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

Dual-Lock Logic with Caesar and Hill Cipher

GROUP PA01

EUGENIA HUWAIDA IMTINAN	**(2406421384)**
KAYLA JOANNA IRSY KUSUMAH	**(2406487014)**
NADIA IZZATI	**(2406487033)**
SAFINA AMARINA	**(2406415665)**

PENDAHULUAN

Puji dan syukur atas Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya sehingga laporan proyek akhir Perancangan Sistem Digital yang berjudul **"Dual-Lock Logic with Caesar and Hill Cipher"** dapat diselesaikan dengan baik. Ucapan terima kasih juga kami sampaikan kepada para asisten laboratorium, serta teman-teman yang telah berkontribusi dalam proses pengerjaan laporan proyek akhir ini.

Laporan ini disusun untuk melengkapi proyek akhir yang merupakan pemenuhan dari Proyek Akhir Praktikum Perancangan Sistem Digital Tahun Ajaran 2024/2025. Laporan ini membahas tentang detail dari proyek yang telah kami buat. Laporan ini meliputi latar belakang, deskripsi, hasil dan analisis dari proyek yang telah kami buat.

Adapun karena keterbatasan pengetahuan maupun pengalaman kami, kami menyadari masih terdapat kekurangan dalam pengerjaan dan penyusunan laporan proyek akhir ini yang perlu diperbaiki. Oleh karena itu, kritik dan saran sangat kami harapkan sehingga dapat dijadikan bahan evaluasi bagi kami kedepannya. Kami juga memohon maaf apabila ada kesalahan dan kekurangan dalam penyusunan laporan ini

Depok, December 02, 2025

Group PA01

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Di era informasi ini, keamanan digital adalah aspek yang sangat penting karena semakin pesatnya teknologi sistem enkripsi yang efisien dan efektif akan sangat dibutuhkan. Enkripsi ini berfungsi untuk melindungi informasi penting yang kita miliki dari akses yang tidak berwenang, agar hanya orang terpilih yang dapat melihat informasi tersebut.

Namun, dibutuhkan beban komputasi yang tinggi jika kita memiliki sistem hardware yang terbatas untuk membuat proses enkripsi dan dekripsi ini. Maka, kita membutuhkan sistem yang mampu mengoptimalkan proses enkripsi menggunakan metode yang lebih efisien, tetapi tetap memberikan keamanan yang dibutuhkan.

Caesar Cipher dan Hill Cipher adalah salah satu algoritma yang banyak digunakan untuk pengamanan data. Caesar Cipher dan Hill Cipher adalah metode yang sering digunakan untuk pengamanan data. Caesar Cipher adalah algoritma yang mudah dipahami dan diterapkan dari Hill Cipher. Walaupun Hill Cipher lebih kompleks, dia menawarkan tingkat keamanan yang lebih tinggi karena menggunakan matrix kunci untuk enkripsinya.

Hardware yang dapat kita gunakan untuk meningkatkan kecepatan dan efisiensi enkripsi ini adalah FPGA atau Field-Programmable Gate Array yang memungkinkan desain dan implementasi algoritma enkripsi dalam bentuk yang lebih cepat dan efisien.

1.2 PROJECT DESCRIPTION

Tujuan utama dari pengembangan proyek ini adalah menciptakan sistem enkripsi dan dekripsi yang mengintegrasikan dua algoritma klasik, yaitu Caesar Cipher dan Hill Cipher. Mekanisme sistem ini dirancang untuk memproses teks melalui dua lapisan keamanan yang berbeda, tergantung pada mode operasi yang sedang dijalankan. Pada tahap awal, sistem akan mengaktifkan Caesar Cipher untuk mengenkripsi teks dengan teknik pergeseran karakter sesuai parameter yang ada. Kemudian, output dari proses ini langsung dijadikan basis input untuk tahap kedua yang menggunakan Hill Cipher. Karena Hill Cipher memanfaatkan matriks kunci yang tertanam dalam kode, metode ini menyediakan kekuatan enkripsi yang

lebih tinggi. Jadi, integrasi dari kedua metode ini akan menciptakan proteksi dua kali lipat yang jauh lebih kuat bagi data yang akan diamankan.

Mengenai alur kerjanya, sistem beroperasi dengan membaca teks sumber dari file input.txt dan kemudian mengeksekusi proses enkripsi atau dekripsi sesuai pilihan mode. Hasil akhirnya dipisahkan secara spesifik ke dalam dua file berbeda, yaitu encryptOutput.txt untuk data yang telah mengalami proses enkripsi dan decryptOutput.txt untuk hasil dekripsi atau pemulihan data. Setelah Caesar Cipher selesai bekerja, hasilnya disimpan sementara di phase.txt sebagai jembatan menuju Hill Cipher. Sebaliknya, pada dekripsi, sistem membalik urutan ini dengan menjalankan Hill Cipher terlebih dahulu untuk membuka lapisan luar, lalu Caesar Cipher sebagai langkah terakhir agar teks asli dapat dipulihkan sepenuhnya ke bentuk awalnya.

Pemanfaatan metode File I/O dalam proyek ini sangat memudahkan pengguna karena mereka dapat memasukkan data dan menerima hasil tanpa perlu berurusan langsung dengan kode program. Arsitektur sistem ini juga dibangun untuk mengelola dan memproses setiap karakter dalam teks input secara terorganisir. Proses yang terstruktur ini memastikan bahwa setiap tahapan, baik enkripsi maupun dekripsi, dieksekusi dengan presisi yang tinggi dan menjamin seluruh hasil pengolahan data tersimpan secara benar dan rapi di dalam file yang sesuai.

1.3 OBJECTIVES

Tujuan dari proyek ini:

1. Sebagai penilaian akhir dari Praktikum Perancangan Sistem Digital
2. Pengimplementasian dari Pemrograman VHDL
3. Perancangan dan Implementasi Sistem Enkripsi dan Dekripsi Berlapis dengan Kombinasi *Caesar Cipher* dan *Hill Cipher*.
4. Peningkatan Keamanan Data Melalui Implementasi Perangkat Keras Berbasis FPGA

1.4 ROLES AND RESPONSIBILITIES

Roles dan tanggung jawab yang diberikan kepada anggota kelompok adalah sebagai berikut:

Roles	Person
Membuat kode, membuat PPT, dan menyusun laporan	Eugenia Huwaida Imtinan
Membuat kode, membuat PPT, dan menyusun laporan	Kayla Joanna Irsy Kusumah
Membuat kode, menyusun Github, menyusun laporan, dan membuat PPT	Nadia Izzati
Membuat kode, membuat PPT, dan menyusun laporan	Safina Amarani

Table 1. Roles and Responsibilities

CHAPTER 2

IMPLEMENTATION

2.1 EQUIPMENT

Tools yang kami gunakan dalam mengerjakan proyek ini, yaitu:

- Visual Studio Code
- ModelSim
- Quartus
- Vivado
- Github

2.2 IMPLEMENTATION

1. Dataflow

Dalam perancangan arsitektur sistem, pengaturan *dataflow* menjadi elemen krusial untuk memastikan transfer informasi antar komponen berjalan tanpa hambatan. Modul ini difungsikan secara spesifik untuk mengelola aliran data pada proses enkripsi Caesar Cipher. Mekanisme yang diterapkan memungkinkan pemrosesan data dilakukan secara simultan, di mana kalkulasi pergeseran setiap karakter dieksekusi secara langsung tanpa harus menunggu input lainnya selesai. Pendekatan ini sangat efektif ketika diimplementasikan pada perangkat keras FPGA, karena kemampuan paralelisme yang dimilikinya mampu mengakselerasi waktu enkripsi secara signifikan, terutama pada data teks yang panjang. Maka, pemanfaatan aliran data langsung ini akan mengoptimalkan efisiensi sistem secara keseluruhan melalui peningkatan throughput dan minimalisasi latency yang terjadi.

2. Behavioral

Pendekatan deskripsi behavioral digunakan untuk merepresentasikan perilaku sistem pada tingkat abstraksi yang lebih tinggi, sehingga kita tidak perlu merinci struktur perangkat keras yang kompleks di tahap awal. Dalam lingkup proyek ini, modul tersebut difokuskan untuk memodelkan alur fungsional dari algoritma Caesar Cipher dan Hill Cipher. Pada Caesar Cipher, misalnya, logika enkripsi dibangun berdasarkan mekanisme pergeseran karakter sesuai parameter kunci yang ditentukan.

Keuntungan utama dari metode ini adalah fleksibilitas dalam perancangan logika sistem yang terlepas dari keterbatasan implementasi fisik. Maka, pendekatan ini sangat krusial untuk memvalidasi proses enkripsi secara teoritis sebelum realisasi pada perangkat keras dilakukan.

3. Testbench

Validasi fungsionalitas perancangan sistem merupakan tahap yang sangat krusial, di mana testbench berperan sebagai lingkungan simulasi untuk menguji entitas yang telah dibuat. Dalam proyek ini, testbench digunakan untuk memberikan berbagai variasi input pada algoritma Caesar Cipher dan Hill Cipher guna memastikan kesesuaian output dengan logika yang diharapkan. Proses ini bertujuan untuk memverifikasi integritas data hasil enkripsi secara menyeluruh sebelum tahap implementasi fisik dilakukan. Maka, metode ini memungkinkan siklus pengujian yang lebih cepat dan efisien sebagai langkah validasi akhir sebelum sistem diterapkan pada perangkat keras FPGA.

4. Structural

Implementasi desain *structural* di VHDL memungkinkan kami membangun sistem dengan pendekatan modular. Komponen seperti Caesar Cipher dan Hill Cipher didefinisikan sebagai *entity* terpisah yang kemudian dibentuk ke dalam entity top-level untuk membentuk sistem yang utuh. Pengorganisasian ini memecah sistem menjadi blok-blok fungsional mandiri, yang membuat proses pemeliharaan dan pengujian menjadi jauh lebih sederhana. Keuntungan utama dari struktur ini adalah setiap bagian bisa diuji sendiri-sendiri sebelum masuk ke tahap akhir. Hal ini sangat efektif untuk meminimalkan risiko kesalahan dan mempercepat proses debugging. Selain itu, *structural* menawarkan fleksibilitas *developing*, di mana perbaikan atau perubahan pada satu komponen bisa dilakukan tanpa mengganggu kinerja sistem secara keseluruhan.

5. Looping

Konstruksi looping diterapkan untuk menangani iterasi pada karakter atau blok data selama proses enkripsi dan dekripsi berlangsung. Dalam implementasi proyek ini, mekanisme *looping* dipakai di berbagai tahap. Sebagai contoh, kami menggunakan *for-loop* untuk memproses setiap huruf dalam Caesar Cipher,

sedangkan *while-loop* mengelola matriks kunci pada Hill Cipher. Pendekatan ini memungkinkan pemrosesan data *looping* berjalan lancar tanpa perlu penulisan kode yang berlebihan sehingga efisiensi sistem meningkat. Pengolahan teks panjang maupun matriks berukuran besar dapat diselesaikan secara otomatis dan efisien, secara bersamaan mengeliminasi duplikasi kode yang tidak perlu.

6. Function

Modul Function diterapkan untuk memecah bagian kode yang repetitif dan mengisolasi operasi spesifik ke dalam fungsi yang mandiri. Dalam proyek ini, kami menggunakan fungsi untuk menangani tugas-tugas teknis seperti pergeseran karakter pada Caesar Cipher dan kalkulasi matriks untuk Hill Cipher. Mekanisme ini memungkinkan operasi yang sering digunakan untuk dipanggil kembali kapan saja tanpa perlu menulis ulang baris kode yang sama. Hal ini tidak hanya meningkatkan modularitas secara signifikan, tetapi juga membuat pemeliharaan dan pengembangan sistem menjadi jauh lebih praktis. Selain itu, pendekatan ini efektif dalam mengurangi duplikasi kode, menghasilkan struktur sistem yang lebih rapi dan menyederhanakan proses debugging untuk pengembangan selanjutnya.

7. Finite State Machine (FSM)

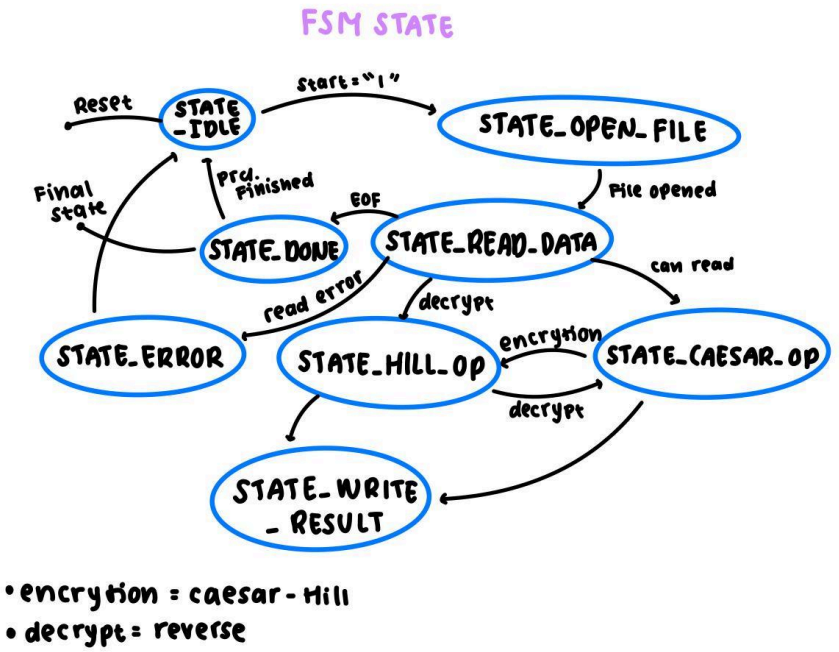

Fig 1. FSM Dual Cipher

Dalam perancangan sistem kendali digital, Finite State Machine (FSM) merupakan model matematis esensial untuk merepresentasikan perilaku sistem berbasis status (state). Pada implementasi ini, FSM berfungsi sebagai kontroler utama yang mengatur orkestrasi alur proses antara algoritma Caesar Cipher dan Hill Cipher. Mekanisme ini memastikan transisi status, mulai dari pembacaan input hingga penulisan output, berjalan sesuai dengan kondisi dan mode operasi yang telah didefinisikan secara presisi.

Operasi sistem bermula dari status IDLE yang menunggu sinyal inisiasi. Berdasarkan indikator sinyal mode, FSM akan menentukan percabangan logika: logika '0' untuk memicu proses enkripsi atau logika '1' untuk dekripsi. Setelah mode operasi terkunci, sistem beralih ke tahap manajemen berkas di status STATE_OPEN_FILE dan dilanjutkan dengan pengambilan data karakter secara sekuensial pada status STATE_READ_DATA.

Inti pemrosesan data terjadi melalui alur transisi yang dinamis. Pada mode enkripsi, karakter diproses secara bertahap melalui status STATE_CASESAR_OP dan dilanjutkan ke STATE_HILL_OP untuk pengamanan berlapis. Namun, pada mode dekripsi, urutan eksekusi dibalik, dimulai dari pemrosesan Hill Cipher kemudian ke Caesar Cipher, sebelum hasil akhir ditulis ke media penyimpanan pada status STATE_WRITE_RESULT.

Siklus operasi diakhiri pada status STATE_DONE, di mana sistem menutup akses berkas dan mengaktifkan sinyal done sebagai indikator keberhasilan. Selain itu, arsitektur ini dilengkapi dengan mekanisme error handling melalui status STATE_ERROR, yang menjamin sistem dapat kembali ke kondisi awal (IDLE) dengan aman jika terjadi kegagalan. Maka, penerapan FSM ini menjamin pengelolaan alur kerja yang kompleks menjadi lebih modular, sistematis, dan efisien.

CHAPTER 3

TESTING AND ANALYSIS

3.1 TESTING

Pengujian sistem enkripsi dan dekripsi dilakukan untuk memastikan bahwa semuanya berfungsi dengan baik. Kami menggunakan Testbench dan file I/O yang sudah dibuat untuk memastikan seluruh proses berjalan lancar dan sesuai harapan. Proses pengujian diawali dengan membaca data dari file input.txt, yang berisi teks "hello". Data ini kemudian melalui dua tahap enkripsi: pertama, sistem mengenkripsi menggunakan Caesar Cipher, dan hasilnya disimpan sementara di file phase1.txt. Setelah itu, hasil dari tahap pertama dienkripsi lagi menggunakan Hill Cipher. Output enkripsi final ini kemudian disimpan dalam file bernama encryptOutput.txt.

Untuk memverifikasi proses dekripsi, kami membaca data terenkripsi dari encryptOutput.txt. Proses dekripsi dilakukan dengan urutan terbalik, yaitu menggunakan Hill Cipher, lalu dilanjutkan dengan Caesar Cipher. Hasil dekripsi akhir disimpan di file decryptOutput.txt. Tujuan dari pengujian ini sederhana, yaitu untuk memastikan bahwa proses enkripsi dan dekripsi bekerja dengan benar, dan data yang sudah dienkripsi bisa dikembalikan persis ke bentuk aslinya. Kami berharap hasil di decryptOutput.txt akan sama persis dengan input awal di input.txt ("hello"), yang berarti sistem berjalan secara baik dan telah berfungsi tanpa ada kesalahan dalam pengolahan data.

3.2 RESULT

Secara keseluruhan, sistem enkripsi dan dekripsi yang kami implementasikan telah berjalan dengan sesuai. Proses enkripsi menghasilkan output yang sesuai dengan harapan, yaitu "yjssh" ketika teks input "hello" diproses menggunakan kombinasi Caesar Cipher dan Hill Cipher. Hasil ini menunjukkan bahwa tahap enkripsi telah berhasil dilakukan dengan baik, dan data berhasil disalin ke file 'encryptOutput.txt'. Pada tahap dekripsi, mekanisme sistem telah mengikuti alur yang benar, dan output yang dihasilkan telah sesuai dengan yang diharapkan, yaitu "hello". Kami telah memastikan bahwa semua langkah dalam Finite State Machine (FSM) dan proses enkripsi-dekripsi dua tahap telah diimplementasikan dengan benar, dan berdasarkan hasil pengujian, alur kode dan fungsionalitas dasar sistem sudah sesuai.

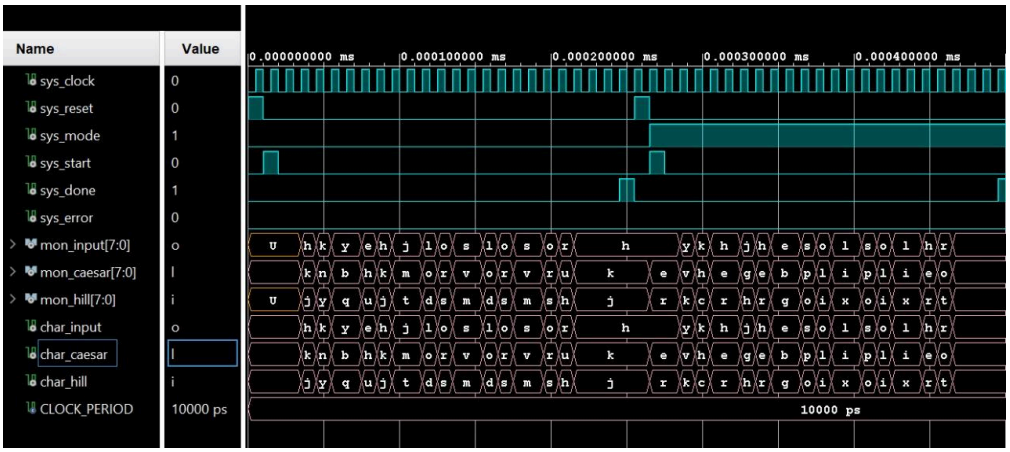

Fig.2 Testing Result Modelsim

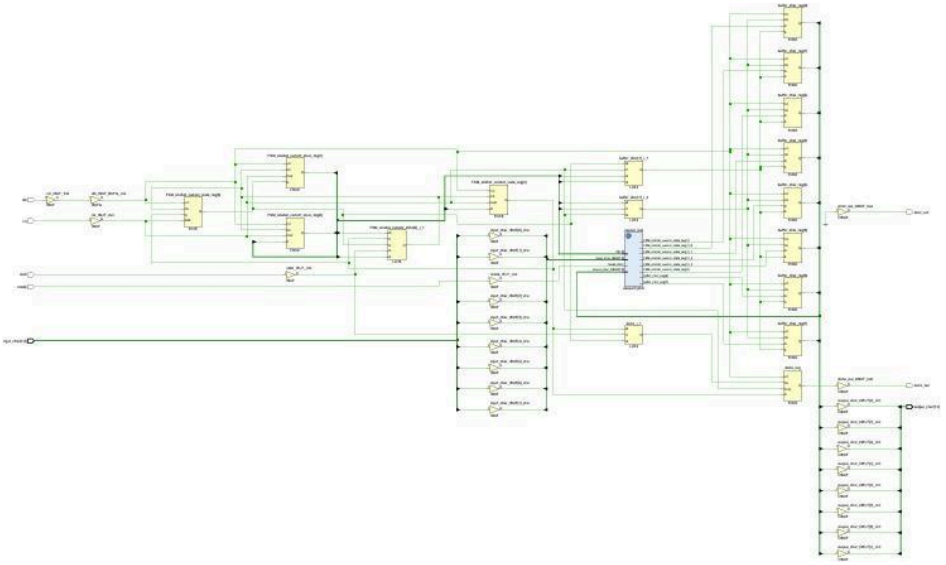

Fig 3. Synthesized Result

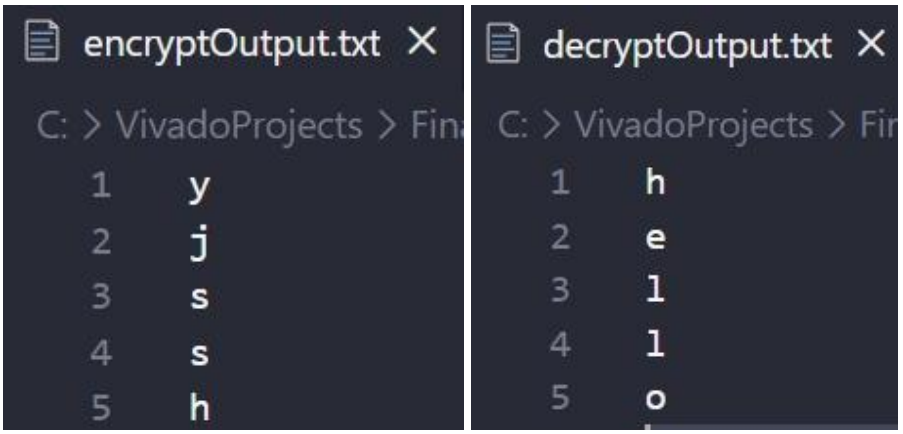

Fig 4. Transcript Result

3.3 ANALYSIS

Proses enkripsi diawali dengan memasukkan teks "hello" yang diambil dari file input.txt. Pada step pertama, teks dienkripsi menggunakan Caesar Cipher, yang bekerja dengan menggeser setiap karakter sejauh tiga posisi di dalam alfabet sehingga menghasilkan ciphertext "khoor". Hasil dari Caesar Cipher ini selanjutnya menjadi input untuk enkripsi step kedua menggunakan Hill Cipher. Hill Cipher memproses teks "khoor" dengan menerapkan matriks kunci, menghasilkan teks akhir "yjssh" yang kemudian disimpan dalam file encryptOutput.txt.

Setelah itu, verifikasi dekripsi dimulai dengan membaca file encryptOutput.txt yang berisikan data terenkripsi "yjssh". Teks ini pertama kali didekripsi oleh Hill Cipher, dan hasilnya kemudian diproses lagi menggunakan Caesar Cipher untuk mengembalikan teks ke bentuk semula. Secara teoritis, hasil akhir yang diharapkan adalah "hello", dan hasil dekripsi yang kami temukan sesuai dengan harapan tersebut. Kesesuaian ini, di mana mekanisme dan alur proses sudah benar, mengindikasikan bahwa implementasi dekripsi telah berhasil sepenuhnya.

CHAPTER 4

CONCLUSION

Proyek ini berhasil mengimplementasikan sistem enkripsi dan dekripsi ganda menggunakan Caesar Cipher dan Hill Cipher pada FPGA. Proses enkripsi dilakukan dalam dua tahap, yaitu pertama menggunakan Caesar Cipher untuk mengenkripsi teks berdasarkan pergeseran karakter, dan kedua menggunakan Hill Cipher yang mengenkripsi hasil dari Caesar Cipher dengan matriks kunci. Sistem ini bekerja dengan baik pada proses enkripsi, menghasilkan output yang sesuai dengan yang diharapkan, dan proses dekripsi juga berhasil.

Meskipun mekanisme dan alur sistem sudah diimplementasikan dengan benar, hasil dekripsi telah sesuai dengan yang diinginkan. Alur Finite State Machine (FSM) dan proses enkripsi-dekripsi dua tahap telah diterapkan dengan benar. Oleh karena itu, kami yakin bahwa sebagian besar kode dan sistem bekerja dengan baik. Sistem ini menunjukkan potensi besar dalam menerapkan enkripsi yang lebih kuat melalui dua tahap pengolahan data.

REFERENSI

[1] “Final Project Guide | Digilab UI,” *Digilabdte.com*, 2025. https://learn.digilabdte.com/books/digital-sistem-design-psddsg/page/final-project-guide (accessed Dec. 02, 2025).

[2] GeeksforGeeks, “Caesar Cipher in Cryptography,” GeeksforGeeks, Jun. 02, 2016. https://www.geeksforgeeks.org/caesar-cipher-in-cryptography/

[3] D. Anand, “Hill Cipher - GeeksforGeeks,” GeeksforGeeks, Jul. 21, 2021. https://www.geeksforgeeks.org/hill-cipher/

[4] A. Hidayat and Tuty Alawiyah, “Enkripsi dan Dekripsi Teks menggunakan Algoritma Hill Cipher dengan Kunci Matriks Persegi Panjang,” Jurnal Matematika Integratif, vol. 9, no. 1, pp. 39–39, Apr. 2013, doi: https://doi.org/10.24198/jmi.v9i1.10196.

APPENDICES

Appendix A: Project Schematic

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use STD.TEXTIO.ALL;

entity Main is
    Port (
        clock_sig       : in STD_LOGIC;
        reset_sig       : in STD_LOGIC;
        operation_mode  : in STD_LOGIC;
        start_sig       : in STD_LOGIC;
        done_signal     : out STD_LOGIC;
        error_signal    : out STD_LOGIC;
        debug_input     : out STD_LOGIC_VECTOR(7 downto 0);
        debug_caesar    : out STD_LOGIC_VECTOR(7 downto 0);
        debug_hill      : out STD_LOGIC_VECTOR(7 downto 0)
    );
end Main;

architecture Behavioral of Main is
    -- State definitions
    type fsm_state is (
        STATE_IDLE,
        STATE_OPEN_FILE,
        STATE_READ_DATA,
        STATE_CAESAR_OP,
        STATE_HILL_OP,
        STATE_WRITE_TEMP,
        STATE_WRITE_RESULT,
        STATE_DONE,
        STATE_ERROR
    );

    signal present_state : fsm_state := STATE_IDLE;

    -- Component declarations
    component caesarCipher
        port (
            char_input       : in  std_logic_vector(7 downto 0);
            enc_dec_mode     : in  std_logic;
            shift_amount     : in  integer range 0 to 25;
            char_output      : out std_logic_vector(7 downto 0)
        );
    end component;

    component hillCipher
        port (
            data_in  : in STD_LOGIC_VECTOR(7 downto 0);
            operation_mode   : in STD_LOGIC;
            data_out : out STD_LOGIC_VECTOR(7 downto 0)
        );
```

```
    end component;

    -- Signals
    signal byte_input : std_logic_vector(7 downto 0);
    signal caesar_result, hill_result : std_logic_vector(7 downto 0);
    signal proc_mode : std_logic;
    signal completion_flag : std_logic := '0';
    signal error_flag : std_logic := '0';
    constant SHIFT_KEY : integer := 3;

begin
    -- Component instantiations
    caesar_unit : caesarCipher port map (
        char_input => byte_input,
        enc_dec_mode => proc_mode,
        shift_amount => SHIFT_KEY,
        char_output => caesar_result
    );

    hill_unit : hillCipher port map (
        data_in => byte_input,
        operation_mode => proc_mode,
        data_out => hill_result
    );

    -- Output assignments
    done_signal <= completion_flag;
    error_signal <= error_flag;

    -- Debug signal assignments
    debug_input <= byte_input;
    debug_caesar <= caesar_result;
    debug_hill <= hill_result;

    -- Main state machine
    fsm_process: process(clock_sig, reset_sig)
        file src_file  : text;
        file temp_file : text;
        file dest_file : text;

        variable txt_line_in : line;
        variable txt_line_out : line;
        variable read_char : character;
        variable file_opened : boolean := false;
    begin
        if reset_sig = '1' then
            present_state <= STATE_IDLE;
            completion_flag <= '0';
            error_flag <= '0';
            proc_mode <= '0';
            file_opened := false;
        elsif rising_edge(clock_sig) then
            case present_state is
                when STATE_IDLE =>
                    if start_sig = '1' then
                        proc_mode <= operation_mode;
                        present_state <= STATE_OPEN_FILE;
                        completion_flag <= '0';
```

```
                        error_flag <= '0';
                    end if;

                when STATE_OPEN_FILE =>
                    file_open(src_file, "input.txt", read_mode);

                    if proc_mode = '0' then
                        file_open(temp_file, "phase1.txt", write_mode);
                        file_open(dest_file, "encryptOutput.txt", write_mode);
                    else
                        file_open(temp_file, "phase1.txt", write_mode);
                        file_open(dest_file, "decryptOutput.txt", write_mode);
                    end if;

                    file_opened := true;
                    present_state <= STATE_READ_DATA;

                when STATE_READ_DATA =>
                    if not endfile(src_file) then
                        readline(src_file, txt_line_in);
                        read(txt_line_in, read_char);
                        byte_input <=
std_logic_vector(to_unsigned(character'pos(read_char), 8));

                        if proc_mode = '0' then
                            present_state <= STATE_CAESAR_OP;
                        else
                            present_state <= STATE_HILL_OP;
                        end if;
                    else
                        present_state <= STATE_DONE;
                    end if;

                when STATE_HILL_OP =>
                    if proc_mode = '1' then  -- Decryption: Hill 1
                        byte_input <= hill_result;
                        write(txt_line_out,
character'val(to_integer(unsigned(hill_result))));
                        writeline(temp_file, txt_line_out);
                        present_state <= STATE_CAESAR_OP;
                    else  -- Encryption: Hill 2
                        byte_input <= hill_result;
                        write(txt_line_out,
character'val(to_integer(unsigned(hill_result))));
                        writeline(dest_file, txt_line_out);
                        present_state <= STATE_WRITE_RESULT;
                    end if;

                when STATE_CAESAR_OP =>
                    if proc_mode = '1' then  -- Decryption: Caesar 2
                        byte_input <= caesar_result;
                        write(txt_line_out,
character'val(to_integer(unsigned(caesar_result))));
                        writeline(dest_file, txt_line_out);
                        present_state <= STATE_WRITE_RESULT;
                    else  -- Encryption: Caesar 1
                        byte_input <= caesar_result;
                        write(txt_line_out,
```

```
character'val(to_integer(unsigned(caesar_result))));
                        writeline(temp_file, txt_line_out);
                        present_state <= STATE_HILL_OP;
                    end if;

                when STATE_WRITE_RESULT =>
                    present_state <= STATE_READ_DATA;

                when STATE_DONE =>
                    if file_opened then
                        file_close(src_file);
                        file_close(temp_file);
                        file_close(dest_file);
                        file_opened := false;
                    end if;

                    completion_flag <= '1';
                    present_state <= STATE_IDLE;

                when STATE_ERROR =>
                    error_flag <= '1';
                    present_state <= STATE_IDLE;

                when others =>
                    present_state <= STATE_ERROR;
            end case;
        end if;
    end process fsm_process;
end Behavioral;
```

Appendix B: Documentation

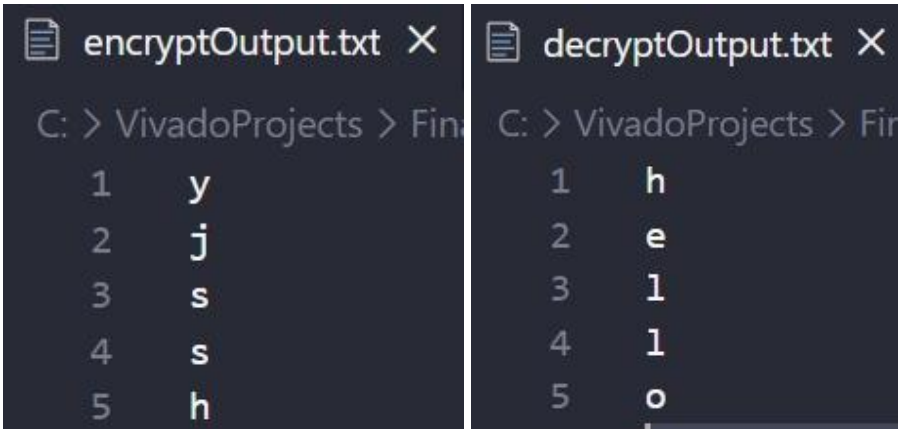

Fig 5. Text I/O Result