



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

SISTEM ENKRIPSI DATA MENGGUNAKAN
ALGORITMA RSA DENGAN RANDOM
PRIME GENERATOR

GROUP PA-05

Maharaka Fadhilah	2306225520
R. Aisha Syauqi Ramadhani	2306250554
Tri Yoga Arsyad	2306161920

PREFACE

Puji syukur kami panjatkan kepada Tuhan Yang Maha Esa, yang telah memberikan rahmat dan karunia-Nya sehingga kami dapat menyelesaikan proyek akhir ini dengan baik. Proyek ini merupakan salah satu langkah penting dalam penerapan teori dan pengetahuan yang kami peroleh selama menjalani studi, khususnya dalam bidang sistem digital dan kriptografi. Kami mengucapkan terima kasih kepada semua pihak yang telah mendukung dan memberikan bantuan selama pengerjaan proyek ini.

Proyek akhir ini berjudul “Sistem Enkripsi Data Menggunakan Algoritma RSA dengan Random Prime Generator.”. Tujuan dari proyek ini adalah untuk merancang dan mengimplementasikan sistem enkripsi RSA menggunakan bahasa pemrograman VHDL, yang melibatkan pembuatan bilangan prima acak, pembangkitan kunci publik dan privat, serta proses enkripsi dan dekripsi pesan secara aman. Dengan adanya sistem ini, diharapkan dapat memahami lebih dalam mengenai penerapan algoritma kriptografi RSA dalam bentuk perangkat keras yang dapat dijalankan menggunakan VHDL.

Kami juga ingin mengucapkan terima kasih kepada asisten digital lab yang telah memberikan bimbingan, arahan, dan masukan yang sangat berharga selama proses perancangan dan implementasi sistem ini. Tanpa bimbingan dan dukungan dari beliau, kami tidak akan dapat menyelesaikan proyek ini dengan baik. Ucapan terima kasih juga kami sampaikan kepada teman-teman sekelompok PA-05 yang telah bekerja sama dengan penuh semangat dan komitmen.

Proyek ini sepenuhnya diuji menggunakan ModelSim untuk simulasi sistem yang telah kami bangun. Hasil simulasi dengan input karakter menunjukkan bahwa sistem enkripsi dan dekripsi RSA yang kami desain berjalan dengan baik dan sesuai dengan prinsip-prinsip dasar kriptografi. Kami berharap proyek ini dapat memberikan kontribusi dalam memahami implementasi sistem kriptografi di ranah perangkat keras dan memberikan manfaat bagi pengembangan teknologi di masa depan.

Depok, December 4, 2024

Group PA-05

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION.....	
1.1 Background.....	
1.2 Project Description.....	
1.3 Objectives.....	
1.4 Roles and Responsibilities.....	
CHAPTER 2: IMPLEMENTATION4.....	
2.1 Equipment.....	
2.2 Implementation.....	
CHAPTER 3: TESTING AND 	
3.1 Testing.....	
3.2 Result.....	
3.3 Analysis.....	
CHAPTER 4: CONCLUSION4.....	
REFERENCES4.....	
APPENDICES4	
Appendix A: Project Schematic	
Appendix B: Documentation	

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Perkembangan teknologi informasi yang pesat membawa dampak besar terhadap cara kita mengelola dan berbagi data. Seiring dengan semakin kompleksnya komunikasi digital, masalah **keamanan data** menjadi semakin penting. Data pribadi, informasi bisnis, serta data sensitif lainnya sering kali ditransmisikan melalui jaringan yang tidak sepenuhnya aman, sehingga rentan terhadap serangan dan penyalahgunaan. Dalam konteks ini, kriptografi menjadi solusi utama untuk memastikan bahwa data yang dikirimkan tetap aman dan hanya dapat diakses oleh pihak yang berwenang.

Salah satu algoritma kriptografi yang paling banyak digunakan adalah **RSA** (Rivest-Shamir-Adleman), yang ditemukan pada tahun 1977. RSA menggunakan prinsip matematika yang melibatkan **faktorisasi bilangan bulat besar**. Kunci publik dan kunci privat yang digunakan dalam RSA memungkinkan proses enkripsi dan dekripsi yang aman, di mana kunci publik digunakan untuk mengenkripsi pesan dan kunci privat digunakan untuk mendekripsinya. Meskipun algoritma ini sangat aman, kecepatan pemrosesan dan efisiensi tetap menjadi tantangan utama dalam aplikasinya.

Penerapan RSA umumnya dilakukan dalam perangkat lunak, namun implementasi kriptografi menggunakan perangkat keras menawarkan beberapa keuntungan. Dengan menggunakan perangkat keras, algoritma RSA dapat dioptimalkan untuk meningkatkan **kecepatan pemrosesan** dan mengurangi **latensi** dalam proses enkripsi dan dekripsi data. Selain itu, implementasi berbasis perangkat keras cenderung lebih aman karena dapat mengurangi risiko serangan perangkat lunak yang sering kali dapat mengeksploitasi kelemahan pada sistem berbasis perangkat lunak.

Implementasi RSA dalam perangkat keras umumnya menggunakan bahasa deskripsi perangkat keras seperti **VHDL (VHSIC Hardware Description Language)**. VHDL memungkinkan desainer untuk menggambarkan dan mengimplementasikan logika sirkuit digital secara lebih terstruktur dan efisien. Melalui penggunaan VHDL, RSA dapat diintegrasikan dalam sistem digital, memanfaatkan kemampuan perangkat keras untuk melakukan operasi matematika

yang diperlukan untuk kriptografi dengan lebih cepat dan lebih efisien daripada implementasi perangkat lunak.

Proyek ini bertujuan untuk mengimplementasikan algoritma RSA dalam VHDL, dengan fokus pada pembuatan sistem yang dapat menghasilkan **kunci publik dan privat** secara otomatis dan melakukan proses **enkripsi dan dekripsi** dengan cepat dan aman. Dengan demikian, diharapkan sistem RSA berbasis perangkat keras ini dapat memberikan solusi yang lebih baik dalam hal keamanan dan efisiensi, khususnya untuk aplikasi yang membutuhkan pemrosesan data dalam jumlah besar dengan tingkat keamanan yang tinggi.

1.2 PROJECT DESCRIPTION

Proyek ini bertujuan untuk merancang sistem enkripsi dan dekripsi menggunakan algoritma RSA yang diimplementasikan dalam bahasa VHDL. Sistem ini akan berfungsi untuk mengenkripsi dan mendekripsi pesan karakter yang dikirimkan antara pengirim dan penerima dengan menggunakan kunci publik dan privat yang dihasilkan oleh generator bilangan prima acak. Proses enkripsi dilakukan dengan mengaplikasikan kunci enkripsi (e) dan kunci n , sementara dekripsi menggunakan kunci dekripsi (d) untuk memulihkan pesan asli.

Sistem ini dirancang untuk beroperasi pada dua modul utama, yaitu `Finpro_tb` dan `DecryptorUnit`, yang saling berinteraksi. `Finpro_tb` mengirimkan karakter yang akan dienkripsi, sementara `DecryptorUnit` melakukan enkripsi dan dekripsi berdasarkan kunci yang dihasilkan oleh komponen `KeyGenUnit`. Pengujian dan simulasi sistem dilakukan sepenuhnya di ModelSim, dengan menggunakan input berupa string karakter yang dikirim dari `Finpro_tb` ke `DecryptorUnit`.

Simulasi di ModelSim bertujuan untuk menguji proses enkripsi dan dekripsi karakter dengan bilangan prima yang dihasilkan oleh `PrimeGen`. Setiap langkah proses enkripsi dan dekripsi ditangani dalam urutan state yang terstruktur, dimulai dari `FETCH`, `ENCRYPT`, `SHOW_ENCRYPTION`, `DECRYPT`, hingga `SHOW_DECRYPTION`. Proses ini memastikan bahwa sistem dapat berjalan sesuai dengan fungsinya, dengan memastikan kunci yang digunakan tetap konsisten selama proses.

Fitur utama dari sistem ini termasuk kemampuan untuk menghasilkan kunci publik dan privat secara otomatis, serta mengaplikasikan algoritma RSA untuk enkripsi dan dekripsi pesan. Selain itu, sistem juga dilengkapi dengan generator bilangan prima acak yang digunakan dalam pembangkitan kunci RSA. Fitur lainnya adalah pengujian dan analisis kesalahan yang dilakukan selama simulasi untuk memastikan bahwa sistem dapat berfungsi dengan baik dalam kondisi yang telah ditentukan.

1.3 OBJECTIVES

Proyek ini memiliki tujuan utama sebagai berikut:

1. Merancang sistem enkripsi RSA menggunakan VHDL.
2. Menghasilkan kunci publik dan privat berdasarkan angka prima acak.
3. Mengimplementasikan algoritma enkripsi dan dekripsi RSA.
4. Melakukan pengujian sistem dengan simulasi di ModelSim menggunakan input karakter.
5. Menyediakan fitur penghasil angka prima, generator kunci, pengirim dan penerima pesan.

1.4 ROLES AND RESPONSIBILITIES

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

Roles	Responsibilities	Person
System Designer	Mendesain sistem secara keseluruhan	Raka, Ica, Yoga
Prime Number Generator Developer	Mengembangkan dan mengimplementasikan komponen PrimeGen untuk menghasilkan bilangan prima	Raka, Ica, Yoga
Key Generator Developer	Mengembangkan komponen KeyGenUnit untuk	Raka, Ica, Yoga

	menghasilkan kunci publik dan private	
Finpro_tb and DecryptorUnit Developer	Mengembangkan dan mengimplementasikan testbench dan komponen DecryptorUnit untuk enkripsi dan dekripsi	Raka, Ica, Yoga
System Tester	Mengkompilasi dan mensimulasikan kode pada Modelsim	Raka, Ica, Yoga
Menyusun laporan, PPT, dan Readme	Menulis laporan dan dokumentasi serta README terkait proyek akhir	Raka, Ica, Yoga

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:

- ModelSim
- Git
- GitHub
- VHDL
- Quartus
- Draw.io
- Visual Studio Code

2.2 IMPLEMENTATION

Implementasi sistem enkripsi RSA melibatkan beberapa komponen utama yang bekerja bersama untuk menghasilkan kunci, mengenkripsi, dan mendekripsi pesan. Berikut adalah gambaran komponen-komponen utama dalam sistem ini yang diimplementasi dalam file kode:

1. “PrimeGen.vhd”

Komponen ini menghasilkan dua bilangan prima besar, yaitu p dan q , yang sangat penting dalam proses pembuatan kunci. Menggunakan generator angka acak (Random Prime Generator) dan memeriksa keprimaannya dengan metode uji coba pembagian (trial division). Setelah dua bilangan prima ditemukan, nilai-nilai tersebut diteruskan untuk digunakan dalam pembuatan kunci.

2. “KeyGenUnit.vhd”

Komponen KeyGenUnit bertugas untuk menghasilkan kunci publik dan kunci privat berdasarkan bilangan prima p dan q yang telah dihasilkan oleh PrimeGen.

Komponen ini menghitung:

- $n = p * q$ (digunakan dalam kedua kunci, publik dan privat).
- $\phi(n) = (p-1) * (q-1)$ (fungsi totient dari n , digunakan untuk mencari kunci enkripsi dan dekripsi).
- Kunci enkripsi (e) yang merupakan bilangan bulat yang relatif prima terhadap $\phi(n)$.
- Kunci dekripsi (d) yang merupakan kebalikan modular dari e mod $\phi(n)$, sehingga memenuhi syarat $d * e \equiv 1 \pmod{\phi(n)}$.

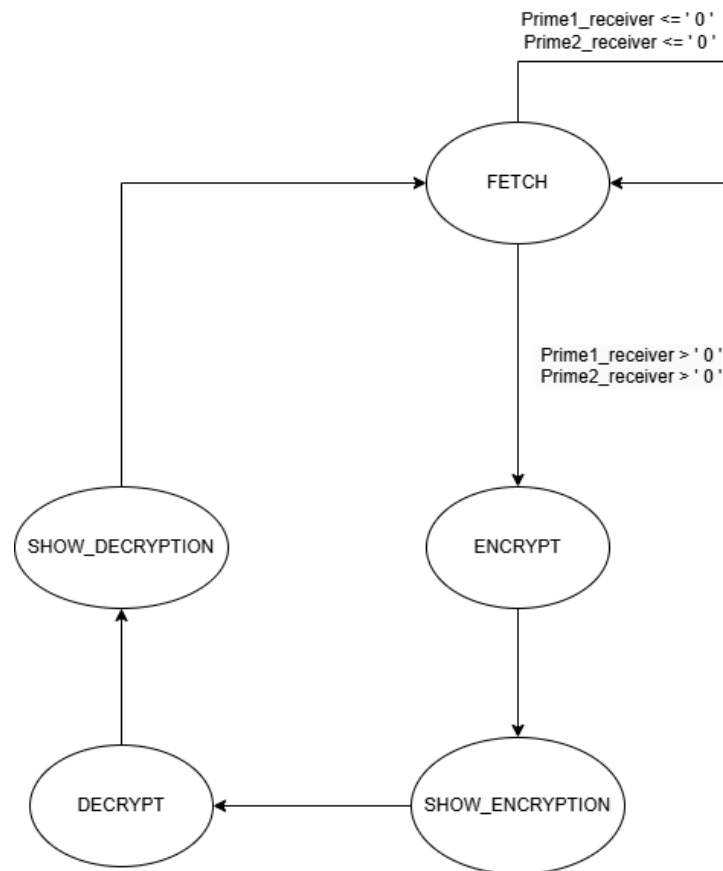
3. “Finpro_tb.vhd”

Komponen ini bertugas untuk mengirimkan pesan yang telah dienkripsi sekaligus menguji semua komponen sistem RSA yang telah diimplementasikan. Komponen ini menerima input berupa pesan, kemudian mengenkripsi pesan menggunakan kunci publik yang dihasilkan oleh Alice (penerima kunci). Setelah pesan dienkripsi, pesan terenkripsi dikirim ke penerima melalui saluran komunikasi yang tidak aman. Sebagai testbench komponen ini juga memastikan bahwa proses enkripsi dan dekripsi berjalan dengan benar serta bekerja sesuai dengan yang diinginkan dalam pengujian simulasi

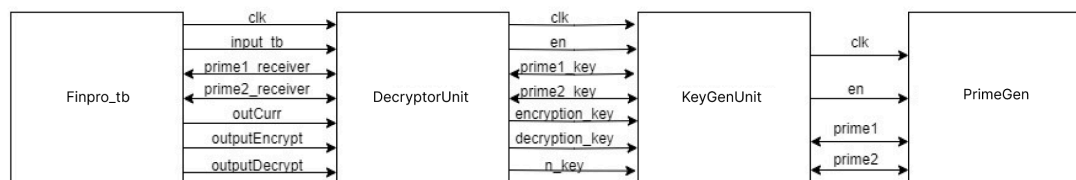
4. “DecryptorUnit.vhd”

Komponen ini bertugas untuk menerima pesan yang terenkripsi dan mendekripsinya menggunakan kunci privat. DecryptorUnit ini akan menggunakan kunci privat untuk mendekripsi pesan yang diterima. Proses ini melibatkan state machine yang mengatur alur data dan kontrol antara enkripsi dan dekripsi pada setiap karakter input.

Diagram Alur Proses



Gambar 1. State Diagram Program



Gambar 2. Block Diagram Program

Secara keseluruhan, sistem ini mengalir melalui beberapa tahap berikut:

1. **Pembuatan Kunci:** Menghasilkan kunci publik dan privat menggunakan bilangan prima acak.
2. **Enkripsi:** Menggunakan kunci publik untuk mengenkripsi pesan.
3. **Dekripsi:** Menggunakan kunci privat untuk mendekripsi pesan yang diterima.

Komponen-komponen ini bekerja secara sinergis untuk memastikan bahwa sistem RSA berfungsi dengan baik dan aman dalam komunikasi data.

Implementasi pada Setiap Modul

1. Dataflow

Modul ini menangani aliran data selama proses pembangkitan bilangan prima dan perhitungan aritmatika modular, memastikan bahwa data dapat mengalir dengan benar antar komponen.

2. Behavioral

Modul ini mendefinisikan logika secara deskriptif untuk proses enkripsi, dekripsi, dan pembangkitan kunci, menggunakan pendekatan perilaku (behavioral) dalam penulisan kode VHDL.

3. Testbench

Modul ini berfungsi untuk menguji berbagai komponen sistem, termasuk pembangkitan bilangan prima, generator kunci, serta proses enkripsi dan dekripsi, untuk memastikan bahwa setiap bagian sistem bekerja dengan baik.

4. Structural

Modul ini digunakan untuk mengintegrasikan semua komponen utama sistem, seperti PrimeGen, KeyGenUnit, Finpro_tb, dan DecryptorUnit, untuk membentuk keseluruhan sistem yang saling terhubung.

5. Looping

Modul ini mengimplementasikan iterasi dalam perhitungan algoritma RSA, seperti mencari bilangan coprime dan perhitungan eksponensial modular, yang diperlukan dalam pembuatan kunci dan enkripsi/dekripsi pesan.

6. FSM (Finite State Machine)

Modul ini mengatur tahapan-tahapan dalam proses enkripsi dan dekripsi dengan menggunakan state machine, termasuk pengendalian alur data di Finpro_tb dan DecryptorUnit selama proses enkripsi dan dekripsi.

```

LIBRARY IEEE;

USE IEEE.std_logic_1164.ALL;

USE IEEE.numeric_std.ALL;

ENTITY Finpro_tb IS

    PORT (

        clk : IN STD_LOGIC;

        prime1_receiver, prime2_receiver : BUFFER INTEGER;

        outCurr : OUT CHARACTER;

        outputEncrypt : OUT CHARACTER;

        outputDecrypt : OUT CHARACTER

    );

END ENTITY Finpro_tb;

ARCHITECTURE rtl OF Finpro_tb IS

    COMPONENT DecryptorUnit IS

        PORT (

            clk : IN STD_LOGIC;

            input : IN STRING (1 TO 19);

            prime1_receiver, prime2_receiver : BUFFER INTEGER;

```

```

        outCurr : OUT CHARACTER;

        outputEncrypt : OUT CHARACTER;

        outputDecrypt : OUT CHARACTER

    );

    END COMPONENT DecryptorUnit;

    SIGNAL input_tb : STRING(1 TO 19) := "B C D E F G H I J K";
BEGIN

    Receiver1 : DecryptorUnit PORT MAP(

        clk => clk,

        input => input_tb,

        prime1_receiver => prime1_receiver,

        prime2_receiver => prime2_receiver,

        outCurr => outCurr,

        outputEncrypt => outputEncrypt,

        outputDecrypt => outputDecrypt

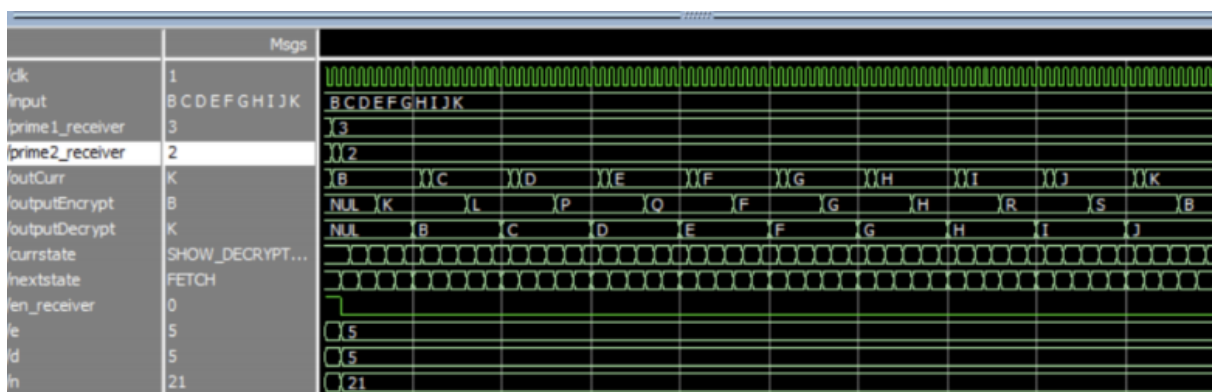
    );

END ARCHITECTURE rtl;

```

MAIN PROGRAM

- DecryptorUnit.vhd



Gambar 4. Output Program DecryptorUnit (receiver)

Source Code:

```

LIBRARY IEEE;

USE IEEE.std_logic_1164.ALL;

USE IEEE.numeric_std.ALL;

ENTITY DecryptorUnit IS

    PORT (

        clk : IN STD_LOGIC;

        input : IN STRING (1 TO 19) := "B C D E F G H I J K";

        prime1_receiver, prime2_receiver : BUFFER INTEGER;

        outCurr : OUT CHARACTER;

        outputEncrypt : OUT CHARACTER;

        outputDecrypt : OUT CHARACTER

    );

END ENTITY DecryptorUnit;

ARCHITECTURE rtl OF DecryptorUnit IS

    COMPONENT KeyGenUnit IS

        PORT (

            clk : IN STD_LOGIC;

            en : IN STD_LOGIC;

            prime1_key, prime2_key : BUFFER INTEGER;

            encryption_key, decryption_key, n_key : OUT INTEGER

        );

    END COMPONENT KeyGenUnit;

    TYPE state IS (FETCH, ENCRYPT, SHOW_ENCRYPTION, DECRYPT, SHOW_DECRYPTION);

    SIGNAL currstate, nextstate : state;

    SIGNAL en_receiver : STD_LOGIC := '1';

    SIGNAL e, d, n : INTEGER;

BEGIN

```

```

KeyGen : KeyGenUnit PORT MAP(

    clk => clk,

    en => en_receiver,

    prime1_key => prime1_receiver,

    prime2_key => prime2_receiver,

    encryption_key => e,

    decryption_key => d,

    n_key => n

);

PROCESS (clk)

    VARIABLE i_int, e_int, d_int : INTEGER := 0;

    VARIABLE pc : INTEGER := 1;

BEGIN

    IF rising_edge(clk) THEN

        outCurr <= input(pc);

        CASE currstate IS

            WHEN FETCH =>

                i_int := CHARACTER'pos(input(pc)) - 64;

                IF prime1_receiver > 0 AND prime2_receiver > 0 THEN

                    en_receiver <= '0';

                    nextstate <= ENCRYPT;

                END IF;

            WHEN ENCRYPT =>

                e_int := (i_int ** e) MOD n;

                nextstate <= SHOW_ENCRYPTION;

            WHEN SHOW_ENCRYPTION =>

                outputEncrypt <= CHARACTER'val(e_int + 64);

                nextstate <= DECRYPT;

            WHEN DECRYPT =>

                d_int := (e_int ** d) MOD n;

                nextstate <= SHOW_DECRYPTION;

```

```

        WHEN SHOW_DECRYPTION =>

            outputDecrypt <= CHARACTER'val(d_int + 64);

            pc := pc + 1;

            nextstate <= FETCH;

        END CASE;

    END IF;

END PROCESS;

PROCESS (clk)

BEGIN

    IF rising_edge(clk) THEN

        currstate <= nextstate;

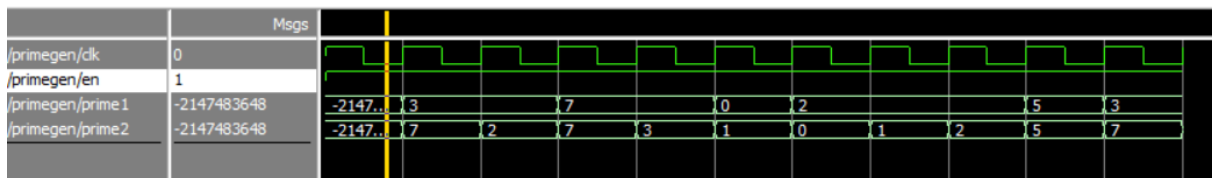
    END IF;

END PROCESS;

END ARCHITECTURE rtl;

```

- PrimeGen.vhd



Gambar 5. Output Program PrimeGen

Source Code:

```

library IEEE;

use IEEE.std_logic_1164.all;

use IEEE.numeric_std.all;

use IEEE.math_real.all;

entity PrimeGen is

    port (

        clk: in std_logic;

```



```

        en: in std_logic;

        prime1, prime2: buffer integer

    );
end entity PrimeGen;

architecture rtl of PrimeGen is

begin

    process (clk)

        variable seed1: integer := 1;

        variable seed2: integer := 2;

        variable rand_int1, rand_int2: integer;

        variable is_prime1, is_prime2: boolean;

        --Random Number Generator Function

        impure function rand_int_gen(

            min: integer;

            max: integer

        ) return integer is

            variable randomValue: real;

        begin

            uniform(seed1, seed2, randomValue);

            return integer(round(randomValue * real(max - min + 1) + real(min) -
0.5));

        end function;

    begin

        if rising_edge(clk) AND en = '1' then

            is_prime1 := false;

            is_prime2 := false;

            while not (is_prime1 and is_prime2) loop

                rand_int1 := rand_int_gen(0, 8);

                rand_int2 := rand_int_gen(0, 8);

```

```

--Primality testing using trial division method

is_prime1 := true;

for i in 2 to integer(sqrt(real(rand_int1))) loop

    if rand_int1 mod i = 0 then

        is_prime1 := false;

        exit;

    end if;

end loop;

is_prime2 := true;

for i in 2 to integer(sqrt(real(rand_int2))) loop

    if rand_int2 mod i = 0 then

        is_prime2 := false;

        exit;

    end if;

end loop;

end loop;

prime1 <= rand_int1;

prime2 <= rand_int2;








end if;

end process;

end architecture rtl;

```

- KeyGenUnit.vhd

	Msgs	
/keygenunit/dk	0	
/keygenunit/en	1	
/keygenunit/prime1...	2	
/keygenunit/prime2...	1	
/keygenunit/encryp...	5	
/keygenunit/decry...	5	
/keygenunit/h_key	0	

Gambar 6. Output Program KeyGenUnit

Source Code:

```

library IEEE;

use IEEE.std_logic_1164.all;

use IEEE.numeric_std.all;

entity KeyGenUnit is

    port (

        clk: in std_logic;

        en: in std_logic;

        prime1_key, prime2_key: buffer integer;

        encryption_key, decryption_key, n_key: out integer

    );

end entity KeyGenUnit;

architecture rtl of KeyGenUnit is

    component PrimeGen is

        port (

            clk: in std_logic;

            en: in std_logic;

            prime1, prime2: out integer

        );

    end component PrimeGen;

    --Coprime Testing Function

    function IsCoprime(A, B: INTEGER) return BOOLEAN is

        variable Remainder: INTEGER;

        variable TempA, TempB: INTEGER;

    begin

        TempA := A;

        TempB := B;

        if TempA > TempB then

            TempA := B;

            TempB := A;

```

```

        end if;

        loop

            Remainder := TempB mod TempA;

            exit when Remainder = 0;

            TempB := TempA;

            TempA := Remainder;

        end loop;

        return TempA = 1;

    end IsCoprime;

begin

    PrimeGen1: PrimeGen port map (

        clk => clk,

        en => en,

        prime1 => prime1_key,

        prime2 => prime2_key

    );

    process(clk)

        variable N, T, e, d: integer := 0;

    begin

        if rising_edge(clk) AND en = '1' then

            N := prime1_key * prime2_key;

            T := (prime1_key - 1) * (prime2_key - 1);

            --Algorithm to find e (encryption key)

            for i in 2 to T - 1 loop

                if T mod i /= 0 and N mod i /= 0 and IsCoprime(i, T) and
IsCoprime(i, N) then

                    e := i;

                    exit;

                end if;

```

```

        end loop;

        --Algorithm to find d (decryption key)
        for i in 1 to T loop
            if (e * i) mod T = 1 then
                d := i;
                exit;
            end if;
        end loop;
    end if;

    encryption_key <= e;
    decryption_key <= d;

    n_key <= N;
end process;

end architecture rtl;

```

3.2 ANALYSIS

- Finpro_tb.vhd

Prime1_receiver dan prime2_receiver adalah bilangan prima yang dihasilkan oleh PrimeGen. outCurr merupakan karakter yang sedang melalui proses enkripsi dan dekripsi. outputEncrypt adalah hasil enkripsi dari input, sedangkan outputDecrypt adalah hasil dekripsi. input_tb adalah input dari testbench, yang nantinya akan dikirim dari sender ke input receiver.

- DecryptorUnit.vhd

E merupakan kunci enkripsi, d merupakan kunci dekripsi, dan n adalah kunci n. Jika tidak menggunakan testbench, input dapat dimasukkan secara manual, dan pada dasarnya tidak perlu ada nilai apapun di input DecryptorUnit. Urutan statenya adalah sebagai berikut:

FETCH → ENCRYPT → SHOW_ENCRYPTION → DECRYPT → SHOW_DECRYPTION.

- PrimeGen.vhd

Fungsi dan nilai e yang digunakan dalam enkripsi sama seperti yang dihasilkan oleh KeyGenUnit. Di sini, meskipun generator angka acak dapat digunakan, di penerima akan selalu menggunakan prime1 dan prime2 yang memiliki nilai 3 dan 7. Mengapa? Karena jika setiap huruf menggunakan prime1 dan prime2 yang berbeda, maka kunci enkripsi (e key), kunci dekripsi (d key), dan kunci n (n key) akan berbeda untuk setiap huruf. Hal ini menyebabkan proses enkripsi-dekripsi menjadi salah. Jadi, dapat dikatakan bahwa urutan angka acak yang digunakan pada penerima hanya menggunakan yang pertama kali muncul. Ini tidak benar-benar acak di penerima, tetapi informasi ini dimasukkan ke dalam laporan untuk menunjukkan bahwa PrimeGen berfungsi. Pada bagian pengembangan atau catatan kesalahan program, akan dijelaskan bahwa penerima tidak dapat mengambil nilai prima selain prime1 dan prime2 yang pertama.

- KeyGenUnit.vhd

En digunakan untuk mengaktifkan komponen KeyGenerator sehingga dapat berjalan. Nilai en diperoleh dari DecryptorUnit dan fungsinya adalah agar KeyGenUnit tidak terus-menerus menghasilkan kunci saat proses enkripsi-dekripsi berlangsung di DecryptorUnit. Jika proses berjalan terus menerus, maka akan muncul kesalahan karena DecryptorUnit akan mendapatkan kunci pertama, sedang diproses, dan sebelum proses selesai, masuk lagi dengan kunci kedua yang memiliki nilai berbeda. Oleh karena itu, en dinonaktifkan saat DecryptorUnit sudah memiliki kunci, dan kondisi ini dapat dilihat pada bagian state FETCH di DecryptorUnit.

CHAPTER 4

4.1 CONCLUSION

Sistem enkripsi RSA yang dirancang menggunakan VHDL dalam proyek ini telah berhasil diimplementasikan dan diuji dengan baik. Komponen-komponen utama, seperti PrimeGen, KeyGenUnit, Finpro_tb, dan DecryptorUnit, bekerja secara sinergis untuk memastikan proses enkripsi dan dekripsi pesan berjalan sesuai dengan algoritma RSA. Pengujian melalui testbench di ModelSim membuktikan bahwa sistem dapat menghasilkan kunci publik dan privat secara acak, serta memproses pesan terenkripsi dan mendekripsinya kembali ke bentuk aslinya dengan akurasi tinggi.

Proyek ini menunjukkan bahwa algoritma RSA dapat diterapkan secara efisien dalam desain sistem digital berbasis perangkat keras. Dengan memanfaatkan pendekatan modular, aliran data yang terstruktur, dan kontrol menggunakan Finite State Machine (FSM), sistem ini memberikan wawasan penting tentang pengembangan teknologi kriptografi yang aman dan andal. Implementasi ini juga membuka peluang untuk eksplorasi lebih lanjut dalam aplikasi keamanan data berbasis perangkat keras di masa depan.

4.2 REFLEKSI

Meskipun program sudah berjalan, masih ada beberapa aspek yang dapat dikembangkan lebih lanjut. Salah satunya adalah pada komponen DecryptorUnit.vhd, yang saat ini hanya mampu membaca sekuens bilangan prima pertama saja. Jika nilai bilangan prima berubah, program akan mengalami kesalahan. Selain itu, kami juga menghadapi masalah saat melakukan sintesis program di aplikasi Quartus. Kesalahan ini disebabkan oleh ketidakcocokan antara program kami dan *compiler* Quartus. Sebagai solusinya, kami menggantinya dengan sebuah diagram blok yang dibuat menggunakan *draw.io* (<https://app.diagrams.net/>). Ke depannya, kami berharap dapat memperbaiki kekurangan dan menyempurnakan program ini.

REFERENCES

- [1]. Rouse, M. (2017) What is a random number generator (RNG)? Available at: <https://www.techopedia.com/definition/9091/random-number-generator-rng> (Accessed: 4 December 2024).
- [2]. Cobb, M. (2021) What is the RSA algorithm? Available at: <https://www.techtarget.com/searchsecurity/definition/RSA> (Accessed: 4 December 2024).
- [3]. Encyclopedia, (no date) How does a Random Number Generator work? Available at: <https://www.hypr.com/security/encyclopedia/random-number-generator> (Accessed: 4 December 2024).
- [4]. Jensen, J.J. (2023) How to create a finite-state machine in VHDL. Available at: <https://vhdlwhiz.com/finite-state-machine/> (Accessed: 4 December 2024).
- [5]. GeeksforGeeks (2021) Trial Division algorithm for prime factorization. Available at: <https://www.geeksforgeeks.org/trial-division-algorithm-for-prime-factorization/> (Accessed: 4 December 2024).
- [6]. Encryption Consulting (2020) What is RSA? how does RSA work? Available at: <https://www.encryptionconsulting.com/education-center/what-is-rsa/> (Accessed: 4 December 2024).
- [7]. GeeksforGeeks (2023) RSA algorithm in cryptography. Available at: <https://www.geeksforgeeks.org/rsa-algorithm-cryptography/> (Accessed: 4 December 2024).

APPENDICES

Appendix A: Project Schematic

Put your final project latest schematic here

Appendix B: Documentation

Put the documentation (photos) during the making of the project