODE        : IN STD_LOGIC_VECTOR(2 DOWNTO 0)
);
END COMPONENT ALU;

COMPONENT MMU IS
PORT (
    CLK           : IN STD_LOGIC;
    ENABLE_MMU    : IN STD_LOGIC;
    OPCODE        : IN STD_LOGIC_VECTOR(2 DOWNTO 0)
);
END COMPONENT MMU;

-- Types

TYPE state_type IS (IDLE, FETCH, DECODE, EXECUTE, COMPLETE);

-- Signals

SIGNAL state      : state_type := IDLE;
SIGNAL PC         : INTEGER := 0;
SIGNAL ENABLE_ALU : STD_LOGIC := '0';

```

```

        SIGNAL ENABLE_MMU    : STD_LOGIC := '0';
        SIGNAL ADDRESS_FLAG : STD_LOGIC := '0';
        SIGNAL OPCODE       : STD_LOGIC_VECTOR(2 DOWNTO 0) := (OTHERS =>
'0');
        SIGNAL INSTRUCTION  : STD_LOGIC_VECTOR(3 DOWNTO 0) := (OTHERS =>
'0');

BEGIN

    -- Instantiate components and do port mapping

    DECODER_COMP : DECODER PORT MAP(
        CLK          => CLK,
        ENABLE       => ENABLE,
        INSTRUCTION  => INSTRUCTION,
        OPCODE       => OPCODE,
        ADDRESS_FLAG => ADDRESS_FLAG
    );

    ALU_COMP : ALU PORT MAP(
        CLK          => CLK,
        ENABLE_ALU   => ENABLE_ALU,
        OPCODE       => OPCODE
    );

    MMU_COMP : MMU PORT MAP(
        CLK          => CLK,
        ENABLE_MMU   => ENABLE_MMU,
        OPCODE       => OPCODE
    );

PROCESS (CLK, RESET) IS

```

```

BEGIN

    IF RESET = '1' THEN

        -- Reset all signals

        PC          <= 0;

        INSTRUCTION <= (OTHERS => '0');

        ENABLE_ALU  <= '0';

        ENABLE_MMU  <= '0';

        state       <= IDLE;

ELSIF RISING_EDGE(CLK) THEN

    IF ENABLE = '1' THEN

        CASE state IS

            WHEN IDLE =>

                PC <= PC + 1;

            IF PC = 1 THEN

                state <= FETCH;

            END IF;

            WHEN FETCH =>

                PC          <= PC + 1;

                INSTRUCTION <= INSTRUCTION_RAW;

            IF PC = 2 THEN

                state <= DECODE;

            END IF;

```

```

WHEN DECODE =>

PC <= PC + 1;

IF PC = 3 THEN
    state <= EXECUTE;
END IF;

WHEN EXECUTE =>

PC <= PC + 1;

IF ADDRESS_FLAG = '1' THEN
    ENABLE_MMU <= '1';
ELSE
    ENABLE_ALU <= '1';
END IF;

IF PC = 5 THEN
    state <= COMPLETE;
    ENABLE_ALU <= '0';
    ENABLE_MMU <= '0';
END IF;

WHEN COMPLETE =>

PC <= 0;
state <= IDLE;

END CASE;

```

```

        END IF;

    END IF;

END PROCESS;

END ARCHITECTURE RTL;

```

2. Decoder

Decoder menerjemahkan instruksi 4-bit menjadi:

- Opcode (3-bit): Menentukan kernel yang digunakan.
- Address Flag (1-bit): Menentukan apakah ALU atau MMU yang aktif.
- Decoder membaca instruksi dari CPU dan memisahkan 1 bit terakhir sebagai **Address Flag** dan 3 bit pertama sebagai **Opcode**.

```

LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

ENTITY DECODER IS

PORT (
    -- Inputs
    CLK          : IN STD_LOGIC;
    ENABLE       : IN STD_LOGIC;
    INSTRUCTION  : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    -- Outputs
    OPCODE       : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
    ADDRESS_FLAG : OUT STD_LOGIC
);

END ENTITY DECODER;

ARCHITECTURE RTL OF DECODER IS
BEGIN

```

```

PROCESS (CLK)
BEGIN
    IF RISING_EDGE(CLK) THEN
        -- Opcode determines the kernel operation
        -- Address flag determines which unit to use
        IF ENABLE = '1' THEN
            OPCODE      <= INSTRUCTION(2 DOWNTO 0);
            ADDRESS_FLAG <= INSTRUCTION(3);
        ELSE
            OPCODE      <= (OTHERS => '0');
            ADDRESS_FLAG <= '0';
        END IF;
    END IF;
END PROCESS;
END RTL;

```

3. ALU

ALU digunakan untuk menghitung operasi konvolusi pada gambar menggunakan kernel sederhana (misalnya Box Blur, Sharpening).

- Membuka file kernel berdasarkan opcode.
- Membaca file gambar BMP.
- Melakukan operasi konvolusi dengan kernel dan menyimpan hasilnya.
- Menulis gambar hasil konvolusi ke file output.

```
FOR i IN 1 TO image_height LOOP
```

```

row := image(i);

FOR j IN 1 TO image_width LOOP
    -- Compute the red channel with clamping
    red_result := INTEGER(
        kernel(0, 0) *
REAL(to_integer(unsigned(temp_image(i - 1)(j - 1).red))) +
        kernel(0, 1) *
REAL(to_integer(unsigned(temp_image(i - 1)(j).red))) +
        kernel(0, 2) *
REAL(to_integer(unsigned(temp_image(i - 1)(j + 1).red))) +
        kernel(1, 0) *
REAL(to_integer(unsigned(temp_image(i)(j - 1).red))) +
        kernel(1, 1) *
REAL(to_integer(unsigned(temp_image(i)(j).red))) +
        kernel(1, 2) *
REAL(to_integer(unsigned(temp_image(i)(j + 1).red))) +
        kernel(2, 0) *
REAL(to_integer(unsigned(temp_image(i + 1)(j - 1).red))) +
        kernel(2, 1) *
REAL(to_integer(unsigned(temp_image(i + 1)(j).red))) +
        kernel(2, 2) *
REAL(to_integer(unsigned(temp_image(i + 1)(j + 1).red)))
    );

```

;

```

IF red_result > 255 THEN
    red_result := 255; -- Clamp to 255
ELSIF red_result < 0 THEN
    red_result := 0; -- Clamp to 0 (in case of
underflow)
END IF;

row(j).red :=
STD_LOGIC_VECTOR(to_unsigned(red_result, 8));

```

-- Compute the green channel with clamping

```

green_result := INTEGER(
    kernel(0, 0) *
REAL(to_integer(unsigned(temp_image(i - 1)(j - 1).green))) +
    kernel(0, 1) *
REAL(to_integer(unsigned(temp_image(i - 1)(j).green))) +
    kernel(0, 2) *
REAL(to_integer(unsigned(temp_image(i - 1)(j + 1).green))) +
    kernel(1, 0) *
REAL(to_integer(unsigned(temp_image(i)(j - 1).green))) +
    kernel(1, 1) *
REAL(to_integer(unsigned(temp_image(i)(j).green))) +
    kernel(1, 2) *
REAL(to_integer(unsigned(temp_image(i)(j + 1).green))) +
    kernel(2, 0) *
REAL(to_integer(unsigned(temp_image(i + 1)(j - 1).green))) +
    kernel(2, 1) *
REAL(to_integer(unsigned(temp_image(i + 1)(j).green))) +
    kernel(2, 2) *
REAL(to_integer(unsigned(temp_image(i + 1)(j + 1).green)))
);

IF green_result > 255 THEN
    green_result := 255; -- Clamp to 255
ELSIF green_result < 0 THEN
    green_result := 0; -- Clamp to 0
END IF;

row(j).green :=
STD_LOGIC_VECTOR(to_unsigned(green_result, 8));

-- Compute the blue channel with clamping
blue_result := INTEGER(
    kernel(0, 0) *
REAL(to_integer(unsigned(temp_image(i - 1)(j - 1).blue))) +
    kernel(0, 1) *
REAL(to_integer(unsigned(temp_image(i - 1)(j).blue))) +
    kernel(0, 2) *
REAL(to_integer(unsigned(temp_image(i - 1)(j + 1).blue))) +
    kernel(1, 0) *
REAL(to_integer(unsigned(temp_image(i)(j - 1).blue))) +
    kernel(1, 1) *
REAL(to_integer(unsigned(temp_image(i)(j).blue))) +
    kernel(1, 2) *
REAL(to_integer(unsigned(temp_image(i)(j + 1).blue))) +
    kernel(2, 0) *
REAL(to_integer(unsigned(temp_image(i + 1)(j - 1).blue))) +
    kernel(2, 1) *
REAL(to_integer(unsigned(temp_image(i + 1)(j).blue))) +
    kernel(2, 2) *
REAL(to_integer(unsigned(temp_image(i + 1)(j + 1).blue)))
);

```

```

            kernel(0, 2) *
REAL(to_integer(unsigned(temp_image(i - 1)(j + 1).blue))) +
            kernel(1, 0) *
REAL(to_integer(unsigned(temp_image(i)(j - 1).blue))) +
            kernel(1, 1) *
REAL(to_integer(unsigned(temp_image(i)(j).blue))) +
            kernel(1, 2) *
REAL(to_integer(unsigned(temp_image(i)(j + 1).blue))) +
            kernel(2, 0) *
REAL(to_integer(unsigned(temp_image(i + 1)(j - 1).blue))) +
            kernel(2, 1) *
REAL(to_integer(unsigned(temp_image(i + 1)(j).blue))) +
            kernel(2, 2) *
REAL(to_integer(unsigned(temp_image(i + 1)(j + 1).blue)))

);

IF blue_result > 255 THEN
    blue_result := 255; -- Clamp to 255
ELSIF blue_result < 0 THEN
    blue_result := 0; -- Clamp to 0
END IF;

row(j).blue :=
STD_LOGIC_VECTOR(to_unsigned(blue_result, 8));

END LOOP;
END LOOP;

```

4. MMU

MMU mempercepat proses konvolusi dengan mendukung kernel kompleks (misalnya Sobel, Laplacian). MMU dirancang untuk menghitung dot product (perkalian matriks).

- Membaca kernel dari file.
- Memproses gambar BMP sebagai matriks.

- Melakukan perkalian matriks menggunakan kernel untuk menghasilkan gambar keluaran.
- Menulis gambar hasil konvolusi ke file output.

```

FOR i IN 0 TO image_height * image_width - 1 LOOP

    row := image(i / image_width);

    -- Initialize the sums to 0
    sum_red      := 0.0;
    sum_blue     := 0.0;
    sum_green    := 0.0;

    -- Iterate over each column of resized image
matrix

    FOR j IN 0 TO KERNEL_SIZE * KERNEL_SIZE - 1 LOOP

        -- Compute the dot product of the kernel and
the image matrix

        sum_red      := sum_red +
REAL(to_integer(unsigned(image_resized(i)(j).red))) * kernel_flat(j);

        sum_blue     := sum_blue +
REAL(to_integer(unsigned(image_resized(i)(j).blue))) * kernel_flat(j);

        sum_green    := sum_green +
REAL(to_integer(unsigned(image_resized(i)(j).green))) * kernel_flat(j);

    END LOOP;

    -- Clamp the values to 0-255 for red channel
    IF sum_red < 0.0 THEN
        sum_red := 0.0;
    ELSIF sum_red > 255.0 THEN
        sum_red := 255.0;
    END IF;
END LOOP;

```

```

        END IF;

-- Clamp the values to 0-255 for blue channel

IF sum_blue < 0.0 THEN
    sum_blue := 0.0;
ELSIF sum_blue > 255.0 THEN
    sum_blue := 255.0;
END IF;

-- Clamp the values to 0-255 for green channel

IF sum_green < 0.0 THEN
    sum_green := 0.0;
ELSIF sum_green > 255.0 THEN
    sum_green := 255.0;
END IF;

-- Assign the new pixel values to the pointer rows
for output image

    row(i MOD image_width).red    :=
STD_LOGIC_VECTOR(to_unsigned(INTEGER(sum_red), 8));

    row(i MOD image_width).blue   :=
STD_LOGIC_VECTOR(to_unsigned(INTEGER(sum_blue), 8));

    row(i MOD image_width).green  :=
STD_LOGIC_VECTOR(to_unsigned(INTEGER(sum_green), 8));

END LOOP;

```

CHAPTER 3

TESTING AND ANALYSIS

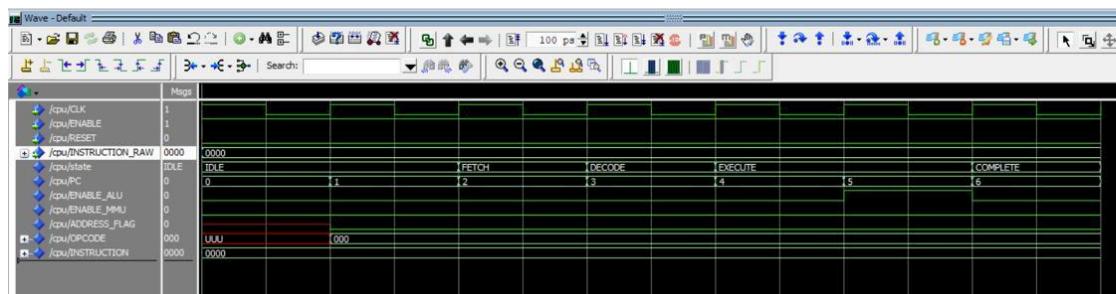
3.1 TESTING

A. Box Blur

Kernel yang dipakai:

$$K = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$

Hasil Modelsim



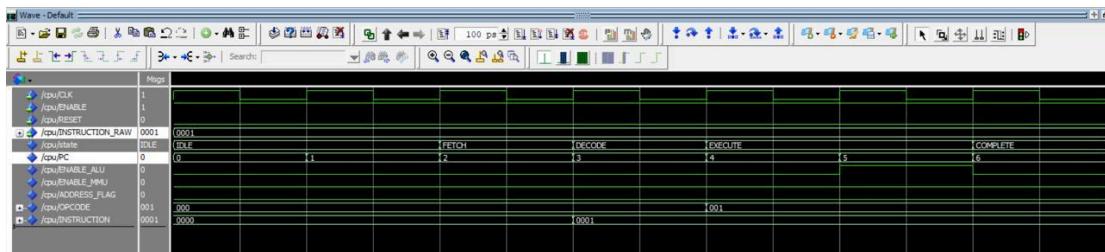
Operasi yang dilakukan adalah mengalikan kernel ini dengan nilai piksel gambar di sekitar piksel target, lalu menjumlahkan hasilnya. Nilai akhirnya menjadi nilai baru piksel target, menghasilkan efek “blur” atau pelembutan gambar. Gambar ini adalah hasil simulasi dari ModelSim, yang menunjukkan bagaimana kernel Box Blur diterapkan dalam implementasi VHDL.

B. Horizontal Edge

Kernel yang dipakai:

$$K = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Hasil Modelsim



Matriks ini dirancang untuk mendeteksi perubahan intensitas horizontal dalam gambar. Berikut cara kerja kernel di atas:

- Baris pertama $(-1, -1, -1)$ menangkap piksel gelap di atas.
- Baris ketiga $(1, 1, 1)$ menangkap piksel terang di bawah.
- Baris tengah $(0, 0, 0)$ mengabaikan piksel di tingkat yang sama.

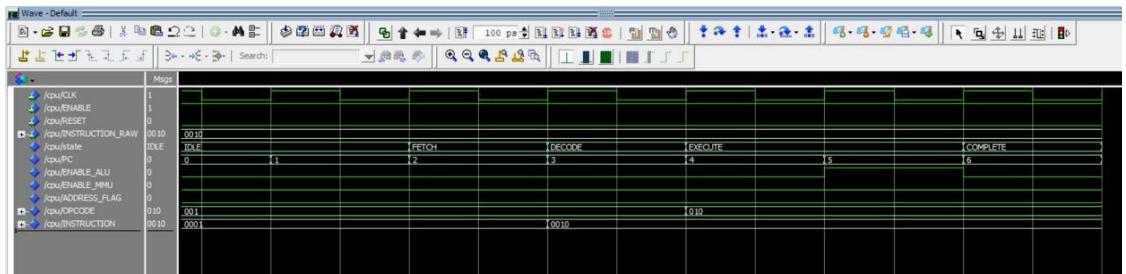
Hasil konvolusi menggunakan kernel ini akan menghasilkan garis-garis horizontal yang mencolok

C. Vertical Edge

Kernel yang dipakai:

$$K = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Hasil Modelsim



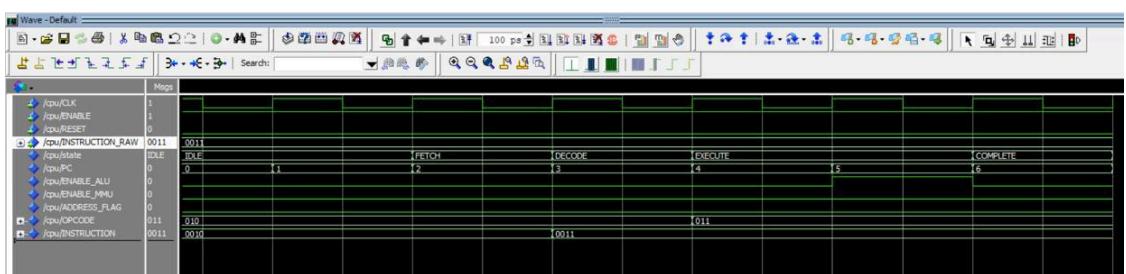
Kernel mendeteksi perubahan intensitas dalam arah vertikal dengan bobot negatif di kolom kiri, nol di tengah, dan positif di kolom kanan. Sama seperti horizontal edge, ALU menghitung dot product kernel dan area piksel untuk setiap piksel gambar. Tapi vertikal terlihat lebih jelas pada gambar output. Instruksi 0010 mengarahkan ALU untuk menerapkan kernel ini.

D. Laplacian

Kernel yang dipakai:

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

Hasil Modelsim



Kernel Laplacian dirancang untuk mendeteksi perubahan kedua (second-order changes) dalam intensitas gambar.

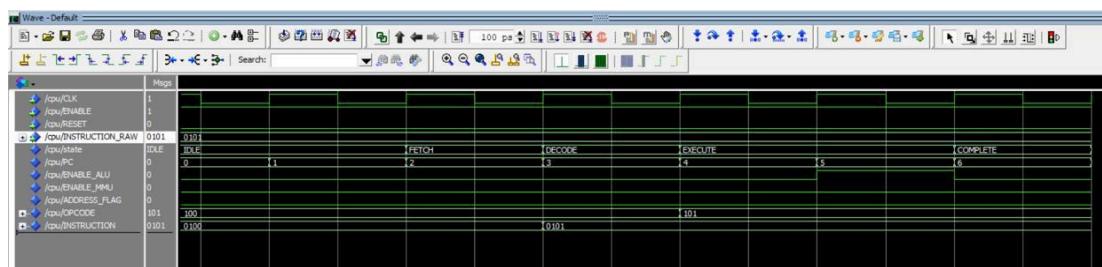
ALU menghitung hasil dari kernel dengan nilai pusat tinggi (biasanya positif) dan nilai negatif di sekitarnya. Proses ini mendeteksi tepi dengan lebih halus dan sensitif terhadap detail. Instruksi 0011 digunakan untuk menjalankan kernel ini melalui ALU.

E. Sobel Horizontal

Kernel yang dipakai:

$$K = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Hasil Modelsim



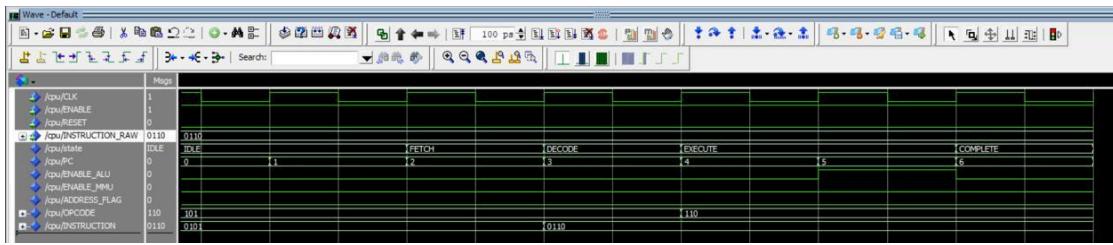
Sobel horizontal dirancang untuk memberikan sensitivitas lebih tinggi terhadap perubahan intensitas horizontal dibandingkan kernel horizontal edge biasa. ALU memproses kernel ini dengan bobot yang lebih besar pada kolom tengah, menghasilkan efek deteksi tepi horizontal yang lebih tajam. Instruksi 0101 memicu eksekusi kernel Sobel Horizontal di ALU.

F. Sobel Vertical

Kernel yang dipakai:

$$K = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Hasil Modelsim



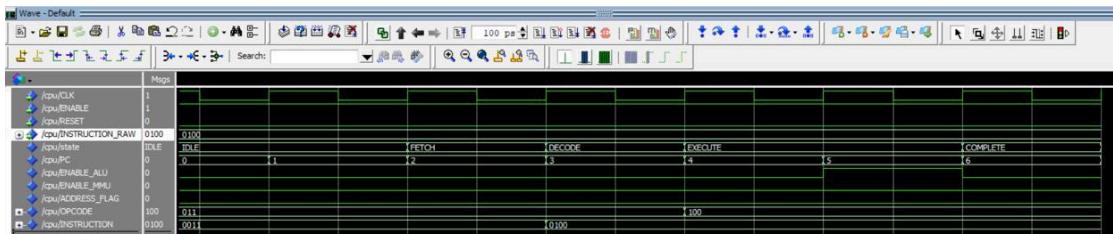
Sobel vertical memberikan sensitivitas tinggi terhadap perubahan intensitas vertikal. Sama seperti Sobel Horizontal, kernel diterapkan pada area piksel oleh ALU untuk mendeteksi perubahan intensitas vertikal secara detail. Instruksi 0110 digunakan untuk menjalankan proses ini.

G. Sharpening

Kernel yang digunakan untuk sharpening adalah sebagai berikut.

$$K = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Hasil Modelsim

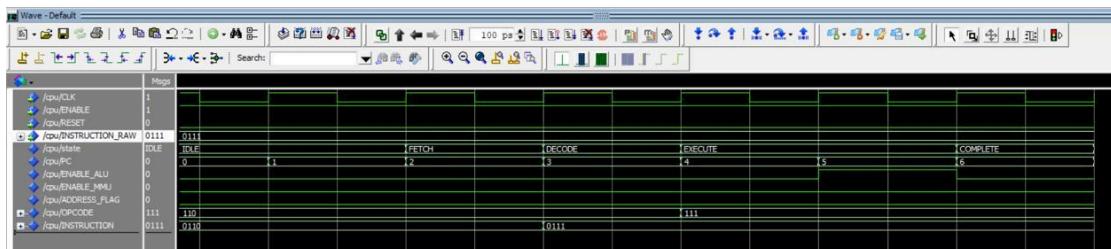


Kernel sharpening menekankan perbedaan intensitas antar piksel dengan memberikan bobot tinggi pada nilai pusat kernel. ALU menghitung nilai piksel dengan kernel ini, menghasilkan gambar output yang lebih tajam dan detail. Proses ini diaktifkan oleh instruksi 0100.

H. Custom

Untuk custom kernel, user dapat menginput kernel 3x3 yang diinginkannya pada file “custom.txt”. Untuk setiap angka dalam kernel, dipastikan ada spasi. Kernel bisa berisi angka pecahan yang direpresentasikan dengan adanya tanda bagi atau *slash*, atau bisa angka desimal (*real*).

Hasil Modelsim



Custom kernel memberikan fleksibilitas kepada pengguna untuk menentukan nilai kernel sesuai kebutuhan. Kernel ini diproses menggunakan MMU karena kompleksitas perhitungan matrix multiplication yang lebih besar. MMU membaca kernel dari file input, melakukan perkalian matriks dengan area piksel, dan menghasilkan nilai piksel output. Instruksi 1111 diarahkan ke MMU untuk menjalankan operasi custom kernel ini.

Semua kernel di atas harus melalui tahapan-tahapan berikut dalam CPU sebagai salah satu implementasi dari FSM:

1. **FETCH**: Instruksi dibaca dari input, seperti 0000 untuk Box Blur atau 1111 untuk Custom Kernel.
2. **DECODE**: Decoder memetakan instruksi menjadi sinyal kontrol (opcode dan flag alamat).
3. **EXECUTE**: ALU atau MMU diaktifkan sesuai instruksi, dan kernel diterapkan pada area piksel.
4. **COMPLETE**: Hasil konvolusi ditulis ke file output (gambar BMP).

3.2 RESULT

A. Original



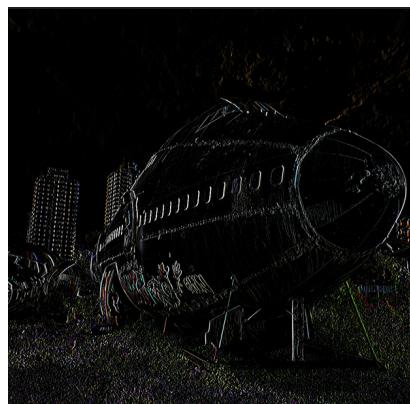
B. Box Blur



C. Horizontal Edge



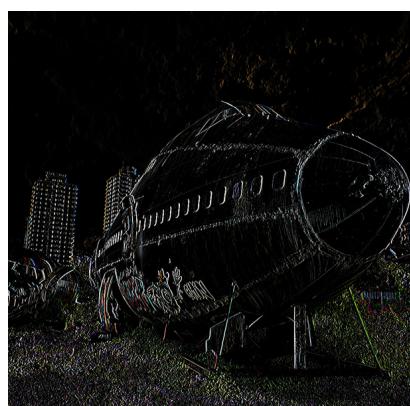
D. Vertical Edge



E. Laplacian



F. Sobel Horizontal



G. Sobel Vertical



H. Sharpening



CHAPTER 4

CONCLUSION

Proyek ini berhasil mengembangkan sistem konvolusi gambar berbasis VHDL yang memanfaatkan integrasi antara **Matrix Multiplier Unit (MMU)** dan **Arithmetic Logic Unit (ALU)**. Kombinasi ini memungkinkan pengolahan berbagai kernel konvolusi, mulai dari yang sederhana hingga kompleks, secara dinamis melalui penggunaan opcode. Pendekatan ini memberikan fleksibilitas tinggi dalam mengatur jenis kernel, termasuk kernel custom yang dapat ditentukan pengguna melalui file input.

Sistem ini juga dirancang untuk memastikan nilai piksel hasil konvolusi berada dalam rentang valid (0-255) dengan menerapkan teknik *clamping*. Hal ini menjaga keakuratan hasil pemrosesan gambar tanpa menyebabkan artefak akibat nilai piksel yang keluar dari batas. Keandalan sistem semakin ditingkatkan dengan dukungan algoritma konvolusi yang dapat diterapkan melalui dua metode utama: *direct convolution* dan im2col.

Metode im2col menawarkan efisiensi lebih tinggi dalam proses konvolusi dengan mengoptimalkan perkalian matriks, yang merupakan inti dari operasi konvolusi itu sendiri. Dengan algoritma ini, sistem dapat mencapai kecepatan pemrosesan yang lebih baik dibandingkan metode konvolusi langsung, sehingga cocok untuk berbagai aplikasi yang membutuhkan kinerja tinggi dalam pengolahan gambar.

REFERENCES

- [1] J. J. Jensen, “BMP file bitmap image read using TEXTIO,” *VHDLwhiz*, Aug. 11, 2024. <https://vhdlwhiz.com/read-bmp-file/>
- [2] GeeksforGeeks, “Types of convolution kernels,” *GeeksforGeeks*, Jul. 22, 2024. <https://www.geeksforgeeks.org/types-of-convolution-kernels/>
- [3] E. Roe, “Convolution: Image filters, CNNs and examples in Python & PyTorch,” *Medium*, Jun. 08, 2023. [Online]. Available: https://medium.com/@er_95882/convolution-image-filters-cnns-and-examples-in-pytorch-bd3f3ac5df9c
- [4] S. Team, “VHDL component and port map tutorial,” *Invent Logics*, Jan. 10, 2018. <https://allaboutfpga.com/vhdl-component-port-map-tutorial/>
- [5] “FPGA Projects, Verilog Projects, VHDL Projects - FPGA4student.com.” <https://www.fpga4student.com/>