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

AES ENCRYPTION HARDWARE ACCELERATOR

GROUP PA - 16

Fathan Yazid Satriani	**2306250560**
Mutia Casella	**2306202870**
Muhammad Rafli	**2306250730**
Ahmad Malik Ramadhino	**2206059370**

FAKULTAS TEKNIK

UNIVERSITAS INDONESIA

DESEMBER 2024

PREFACE

Puji syukur kami panjatkan kehadirat Tuhan Yang Maha Esa atas berkat dan rahmat-Nya sehingga kami dapat menyelesaikan laporan proyek akhir ini dengan baik. Laporan yang merupakan salah satu syarat kelulusan praktikum Perancangan Sistem Digital ini bertujuan untuk mendesain, mengimplementasikan, serta menguji algoritma *Advanced Encryption Standard* (AES) menggunakan VHDL.

Kami mengucapkan terima kasih kepada Bapak Ruki Harwahyu, S.T., M.Sc., M.T., Ph.D., Bapak Fransiskus Astha Ekadiyanto, dan Bapak Yan Maraden, S.T., M.T., selaku dosen pengampu mata kuliah Perancangan Sistem Digital, serta Kak Giovan Christoffel Sihombing yang telah memberikan bimbingan dan arahan selama proses pengerjaan proyek akhir ini. Kami juga ingin menyampaikan apresiasi kepada rekan satu kelompok atas kerja sama yang baik sehingga proyek ini dapat diselesaikan tepat waktu.

Kami menyadari bahwa laporan ini masih memiliki kekurangan, sehingga kami sangat terbuka terhadap kritik dan saran yang membangun. Akhir kata, kami berharap laporan ini dapat memberikan manfaat bagi pembaca, khususnya dalam memahami penerapan algoritma kriptografi pada sistem digital berbasis hardware.

Depok, 8 Desember 2024

Kelompok PSD PA-16

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Keamanan data telah menjadi perhatian utama dalam era digital saat ini, di mana setiap informasi yang disimpan dan ditransmisikan dapat dengan mudah diakses oleh pihak yang tidak diinginkan. Oleh karena itu, dikembangkan berbagai metode - metode kriptografi yang mengenkripsi data supaya tidak bisa diinterpretasikan oleh pihak yang tidak berwenang. Salah satu metode kriptografi yang banyak digunakan untuk melindungi data adalah algoritma Advanced Encryption Standard (AES). AES merupakan algoritma simetris yang aman dan efisien sehingga telah diterima secara luas sebagai standar enkripsi oleh pemerintah dan industri di seluruh dunia.

AES bekerja dengan mengenkripsi data dalam blok berukuran 128-bit menggunakan sebuah kunci enkripsi. Proses enkripsi AES terdiri dari beberapa tahap, termasuk SubBytes, ShiftRows, MixColumns, dan AddRoundKey, yang dilakukan dalam beberapa ronde tergantung pada panjang kunci yang digunakan. Keamanan dan efisiensi AES menjadikannya pilihan utama dalam aplikasi yang memerlukan enkripsi cepat dan aman, seperti komunikasi data, penyimpanan file, dan transaksi elektronik.

Namun, meskipun AES sangat efisien pada perangkat lunak, kecepatan eksekusi algoritma ini masih terbatas oleh kemampuan pemrosesan perangkat keras. Oleh karena itu, implementasi AES pada perangkat keras, terutama menggunakan FPGA (Field-Programmable Gate Array), merupakan salah satu solusi untuk meningkatkan kecepatan dan efisiensi pemrosesan enkripsi dan dekripsi. FPGA memungkinkan penggunaan paralelisme dan pipelining untuk mempercepat operasi utama AES, menjadikannya lebih cocok untuk aplikasi yang memerlukan pengolahan data dalam jumlah besar serta dengan kecepatan tinggi.

1.2 PROJECT DESCRIPTION

Proyek ini bertujuan untuk mengimplementasikan akselerator enkripsi AES berbasis VHDL yang dijalankan pada FPGA untuk memproses enkripsi dan dekripsi data secara cepat dan efisien. Akselerator ini akan menggunakan teknik paralelisme dan pipelining untuk

meningkatkan performa dari setiap operasi yang dilakukan oleh algoritma AES, yakni SubBytes, ShiftRows, MixColumns, dan AddRoundKey. Algoritma ini akan dijalankan dalam 10 ronde untuk kunci 128-bit, dengan input berupa blok plaintext 128-bit dan kunci enkripsi, serta output berupa ciphertext yang terenkripsi.

Desain akselerator ini bertujuan untuk mencapai kinerja yang lebih tinggi dibandingkan implementasi perangkat lunak, serta menyediakan kemampuan untuk melakukan dekripsi menggunakan algoritma invers dari langkah enkripsi.

1.3 OBJECTIVES

Tujuan dari proyek ini adalah sebagai berikut:

1. Mendesain dan mengimplementasikan algoritma AES-128 berbasis VHDL pada FPGA.
2. Memanfaatkan paralelisme dan pipeline untuk meningkatkan kinerja pada proses enkripsi dan dekripsi.
3. Menguji sistem dalam memproses data enkripsi dan dekripsi menggunakan FPGA.
4. Meningkatkan pemahaman tentang implementasi algoritma kriptografi pada perangkat keras.

1.4 ROLES AND RESPONSIBILITIES

The responsibilities assigned to the group members are as follows:

Responsibilities	Person
● Membuat program bagian Encryption dan Testbench untuk Encryption. ● Membuat program Encryption dalam Bahasa C juga untuk menguji kebenaran hasil dengan program VHDL. ● Membantu menulis dokumentasi pada laporan (chapter 1 dan 2) dan readme.	Fathan Yazid Satriani
● Membuat program VHDL bagian Decryption dan Testbench untuk Decryption.	Mutia Casella

● Membantu menulis dokumentasi pada laporan, powerpoint, dan readme.	
● Membuat program VHDL bagian Decryption dan Testbench untuk Decryption. ● Membantu menulis dokumentasi pada laporan, powerpoint, dan readme.	Muhammad Rafli
● Tidak membantu sama sekali.	Ahmad Malik Ramadhino

Table 1. Roles and Responsibilities

CHAPTER 2

IMPLEMENTATION

2.1 EQUIPMENT

Berikut adalah alat (software) yang digunakan dalam pengembangan proyek ini beserta fungsinya:

- GitHub : Platform penyimpanan dan manajemen proyek
- Visual Studio Code : IDE untuk penulisan kode VHDL
- ModelSim : Mensimulasikan dan menguji kebenaran output
- Quartus Prime : Sintesis RTL

2.2 IMPLEMENTATION

Implementasi dari proyek AES 128-bit ini dalam Bahasa VHDL dibagi menjadi dua bagian utama, yaitu enkripsi dan dekripsi. Kedua bagian ini menjalankan beberapa proses utama AES yang mencakup beberapa operasi utama pada blok data, yaitu operasi SubBytes, ShiftRows, MixColumns, dan AddRoundKey. Operasi ini akan dijalankan dalam beberapa ronde yang bergantung panjang Key yang digunakan. Untuk implementasi ini, karena panjang input dan kunci yang digunakan adalah 128-bit, maka program AES ini akan berjalan selama 10 ronde.

Pada bagian enkripsi ini terbagi menjadi tiga file, yaitu **AES_Package.vhd**, **AES_Encryption.vhd**, dan **AES_Encryption_Testbench.vhd**. Pada file **AES_Package.vhd**, terdapat pendefinisian tipe data yang akan digunakan dalam algoritma AES, yaitu ***state_array***. Ini adalah tipe data array dua dimensi yang digunakan untuk merepresentasikan status atau blok data 4x4 byte (128-bit), yang merupakan format input dan output dalam AES.

```
      -- AES_Package.vhd
      library IEEE;
      use IEEE.STD_LOGIC_1164.ALL;
      package aes_package is
          type state_array is array (0 to 3, 0 to 3) of
STD_LOGIC_VECTOR(7 downto 0);
      end package;
```

Selanjutnya, file **AES_Encryption.vhd** adalah implementasi utama dari enkripsi AES. Pada bagian entity file ini, terdapat pendeklarasian berbagai port yang digunakan input dan output, sebagai berikut:

- ***clk*** dan ***rst*** adalah sinyal clock dan reset yang digunakan untuk mengendalikan waktu dan sinkronisasi operasi yang dilakukan program.
- ***data_in*** adalah input data berukuran 128-bit yang akan dienkripsi.
- ***key*** adalah kunci berukuran 128-bit yang digunakan untuk enkripsi yang unik.
- ***data_out*** adalah hasil enkripsi (ciphertext).
- ***done*** adalah sinyal yang menunjukkan bahwa proses enkripsi telah selesai.
- Debug ports seperti ***debug_state_out*** dan yang lainnya digunakan untuk menguji kebenaran program (debugging). Beberapa port output ini akan mengeluarkan status / hasil dari setiap tahap enkripsi.

```
entity AES_Encryption is
    Port (
        clk : in STD_LOGIC;
        rst : in STD_LOGIC;
        data_in : in STD_LOGIC_VECTOR(127 downto 0);
        key : in STD_LOGIC_VECTOR(127 downto 0);
        data_out : out STD_LOGIC_VECTOR(127 downto 0);
        done : out STD_LOGIC;
        -- Debug ports
        debug_state_out : out state_array;
        debug_sub_bytes_out : out state_array;
        debug_shift_rows_out : out state_array;
        debug_mix_cols_out : out state_array;
        debug_round_key_out : out STD_LOGIC_VECTOR(127 downto 0)
    );
end AES_Encryption;
```

Setelah port - port pada bagian entity, kita mulai masuk ke dalam bagian arsitektur program di mana pertama - tama terdapat berbagai sinyal internal yang digunakan dalam proses enkripsi. Sinyal - sinyal ini adalah sebagai berikut:

- ***state*** adalah data bertipe array yang menyimpan status / nilai dari data yang sedang diproses.
- ***round*** digunakan untuk melacak jumlah ronde dalam proses enkripsi.
- ***round_keys*** adalah array yang menyimpan kunci yang diperluas untuk setiap ronde.
- ***current_round_key*** adalah kunci ronde yang digunakan dalam ronde tertentu.

- ***key_expanded*** dan ***initial_state_loaded*** adalah sinyal untuk digunakan untuk mengecek apakah kunci sudah diperluas dan apakah status awal sudah dimuat supaya program bisa lanjut ke tahap berikutnya.

```
-- State signals
    signal state : state_array;
    signal round : integer range 0 to 10;
    signal done_i : STD_LOGIC;
    signal round_keys : STD_LOGIC_VECTOR(1407 downto 0);
    signal current_round_key : STD_LOGIC_VECTOR(127 downto 0);
    signal key_expanded : STD_LOGIC := '0';
    signal initial_state_loaded : STD_LOGIC := '0';
```

Setelah deklarasi sinyal internal terdapat deklarasi Tabel S-Box dan juga Rcon. Kedua konstanta berbentuk array ini digunakan untuk operasi SubBytes serta pembaruan kunci ronde. S-Box adalah tabel substitusi yang berisi nilai-nilai untuk transformasi SubBytes dan Rcon adalah konstanta ronde yang digunakan dalam operasi penambahan ronde kunci (AddRoundKey).

```
-- S-box and round constants
    type sbox_array is array (0 to 255) of STD_LOGIC_VECTOR(7 downto
0);
    constant SBOX : sbox_array := (
        X"63", X"7c", X"77", X"7b", X"f2", X"6b", X"6f", X"c5", X"30",
X"01", X"67", X"2b", X"fe", X"d7", X"ab", X"76",
        X"ca", X"82", X"c9", X"7d", X"fa", X"59", X"47", X"f0", X"ad",
X"d4", X"a2", X"af", X"9c", X"a4", X"72", X"c0",
        X"b7", X"fd", X"93", X"26", X"36", X"3f", X"f7", X"cc", X"34",
X"a5", X"e5", X"f1", X"71", X"d8", X"31", X"15",
        X"04", X"c7", X"23", X"c3", X"18", X"96", X"05", X"9a", X"07",
X"12", X"80", X"e2", X"eb", X"27", X"b2", X"75",
        X"09", X"83", X"2c", X"1a", X"1b", X"6e", X"5a", X"a0", X"52",
X"3b", X"d6", X"b3", X"29", X"e3", X"2f", X"84",
        X"53", X"d1", X"00", X"ed", X"20", X"fc", X"b1", X"5b", X"6a",
X"cb", X"be", X"39", X"4a", X"4c", X"58", X"cf",
        X"d0", X"ef", X"aa", X"fb", X"43", X"4d", X"33", X"85", X"45",
X"f9", X"02", X"7f", X"50", X"3c", X"9f", X"a8",
        X"51", X"a3", X"40", X"8f", X"92", X"9d", X"38", X"f5", X"bc",
X"b6", X"da", X"21", X"10", X"ff", X"f3", X"d2",
        X"cd", X"0c", X"13", X"ec", X"5f", X"97", X"44", X"17", X"c4",
X"a7", X"7e", X"3d", X"64", X"5d", X"19", X"73",
        X"60", X"81", X"4f", X"dc", X"22", X"2a", X"90", X"88", X"46",
X"ee", X"b8", X"14", X"de", X"5e", X"0b", X"db",
        X"e0", X"32", X"3a", X"0a", X"49", X"06", X"24", X"5c", X"c2",
X"d3", X"ac", X"62", X"91", X"95", X"e4", X"79",
        X"e7", X"c8", X"37", X"6d", X"8d", X"d5", X"4e", X"a9", X"6c",
X"56", X"f4", X"ea", X"65", X"7a", X"ae", X"08",
        X"ba", X"78", X"25", X"2e", X"1c", X"a6", X"b4", X"c6", X"e8",
X"dd", X"74", X"1f", X"4b", X"bd", X"8b", X"8a",
        X"70", X"3e", X"b5", X"66", X"48", X"03", X"f6", X"0e", X"61",
X"35", X"57", X"b9", X"86", X"c1", X"1d", X"9e",
```

```
        X"e1", X"f8", X"98", X"11", X"69", X"d9", X"8e", X"94", X"9b",
X"1e", X"87", X"e9", X"ce", X"55", X"28", X"df",
        X"8c", X"a1", X"89", X"0d", X"bf", X"e6", X"42", X"68", X"41",
X"99", X"2d", X"0f", X"b0", X"54", X"bb", X"16"
    );

    type rcon_array is array (1 to 10) of STD_LOGIC_VECTOR(7 downto 0);
    constant rcon : rcon_array := (
        X"01", X"02", X"04", X"08", X"10", X"20", X"40", X"80", X"1B",
X"36"
    );
```

Selanjutnya ada fungsi ***state_to_output*** yang digunakan untuk mengkonversi status 2D dari ***state_array*** menjadi 1D dalam bentuk ***STD_LOGIC_VECTOR(127 downto 0)***. Ini digunakan supaya nilai hasil setiap operasi dapat disalurkan ke output. Kemudian, ada juga fungsi ***to_hstring*** yang mengkonversi vektor ***std_logic_vector*** ke dalam format string yang mewakili nilai hexadecimal. Fungsi ini bekerja dengan cara membagi vektor bit menjadi nibbles (4 bit), kemudian mengkonversi setiap nibble menjadi karakter hexadecimal (0-9, a-f).

```
function state_to_output(state_in: state_array) return STD_LOGIC_VECTOR
is
        variable result : STD_LOGIC_VECTOR(127 downto 0);
    begin
        for i in 0 to 3 loop
            for j in 0 to 3 loop
                result(127-8*(4*i+j) downto 120-8*(4*i+j)) :=
state_in(j,i);
            end loop;
        end loop;
        return result;
    end function;

    function to_hstring(slv: std_logic_vector) return string is
        variable result : string(1 to slv'length/4);
        variable nibble : std_logic_vector(3 downto 0);
    begin
        for i in result'range loop
            nibble := slv(slv'length-i*4-1 downto slv'length-i*4-4);
            case to_integer(unsigned(nibble)) is
                when  0 => result(i) := '0';
                when  1 => result(i) := '1';
                when  2 => result(i) := '2';
                when  3 => result(i) := '3';
                when  4 => result(i) := '4';
                when  5 => result(i) := '5';
                when  6 => result(i) := '6';
                when  7 => result(i) := '7';
                when  8 => result(i) := '8';
                when  9 => result(i) := '9';
                when 10 => result(i) := 'a';
                when 11 => result(i) := 'b';
```

```
                when 12 => result(i) := 'c';
                when 13 => result(i) := 'd';
                when 14 => result(i) := 'e';
                when 15 => result(i) := 'f';
                when others => result(i) := 'x';
            end case;
        end loop;
        return result;
    end function;
```

Setelah itu fungsi tersebut terdapat fungsi - fungsi lainnya yang digunakan pada operasi utama enkripsi AES, fungsi - fungsi ini adalah sebagai berikut:

- Fungsi ***sub_bytes*** adalah fungsi yang melakukan operasi substitusi pada setiap byte data dengan mengaplikasikan tabel S-Box.

```
-- SubBytes transformation
    function sub_bytes(state_in: state_array) return
state_array is
        variable result : state_array;
    begin
        for i in 0 to 3 loop
            for j in 0 to 3 loop
                result(i,j) :=
SBOX(to_integer(unsigned(state_in(i,j))));
            end loop;
        end loop;
        return result;
    end function;
```

- Fungsi ***shift_rows*** menggeser baris - baris data sesuai dengan aturan AES, yaitu: baris pertama tidak digeser, baris kedua digeser satu posisi ke kiri, dan baris ketiga digeser dua posisi.

```
-- ShiftRows transformation
    function shift_rows(state_in: state_array) return
state_array is
        variable result : state_array;
    begin
        -- Row 0: no shift
        result(0,0) := state_in(0,0);
        result(0,1) := state_in(0,1);
        result(0,2) := state_in(0,2);
        result(0,3) := state_in(0,3);

        -- Row 1: shift left by 1
        result(1,0) := state_in(1,1);
        result(1,1) := state_in(1,2);
        result(1,2) := state_in(1,3);
        result(1,3) := state_in(1,0);
```

```
        -- Row 2: shift left by 2
        result(2,0) := state_in(2,2);
        result(2,1) := state_in(2,3);
        result(2,2) := state_in(2,0);
        result(2,3) := state_in(2,1);

        -- Row 3: shift left by 3
        result(3,0) := state_in(3,3);
        result(3,1) := state_in(3,0);
        result(3,2) := state_in(3,1);
        result(3,3) := state_in(3,2);

        return result;
    end function;
```

- Fungsi ***gf_mult*** melakukan perkalian dalam field Galois GF(2^8), yang digunakan dalam transformasi MixColumns. Fungsi ini mengalikan dua byte menggunakan operasi XOR dan pergeseran bit. Jika bit tertinggi dari hasil perkalian adalah 1, maka hasil perkalian akan direduksi dengan polinomial X^8 + X^4 + X^3 + X + 1.

```
-- GF(2^8) multiplication
    function gf_mult(a, b: STD_LOGIC_VECTOR(7 downto 0))
return STD_LOGIC_VECTOR is
        variable p : STD_LOGIC_VECTOR(7 downto 0);
        variable hi_bit : STD_LOGIC;
        variable temp : STD_LOGIC_VECTOR(7 downto 0);
    begin
        p := (others => '0');
        temp := a;

        for i in 0 to 7 loop
            if (b(i) = '1') then
                p := p xor temp;
            end if;
            hi_bit := temp(7);
            temp := temp(6 downto 0) & '0';
            if (hi_bit = '1') then
                temp := temp xor X"1B";  -- Reduction
polynomial
            end if;
        end loop;
        return p;
    end function;
```

- Fungsi ***mix_columns*** bertanggung jawab untuk mencampur kolom data dengan mengubah hubungan antar byte dalam kolom.

```
-- MixColumns transformation
    function mix_columns(state_in: state_array) return
state_array is
```

```
        variable result : state_array;
    begin
        for col in 0 to 3 loop
            result(0,col) := gf_mult(X"02", state_in(0,col))
xor
                             gf_mult(X"03", state_in(1,col))
xor
                             state_in(2,col) xor
                             state_in(3,col);

            result(1,col) := state_in(0,col) xor
                             gf_mult(X"02", state_in(1,col))
xor
                             gf_mult(X"03", state_in(2,col))
xor
                             state_in(3,col);

            result(2,col) := state_in(0,col) xor
                             state_in(1,col) xor
                             gf_mult(X"02", state_in(2,col))
xor
                             gf_mult(X"03", state_in(3,col));

            result(3,col) := gf_mult(X"03", state_in(0,col))
xor
                             state_in(1,col) xor
                             state_in(2,col) xor
                             gf_mult(X"02", state_in(3,col));
        end loop;
        return result;
    end function;
```

Setelah deklarasi semua fungsi terdapat proses key expansion dalam algoritma AES. Pada awalnya, kunci yang diberikan akan dibagi menjadi empat word (32-bit) untuk membentuk ***w***. Proses ini dimulai dengan mengisi array w dengan key asli, kemudian untuk setiap iterasi, sebuah kata baru dihasilkan dengan mengoperasikan rotasi (RotWord), substitusi byte (SubWord) menggunakan S-Box, serta operasi XOR menggunakan Rcon. Setelah setiap empat kata, nilai-nilai tersebut digabungkan untuk membentuk kunci ronde yang disimpan dalam ***round_keys***. Proses ini dilakukan hingga semua kunci ronde diperlukan (44 word untuk AES-128). Jika proses sudah selesai, sinyal ***key_expanded*** diatur menjadi '1' yang menandakan bahwa proses ini selesai dan Key siap digunakan dalam proses enkripsi.

```
begin
    -- Key expansion process
    process(clk, rst)
        type word_array is array (0 to 43) of STD_LOGIC_VECTOR(31
downto 0);
        variable w : word_array;
        variable temp : STD_LOGIC_VECTOR(31 downto 0);
    begin
        if rst = '1' then
```

```
            key_expanded <= '0';
            round_keys <= (others => '0');
        elsif rising_edge(clk) then
            if not key_expanded then
                -- Store original key as first round key
                round_keys(1407 downto 1280) <= key;

                -- Initialize w with key
                w(0) := key(127 downto 96);
                w(1) := key(95 downto 64);
                w(2) := key(63 downto 32);
                w(3) := key(31 downto 0);

                -- Generate remaining words
                for i in 4 to 43 loop
                    temp := w(i-1);
                    if (i mod 4 = 0) then
                        -- RotWord
                        temp := temp(23 downto 0) & temp(31 downto 24);
                        -- SubWord
                        for j in 0 to 3 loop
                            temp(31-8*j downto 24-8*j) :=
                                SBOX(to_integer(unsigned(temp(31-8*j
downto 24-8*j))));
                        end loop;
                        -- XOR with Rcon
                        temp := temp xor (rcon(i/4) & X"000000");
                    end if;
                    w(i) := w(i-4) xor temp;

                    -- Pack round keys as they are generated
                    if (i mod 4 = 3) and (i > 3) then
                        round_keys(1407-128*(i/4) downto
1280-128*(i/4)) <=
                            w(i-3) & w(i-2) & w(i-1) & w(i);
                    end if;
                end loop;

                key_expanded <= '1';
            end if;
        end if;
    end process;
```

Bagian kode selanjutnya adalah proses enkripsi utama pada algoritma AES. Proses ini dimulai dengan memeriksa apakah ekspansi kunci telah selesai (jika ***key_expanded*** = '1'). Pada ronde pertama (round 0), keadaan awal (state) dimuat dengan data input yang telah di-XOR dengan kunci ronde pertama, lalu kunci untuk ronde berikutnya disiapkan. Pada ronde selanjutnya (round 1 hingga 9), serangkaian transformasi dilakukan, yaitu SubBytes, ShiftRows, dan MixColumns, diikuti AddRoundKey yang meng-XOR-kan keadaan dengan kunci ronde yang sesuai. Pada ronde ke-10, yang merupakan ronde terakhir, hanya dilakukan

SubBytes dan ShiftRows, diikuti dengan AddRoundKey terakhir, dan sinyal ***done_i*** diset menjadi '1' untuk menunjukkan bahwa proses enkripsi selesai.

```
-- Main encryption process
    main_proc: process(clk, rst)
        variable next_state : state_array;
    begin
        if rst = '1' then
            round <= 0;
            done_i <= '0';
            state <= (others => (others => (others => '0')));
        elsif rising_edge(clk) then
            if key_expanded = '1' then  -- Wait for key expansion
                case round is
                    when 0 =>
                        -- Initial state loading and AddRoundKey
                        for col in 0 to 3 loop
                            for row in 0 to 3 loop
                                state(row,col) <=
data_in(127-8*(4*col+row) downto 120-8*(4*col+row)) xor

key(127-8*(4*col+row) downto 120-8*(4*col+row));
                            end loop;
                        end loop;

                        -- Prepare key for next round
                        current_round_key <= round_keys(1279 downto
1152);
                        round <= round + 1;
                        initial_state_loaded <= '1';

                    when 1 to 9 =>
                        -- Apply transformations in sequence
                        next_state := sub_bytes(state);
                        next_state := shift_rows(next_state);
                        next_state := mix_columns(next_state);

                        -- AddRoundKey with proper key selection
                        for col in 0 to 3 loop
                            for row in 0 to 3 loop
                                state(row,col) <= next_state(row,col)
xor

current_round_key(127-8*(4*col+row) downto 120-8*(4*col+row));
                            end loop;
                        end loop;

                        -- Update round key untuk round berikutnya
                        current_round_key <= round_keys(1279-128*round
downto 1152-128*round);
                        round <= round + 1;

                    when 10 =>
                        if not done_i then
                            next_state := sub_bytes(state);
                            next_state := shift_rows(next_state);
```

```
                            -- Final AddRoundKey
                            for i in 0 to 3 loop
                                for j in 0 to 3 loop
                                    state(j,i) <= next_state(j,i) xor
current_round_key(127-8*(4*i+j) downto 120-8*(4*i+j));
                                end loop;
                            end loop;
                            done_i <= '1';
                        end if;

                    when others => null;
                end case;
            end if;
        end if;
    end process;
```

Dibawah proses enkripsi utama, terdapat proses debug yang berfungsi untuk memperbarui sinyal debug pada setiap siklus clock. Proses ini akan menampilkan keadaan internal ***debug_state_out***, kunci ronde saat ini ***debug_round_key_out***, dan hasil transformasi seperti SubBytes, ShiftRows, dan MixColumns selama eksekusi.

```
-- Debug process
    debug_proc: process(clk)
    begin
        if rising_edge(clk) then
            -- Update debug signals
            debug_state_out <= state;
            debug_round_key_out <= current_round_key when round > 0
else key;

            -- Show transformations only after initial state loaded
            if initial_state_loaded = '1' or round > 0 then
                debug_sub_bytes_out <= sub_bytes(state);
                debug_shift_rows_out <= shift_rows(sub_bytes(state));
                debug_mix_cols_out <=
mix_columns(shift_rows(sub_bytes(state)));
            end if;
        end if;
    end process;
```

Setelah enkripsi selesai, sinyal done diberi nilai '1' untuk menandakan bahwa proses enkripsi telah selesai. Di akhir program terdapat proses output di mana keadaan internal (state) diubah menjadi format yang sesuai untuk output, yang terdiri dari data 128-bit dan diberikan pada sinyal ***data_out***.

```
-- Output assignment
    done <= done_i;

    -- Convert state to output
```

```
    output_process: process(state)
        variable temp : STD_LOGIC_VECTOR(127 downto 0);
    begin
        for col in 0 to 3 loop
            for row in 0 to 3 loop
                temp(127-8*(4*col+row) downto 120-8*(4*col+row)) :=
state(row,col);
            end loop;
        end loop;
        data_out <= temp;
    end process;
```

Demikian keseluruhan dari file **AES_Encryption.vhd**. Dalam mengerjakan program ini, tentunya kita harus menguji kebenaran dari setiap bagian dan proses enkripsi untuk mengetahui apakah program ini akan mengoutput hasil (ciphertext) yang sesuai dengan input data dan key yang diberikan. Maka, kami juga membuat program **AES_Encryption_Testbench.vhd** untuk pengujian program.

Program Testbench ini menggunakan kemampuan *file handling* VHDL untuk membaca input data dan key yang diberikan sebuah User paada sebuah file teks serta menulis hasil proses program ke file teks lainnya. Untuk ini, file input, yang bernama **aes_test_vectors.txt**, dapat diisi User dengan data yang mencakup plaintext, key, dan hasil yang diharapkan. Ketiga input ini akan dikonversi menjadi sinyal ***std_logic_vector*** untuk diberikan ke komponen enkripsi. Setelah proses enkripsi, hasil keluaran dari ***data_out*** dibandingkan dengan hasil yang diharapkan, dan status pengujian (PASS/FAIL) dicatat. Semua informasi terkait proses enkripsi, seperti status debug, yang mencakup ***state, sub_bytes, shift_rows, mix_cols, dan round_key*** untuk setiap ronde ditulis ke dalam file output **aes_test_results.txt**.

Berikut adalah penjelasan singkat mengenai bagian - bagian dari program **AES_Encryption_Testbench.vhd**:

```
-- Component declaration
    component AES_Encryption
        Port (
            clk : in STD_LOGIC;
            rst : in STD_LOGIC;
            data_in : in STD_LOGIC_VECTOR(127 downto 0);
            key : in STD_LOGIC_VECTOR(127 downto 0);
            data_out : out STD_LOGIC_VECTOR(127 downto 0);
            done : out STD_LOGIC;
            debug_state_out : out state_array;
            debug_sub_bytes_out : out state_array;
            debug_shift_rows_out : out state_array;
            debug_mix_cols_out : out state_array;
```

```
            debug_round_key_out : out STD_LOGIC_VECTOR(127 downto 0)
        );
    end component;
-- Test signals
    signal clk : STD_LOGIC := '0';
    signal rst : STD_LOGIC := '1';
    signal data_in : STD_LOGIC_VECTOR(127 downto 0);
    signal key : STD_LOGIC_VECTOR(127 downto 0);
    signal data_out : STD_LOGIC_VECTOR(127 downto 0);
    signal done : STD_LOGIC;

    -- Add debug signals
    signal debug_state : state_array;
    signal debug_sub_bytes : state_array;
    signal debug_shift_rows : state_array;
    signal debug_mix_cols : state_array;
    signal debug_round_key : STD_LOGIC_VECTOR(127 downto 0);

    -- Clock period
    constant clk_period : time := 10 ns;

    -- State machine signals
    type test_state_type is (INIT, RUNNING, TEST_DONE);
    signal test_state : test_state_type := INIT;
    signal current_round : integer range 0 to 10 := 0;
```

Bagian Testbench di atas mendeklarasikan komponen ***AES_Encryption*** yang akan diuji, beserta port-port yang diperlukan seperti sinyal clock, reset, input data, input key, output data, serta sinyal - sinyal debugging untuk memantau status internal. Selanjutnya, testbench mendefinisikan sinyal-sinyal tersebut selama pengujian. Lalu, konstanta ***clk_period*** menetapkan periode clock, dan sinyal-sinyal status mesin (***test_state*** dan ***current_round***) digunakan untuk mengontrol alur testbench beserta suatu state machine yang mengecek apakah program masih inisialisasi (***INIT***), masih berjalan (***RUNNING***) atau telah selesai (***TEST_DONE***).

```
-- Single process for state management and stimulus
    stim_proc: process
        file input_file : text;
        file output_file : text;
        variable input_line : line;
        variable output_line : line;
        variable plaintext_str : string(1 to 32);
        variable key_str : string(1 to 32);
        variable expected_str : string(1 to 32);
        variable space : character;
    begin
        -- Open files
        file_open(input_file, "aes_test_vectors.txt", read_mode);
        file_open(output_file, "aes_test_results.txt", write_mode);

        -- Read test vector
```

```
        readline(input_file, input_line);
        read(input_line, plaintext_str);
        read(input_line, space);
        read(input_line, key_str);
        read(input_line, space);
        read(input_line, expected_str);

        -- Write test setup info
        write(output_line, string'("=== AES Encryption Test ==="));
        writeline(output_file, output_line);
        write(output_line, string'("Plaintext: ") & plaintext_str);
        writeline(output_file, output_line);
        write(output_line, string'("Key       : ") & key_str);
        writeline(output_file, output_line);
        write(output_line, string'("Expected  : ") & expected_str);
        writeline(output_file, output_line);
        writeline(output_file, output_line);
```

Kode selanjutnya mendefinisikan proses ***stim_proc***, yang bertanggung jawab untuk membaca data yang diuji dari file input dan menulis hasil uji ke file output. Pada bagian awal, file input (**aes_test_vectors.txt**) dan file output (**aes_test_results.txt**) dibuka dengan mode baca dan tulis, masing-masing. Kemudian, baris pertama dari file input dibaca untuk mendapatkan nilai plaintext, key, dan nilai expected hasil enkripsi. Setelah membaca nilai-nilai tersebut, informasi setup pengujian, termasuk plaintext, kunci, dan nilai ekspektasi, ditulis ke dalam file output.

```
-- Monitor encryption process
        while test_state /= TEST_DONE loop
            -- Debug headers
            write(output_line,
string'("----------------------------------------"));
            writeline(output_file, output_line);
            write(output_line, string'("Round ") &
integer'image(current_round));
            writeline(output_file, output_line);

            -- Debug state values
            write(output_line, string'("Current state: "));
            writeline(output_file, output_line);
            write(output_line, string'("  State     : ") &
state_to_hex(debug_state));
            writeline(output_file, output_line);
            write(output_line, string'("  SubBytes  : ") &
state_to_hex(debug_sub_bytes));
            writeline(output_file, output_line);
            write(output_line, string'("  ShiftRows : ") &
state_to_hex(debug_shift_rows));
            writeline(output_file, output_line);
            write(output_line, string'("  MixCols   : ") &
state_to_hex(debug_mix_cols));
            writeline(output_file, output_line);
```

```
            write(output_line, string'("  RoundKey  : ") &
slv_to_hex(debug_round_key));
            writeline(output_file, output_line);
            writeline(output_file, output_line);

            -- Update state and wait
            if done = '1' then
                test_state <= TEST_DONE;
                write(output_line, string'("========== Encryption
Complete =========="));
                writeline(output_file, output_line);
            elsif current_round < 10 then
                current_round <= current_round + 1;
            end if;

            wait for clk_period;
        end loop;
```

Bagian kode ini berfungsi untuk memantau dan mencatat proses enkripsi AES pada setiap round dalam bentuk debug output yang ditulis ke dalam file output. Selama status ***test_state*** tidak mencapai ***TEST_DONE***, kode ini mencatat berbagai informasi debugging seperti status saat ini (***debug_state***), hasil transformasi SubBytes, ShiftRows, MixCols, dan kunci ronde (***debug_round_key***) pada setiap ronde enkripsi yang sedang berlangsung.

```
-- Write final results
        write(output_line, string'("Final Results:"));
        writeline(output_file, output_line);
        write(output_line, string'("Output   : ") &
slv_to_hex(data_out));
        writeline(output_file, output_line);
        write(output_line, string'("Expected : ") & expected_str);
        writeline(output_file, output_line);

        if data_out = hex_to_slv(expected_str) then
            write(output_line, string'("Status: PASS"));
        else
            write(output_line, string'("Status: FAIL"));
        end if;
        writeline(output_file, output_line);

        -- Cleanup and end
        file_close(input_file);
        file_close(output_file);
        report "Simulation complete";
        std.env.stop(0);
        wait;
    end process;
```

Pada akhir program testbench, bagian ini bertanggung jawab untuk menulis hasil akhir dari simulasi enkripsi AES ke dalam file output dan memverifikasi apakah hasil

enkripsi sesuai dengan yang diharapkan. Setelah proses enkripsi selesai, kode ini mencatat hasil enkripsi (***data_out***) dan membandingkannya dengan nilai yang diharapkan (***expected_str***) yang dibaca dari file input sebelumnya. Hasil output dan nilai yang diharapkan kemudian ditulis dalam format hexadecimal ke file output. Setelah itu, kode membandingkan hasil enkripsi dengan nilai yang diharapkan. Jika hasilnya cocok, maka statusnya ditandai sebagai "PASS"; jika tidak, statusnya ditandai sebagai "FAIL". Informasi ini juga dituliskan ke dalam file output. Setelah menulis hasil verifikasi, kode menutup file input dan output, serta memberikan laporan bahwa simulasi telah selesai dengan perintah report "Simulation complete".

Demikian penjelasan pada bagian pertama, yaitu enkripsi, dari proyek AES 128-bit kami. Pada bagian kedua proyek ini, kami berusaha untuk membuat kebalikan dari program enkripsi, atau lebih tepatnya disebut dengan dekripsi. Bagian ini kami buat dalam dua file, yaitu **AES_Decryption.vhd** dan **AES_Decryption_Testbench.vhd**.

File **AES_Decryption.vhd** adalah implementasi dari proses dekripsi algoritma AES 128-bit. Pada dasarnya, file ini berfungsi untuk membalikkan proses yang telah dilakukan oleh enkripsi, menggunakan kunci yang sama. Entity yang terdapat dalam file ini menerima input berupa data yang telah dienkripsi (***data_in***) dan kunci dekripsi (***key***), kemudian menghasilkan output berupa data yang telah didekripsi (***data_out***).

Seperti halnya pada bagian enkripsi, dekripsi juga melibatkan serangkaian operasi dasar pada data yang inputnya, yaitu Inverse SubBytes, Inverse ShiftRows, Inverse MixColumns, dan AddRoundKey. Tabel SBOX yang digunakan pada enkripsi, digantikan oleh INV_SBOX (Inverse S-Box) pada dekripsi, yang digunakan untuk inversi substitusi pada langkah Inverse SubBytes. Selain itu, ada operasi - operasi lainnya juga dibalik supaya berfungsi untuk membalikkan efek enkripsi, seperti fungsi Inverse ShiftRows dan Inverse MixColumns.

File **AES_Decryption_Testbench.vhd** yang digunakan untuk menguji kebenaran dekripsi, tidak jauh berbeda dengan file **AES_Encryption_Testbench.vhd**. Perbedaannya hnaya terleta pada deklarasi komponen, di mana menggunakan testbench ini menggantikan komponen **AES_Encryption** menjadi **AES_Decryption**. Di dalam file testbench ini, sinyal yang digunakan sama dengan yang ada pada test bench enkripsi. Proses yang dilakukan juga serupa, yaitu membaca test vector, menerapkan input seperti ciphertext dan kunci dekripsi, serta memonitor output dari proses dekripsi.

CHAPTER 3

TESTING AND ANALYSIS

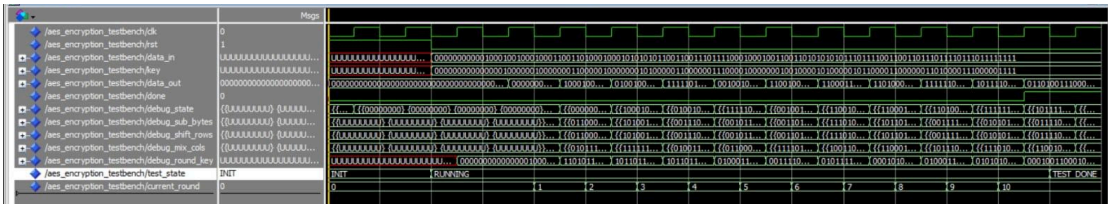
3.1 TESTING

Proses pengujian dilakukan untuk memastikan bahwa proses enkripsi dan dekripsi berjalan sesuai prosedur serta memenuhi persyaratan fungsi dan kinerja yang diinginkan. Pada proyek ini, pengujian dilakukan menggunakan tiga metode utama, yaitu wave test (ModelSim), synthesize test (Quartus), dan pengujian berbasis text file.

Metode wave test digunakan untuk menguji keakuratan sinyal pada setiap tahapan proses enkripsi dan dekripsi, seperti AddRoundKey, SubBytes, ShiftRows, dan MixColumns. Synthesize test digunakan untuk menguji keseluruhan proses enkripsi dan dekripsi dengan berbagai input key dan plaintext, serta memastikan bahwa ciphertext yang dihasilkan dapat didekripsi kembali menjadi plaintext asli secara benar. Pengujian menggunakan text file digunakan untuk membaca input plaintext pada proses enkripsi dan input ciphertext pada proses dekripsi, serta memeriksa hasil dari kedua proses tersebut untuk memastikan bahwa data yang dihasilkan sudah sesuai.

3.2 RESULT

- **Enkripsi**



File Input:

```
00112233445566778899aabbccddeeff 000102030405060708090a0b0c0d0e0f
69c4e0d86a7b0430d8cdb78070b4c55a
```

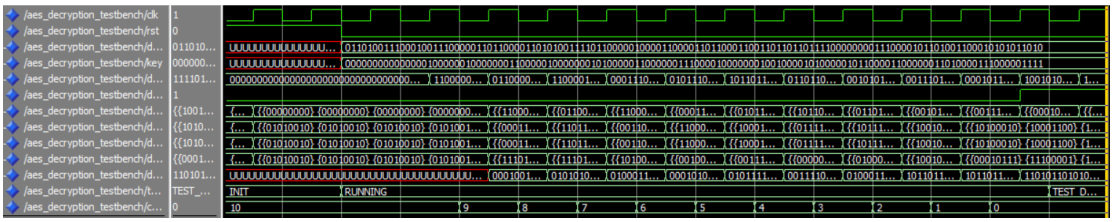
File Output:

```
=== AES Encryption Test ===
Plaintext: 00112233445566778899aabbccddeeff
Key      : 000102030405060708090a0b0c0d0e0f
Expected : 69c4e0d86a7b0430d8cdb78070b4c55a
```

```
----------------------------------------
Round 0
Current state:
  State     : 00000000000000000000000000000000
  SubBytes  : 00000000000000000000000000000000
  ShiftRows : 00000000000000000000000000000000
  MixCols   : 00000000000000000000000000000000
  RoundKey  : 000102030405060708090a0b0c0d0e0f

----------------------------------------
Round 1
Current state:
  State     : 004080c0105090d02060a0e03070b0f0
  SubBytes  : 6309cdbaca536070b7d0e0e10451e78c
  ShiftRows : 6309cdba536070cae0e1b7d08c0451e7
  MixCols   : 5f57f71d72f5beb964bc3bf91592291a
  RoundKey  : d6aa74fdd2af72fadaa678f1d6ab76fe

----------------------------------------
Round 2
Current state:
  State     : 89852dcbd85a181210ce438fe868d8e4
  SubBytes  : a797d81f61beadc9ca8b1a739b456169
  ShiftRows : a797d81fbeadc9611a73ca8b699b4561
  MixCols   : ff31647787d8513a966a51d08451fa09
  RoundKey  : b692cf0b643dbdf1be9bc5006830b3fe

----------------------------------------
Round 3
Current state:
  State     : 4955da1f15e5ca0a59d794638fa0faf7
  SubBytes  : 3bfc57c059d97467cb0e22fb73e02d68
  ShiftRows : 3bfc57c0d974675922fbcb0e6873e02d
  MixCols   : 4cf72c539c713f4d1ef086f266768e56
  RoundKey  : b6ff744ed2c2c9bf6c590cbf0469bf41

----------------------------------------
Round 4
Current state:
  State     : fa25405763b366246a398a4d28c93117
  SubBytes  : 2d3f095bfb6d333602127ee334ddc7f0
  ShiftRows : 2d3f095b6d3336fb7ee30212f034ddc7
  MixCols   : 63fc97758553be47b78d47d69ff98e91
  RoundKey  : 47f7f7bc95353e03f96c32bcfd058dfd

----------------------------------------
Round 5
Current state:
  State     : 24696e887266d24240b3755b23fa326c
  SubBytes  : 36f99fc44033b52c096d9d39262d2350
  ShiftRows : 36f99fc433b52c409d39096d50262d23
  MixCols   : f432751dbce5f1d0d454d63b54d0c53c
  RoundKey  : 3caaa3e8a99f9deb50f3af57adf622aa

----------------------------------------
Round 6
Current state:
```

```
  State     : c89b25b0167a022677c97919bc3b9296
  SubBytes  : e8143fe747da77f7f5ddb6d465e24f90
  ShiftRows : e8143fe7da77f747b6d4f5dd9065e24f
  MixCols   : 98006b8e16f82c5aee7f04d074559c36
  RoundKey  : 5e390f7df7a69296a7553dc10aa31f6b

----------------------------------------
Round 7
Current state:
  State     : c6f7cc842f5e79f9e1ed39cf09c35d5d
  SubBytes  : b4684b5f1558b699f855128a012e4c4c
  ShiftRows : b4684b5f58b69915128af8554c012e4c
  MixCols   : c59af0987e9b5fc61cd24b341586e039
  RoundKey  : 14f9701ae35fe28c440adf4d4ea9c026

----------------------------------------
Round 8
Current state:
  State     : d179b4d687c4556f6c3094f40f0aad1f
  SubBytes  : 3eb68df6171cfca8500422bf766795c0
  ShiftRows : 3eb68df61cfca81722bf5004c0766795
  MixCols   : baa1d55fa0f951413db52c4de76eba23
  RoundKey  : 47438735a41c65b9e016baf4aebf7ad2

----------------------------------------
Round 9
Current state:
  State     : fd0535f1e3e547febad09637d2d74ef1
  SubBytes  : 546b96a111d9a0bbf470909ab50e2fa1
  ShiftRows : 546b96a1d9a0bb11909af470a1b50e2f
  MixCols   : e9021b35f730f23c4e20cc21ecf6f2c7
  RoundKey  : 549932d1f08557681093ed9cbe2c974e

----------------------------------------
Round 10
Current state:
  State     : bdf20b8b6eb561107c7721b63d9e6e89
  SubBytes  : 7a892b3d9fd5efca10f5fd4e270b9fa7
  ShiftRows : 7a892b3dd5efca9ffd4e10f5a7270b9f
  MixCols   : ca4a08aa70b99f83bc93dce9f36fb108
  RoundKey  : 13111d7fe3944a17f307a78b4d2b30c5

========== Encryption Complete ==========
Final Results:
Output   : 69c4e0d86a7b0430d8cdb78070b4c55a
Expected : 69c4e0d86a7b0430d8cdb78070b4c55a
Status: PASS
```

- **Dekripsi**



File Input:

```
69c4e0d86a7b0430d8cdb78070b4c55a 000102030405060708090a0b0c0d0e0f
00112233445566778899aabbccddeeff
```

File Output:

```
=== AES Decryption Test ===
Ciphertext: 69c4e0d86a7b0430d8cdb78070b4c55a
Key       : 000102030405060708090a0b0c0d0e0f
Expected  : 00112233445566778899aabbccddeeff

----------------------------------------
Round 10
Current state:
  State     : 00000000000000000000000000000000
  SubBytes  : 52525252525252525252525252525252
  ShiftRows : 52525252525252525252525252525252
  MixCols   : 52525252525252525252525252525252
  RoundKey  : 00000000000000000000000000000000

----------------------------------------
Round 9
Current state:
  State     : c08bf90cf671eaed85aa6ceb7be0b9c6
  SubBytes  : 1fce6981d62cbb536762b83c03a0dbc7
  ShiftRows : 1fce698153d62cbbb83c6762a0dbc703
  MixCols   : ef5d57e1bd6bb771e04a88b4e6838d7f
  RoundKey  : 13111d7fe3944a17f307a78b4d2b30c5

----------------------------------------
Round 8
Current state:
  State     : 611597bc3a539a0fee81fd418df064db
  SubBytes  : d82f8578a25037fb999121f8b4178c9f
  ShiftRows : d82f8578fba2503721f89991178c9fb4
  MixCols   : ea2f557874ded94d3911f445b219ab1a
  RoundKey  : 549932d1f08557681093ed9cbe2c974e

----------------------------------------
Round 7
Current state:
  State     : c3e9dde58661cc1ed9d44ac587013156
  SubBytes  : 33ebc92adcd827e9e5195c07ea092eb9
  ShiftRows : 33ebc92ae9dcd8275c07e519092eb9ea
  MixCols   : a0f648b3ae7a2e0dca26fe674bb4d527
  RoundKey  : 47438735a41c65b9e016baf4aebf7ad2
```

```
----------------------------------------
Round 6
Current state:
  State     : 1c1906a6dc37844bd4038559b09957f1
  SubBytes  : c48ea5c593b24fcc19d56715fcf9da2b
  ShiftRows : c48ea5c5cc93b24f671519d5f9da2bfc
  MixCols   : 24f65b461fa3bb56b0d2ad0b1d5568b8
  RoundKey  : 14f9701ae35fe28c440adf4d4ea9c026

----------------------------------------
Round 5
Current state:
  State     : 5da5fa1d4a9c5cbaf7e9f34ecc83c7ec
  SubBytes  : 8d2914de5c1ca7c026eb7eb627413183
  ShiftRows : 8d2914dec05c1ca77eb626eb41318327
  MixCols   : 39aa7e925a1f90632eca0dd63f8d4e92
  RoundKey  : 5e390f7df7a69296a7553dc10aa31f6b

----------------------------------------
Round 4
Current state:
  State     : b6808c68b5754868c1b668cd468a3768
  SubBytes  : 793af0f7d23fd4f7dd79f78098cfb2f7
  ShiftRows : 793af0f7f7d23fd4f780dd79cfb2f798
  MixCols   : 017f4d3766bd6136f86ab126297278e5
  RoundKey  : 3caaa3e8a99f9deb50f3af57adf622aa

----------------------------------------
Round 3
Current state:
  State     : 6cda953f9838d3e510f9a858b40c0667
  SubBytes  : b87aad25e276a92a7c696f5ec681a50a
  ShiftRows : b87aad252ae276a96f5e7c6981a50ac6
  MixCols   : 4797db2d8bd03bb568754b27d851069c
  RoundKey  : 47f7f7bc95353e03f96c32bcfd058dfd

----------------------------------------
Round 2
Current state:
  State     : 2a5fa6fdaeab0cd4f60ec140ceb20661
  SubBytes  : 9584c521be0e8119d6d7dd72ec3ea5d8
  ShiftRows : 9584c52119be0e81dd72d6d73ea5d8ec
  MixCols   : 97c1b3a02c9567ad01f6317bd54f20ed
  RoundKey  : b6ff744ed2c2c9bf6c590cbf0469bf41

----------------------------------------
Round 1
Current state:
  State     : 3a64060d78cb2bfbe813c0962ebac310
  SubBytes  : a28ca5f3c1590b63c8821f35c3c0337c
  ShiftRows : a28ca5f363c1590b1f35c882c0337cc3
  MixCols   : 17e1d7c1cff2ca0996ce2ad950967fa8
  RoundKey  : b692cf0b643dbdf1be9bc5006830b3fe

----------------------------------------
Round 0
Current state:
  State     : 17e9c7d9cdf8d81392c23ec5569869b7
```

```
  SubBytes  : a28ca5f3c1590b63c8821f35c3c0337c
  ShiftRows : a28ca5f363c1590b1f35c882c0337cc3
  MixCols   : 17e1d7c1cff2ca0996ce2ad950967fa8
  RoundKey  : d6aa74fdd2af72fadaa678f1d6ab76fe

========== Decryption Complete ==========
Final Results:
Output   : f4b4b4015cb6a8195b9c808a22de55b0
Expected : 00112233445566778899aabbccddeeff
Status: FAIL
```

3.3 ANALYSIS

Pada proses enkripsi AES, program sudah berhasil berjalan dengan baik, di mana setiap langkah telah diimplementasikan sesuai dengan prosedur. Keberhasilan enkripsi ini dapat dilihat pada hasil uji coba pada file output yang menunjukkan hasil enkripsi yang dihasilkan telah sesuai dengan yang diharapkan. Hal ini menunjukkan bahwa semua komponen penting AES, seperti proses substitusi (SubBytes), penyebaran data (ShiftRows, MixColumns), dan penggabungan dengan menggunakan key (AddRoundKey) telah diimplementasikan dengan baik dan benar. Hasil akhir dari proses ini adalah ciphertext yang sudah terenkripsi dan dapat didekripsi kembali ke plaintext asli jika key yang dimasukkan benar.

Pada proses dekripsi AES, terdapat beberapa masalah dalam rangkaian yang menyebabkan proses belum berjalan dengan baik. Salah satu permasalahan utama yang terjadi adalah ketidaksesuaian pada round key. Pada proses dekripsi AES, round key seharusnya digunakan dalam urutan terbalik dari round key yang digunakan pada proses enkripsi. Meskipun urutan round key telah dibalik dan diatur dengan benar, round key yang digunakan pada round pertama dekripsi (yang seharusnya merupakan round key terakhir pada enkripsi) masih belum sesuai, yaitu bernilai “00000000000000000000000000000000”. Round key yang seharusnya digunakan di round pertama justru muncul pada round kedua dalam proses dekripsi.

Kesalahan pada round key ini berdampak langsung pada langkah AddRoundKey, yang merupakan langkah awal setiap round dekripsi. AddRoundKey menggabungkan state dengan round key menggunakan operasi XOR, sehingga ketidaksesuaian pada langkah ini akan mempengaruhi keseluruhan proses dekripsi. Akibatnya, hasil dari langkah Inv ShiftRows juga menjadi tidak sesuai. Kesalahan ini akan berdampak pada hasil substitusi di langkah Inv SubBytes, yang kemudian menyebabkan kesalahan berlanjut pada langkah Inv

MixColumns. Dengan begitu, kesalahan penggunaan round key di awal memiliki efek berantai terhadap seluruh proses dekripsi AES.

CHAPTER 4

CONCLUSION

Proyek AES Encryption Hardware Accelerator bertujuan untuk mengembangkan akselerator enkripsi dan dekripsi berbasis perangkat keras menggunakan algoritma Advanced Encryption Standard (AES) pada FPGA dengan VHDL. Implementasi enkripsi AES 128 berhasil dilakukan dengan memanfaatkan teknik paralelisme dan pipelining untuk meningkatkan efisiensi dibandingkan implementasi perangkat lunak. Pengujian menunjukkan bahwa hasil enkripsi sesuai dengan ekspektasi berdasarkan vektor uji yang digunakan. Namun, proses dekripsi mengalami kendala, terutama pada penggunaan round key yang seharusnya diterapkan dalam urutan terbalik dari proses enkripsi. Ketidaksesuaian ini menyebabkan seluruh langkah dekripsi tidak berjalan dengan benar, sehingga hasil akhirnya tidak sesuai dengan plaintext asli. Meskipun demikian, debugging dilakukan dengan baik menggunakan sinyal status internal untuk memantau proses enkripsi dan dekripsi pada setiap ronde, sehingga sumber masalah pada dekripsi dapat teridentifikasi. Proyek ini berhasil memberikan pemahaman mendalam tentang penerapan algoritma kriptografi pada perangkat keras dan memiliki potensi aplikasi dalam pengolahan data cepat serta aman, seperti transaksi elektronik dan komunikasi terenkripsi.

REFERENCES

[1] M. Dworkin, "*Recommendation for Block Cipher Modes of Operation: Methods and Techniques*," NIST Special Publication 800-38A, 2001. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf

[2] National Institute of Standards and Technology (NIST), "*Advanced Encryption Standard (AES),*" FIPS Publication 197, U.S. Department of Commerce, Nov. 2001. [Online]. Available: https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf

[3] S. Frankel, R. Glenn, S. Kelly, "*The AES-CBC Cipher Algorithm and Its Use with IPsec*," IETF RFC 3602, Sept. 2003. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc3602

[4] National Institute of Standards and Technology (NIST), "*Cryptographic Algorithm Validation Program*," [Online]. Available: https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/block-ciphers

[5] Assistant Digital Laboratory UI, "*Module 4: Testbench*," Universitas Indonesia, [Online]. Available: https://learn.digilabdte.com/books/digital-system-design/chapter/module-4-testbench

[6] Assistant Digital Laboratory UI, "*Module 6: Looping*," Universitas Indonesia, [Online]. Available: https://learn.digilabdte.com/books/digital-system-design/chapter/module-6-looping

[7] Assistant Digital Laboratory UI, "*Module 7: Procedure, Function, and Impure Function*," Universitas Indonesia, [Online]. Available: https://learn.digilabdte.com/books/digital-system-design/chapter/module-7-procedure-function-and-impure-function

[8] Assistant Digital Laboratory UI, "*Module 8 : Finite State Machine*," Universitas Indonesia, [Online]. Available: https://learn.digilabdte.com/books/digital-system-design/chapter/module-8-finite-state-machine

[9] Assistant Digital Laboratory UI, "*Module 7: Module 8 : Finite State Machine*," Universitas Indonesia, [Online]. Available: https://learn.digilabdte.com/books/digital-system-design/chapter/module-9-microprogramming

APPENDICES

Appendix A: Project Schematic

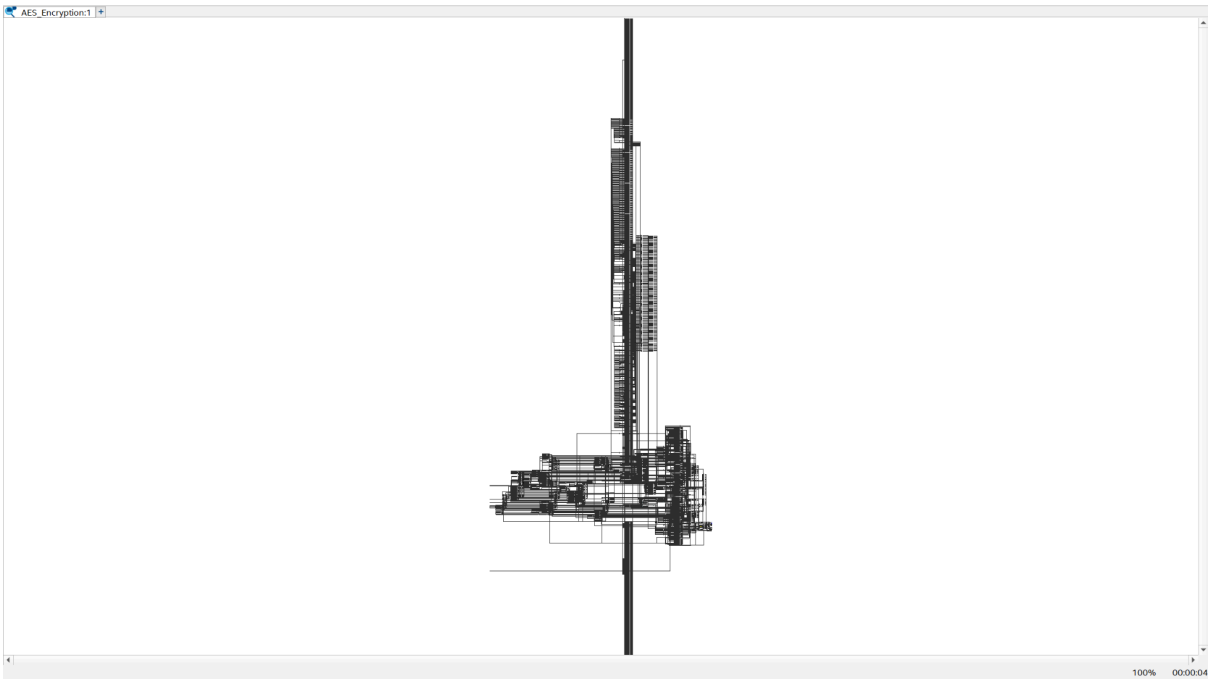
Untuk hasil sintesis program AES_Encryption.vhd menggunakan Quartus Prime, kami mendapatkan error seperti berikut:

- *Error (11802): Can't fit design in device. Modify your design to reduce resources, or choose a larger device. The Intel FPGA Knowledge Database contains many articles with specific details on how to resolve this error. Visit the Knowledge Database at https://www.altera.com/support/support-resources/knowledge-base/search.html and search for this specific error message number.*

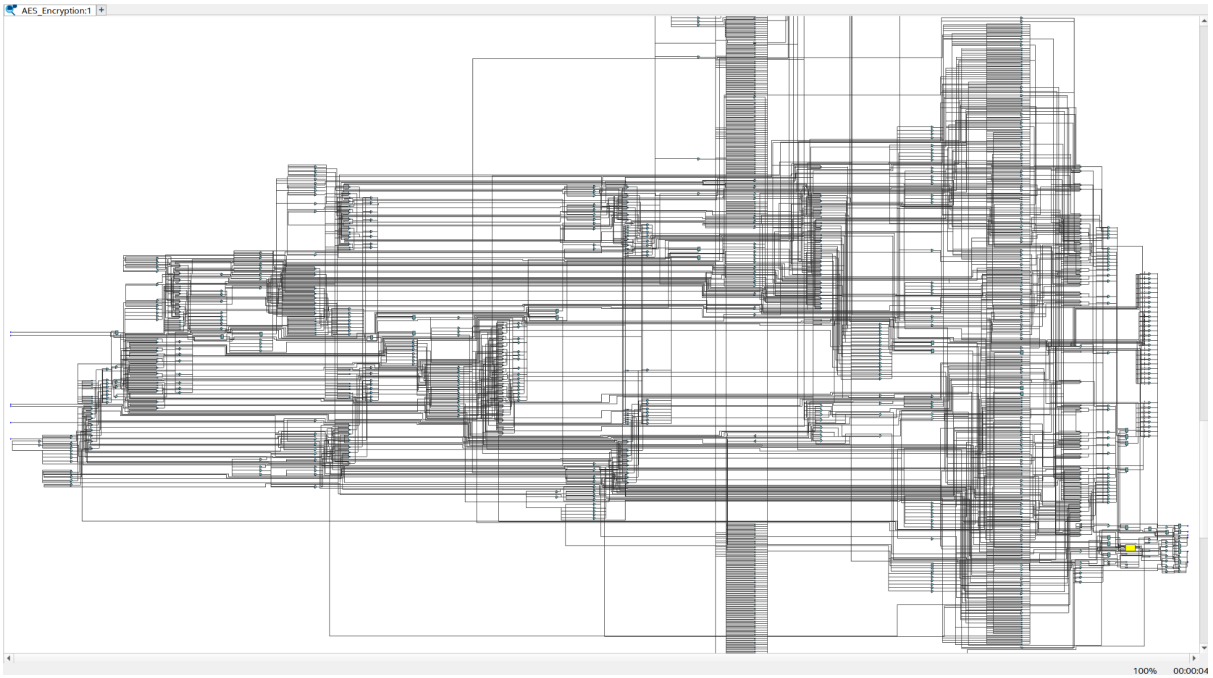
Bahkan, walaupun dengan menghapus semua debug ports, error pada Quartus Prime masih muncul seperti berikut:

- *Error (169281): There are 258 IO input pads in the design, but only 256 IO input pad locations available on the device.*

Oleh karena itu, kami mengubah kode program supaya mengambil data input dan mengeluarkan data output dalam bentuk *chunks,* sehingga Quartus Prime dapat mensintesis program. Perubahan kode yang digunakan untuk sintesis ini terdapat pada folder **AES_Encryption for Synthesis**. Berikut adalah hasil sintesis dari perubahan program tersebut:



Hasil zoom in:



Appendix B: Documentation