



Proyek Akhir PSD

# RSA Encryptor

By: Kelompok PA-05

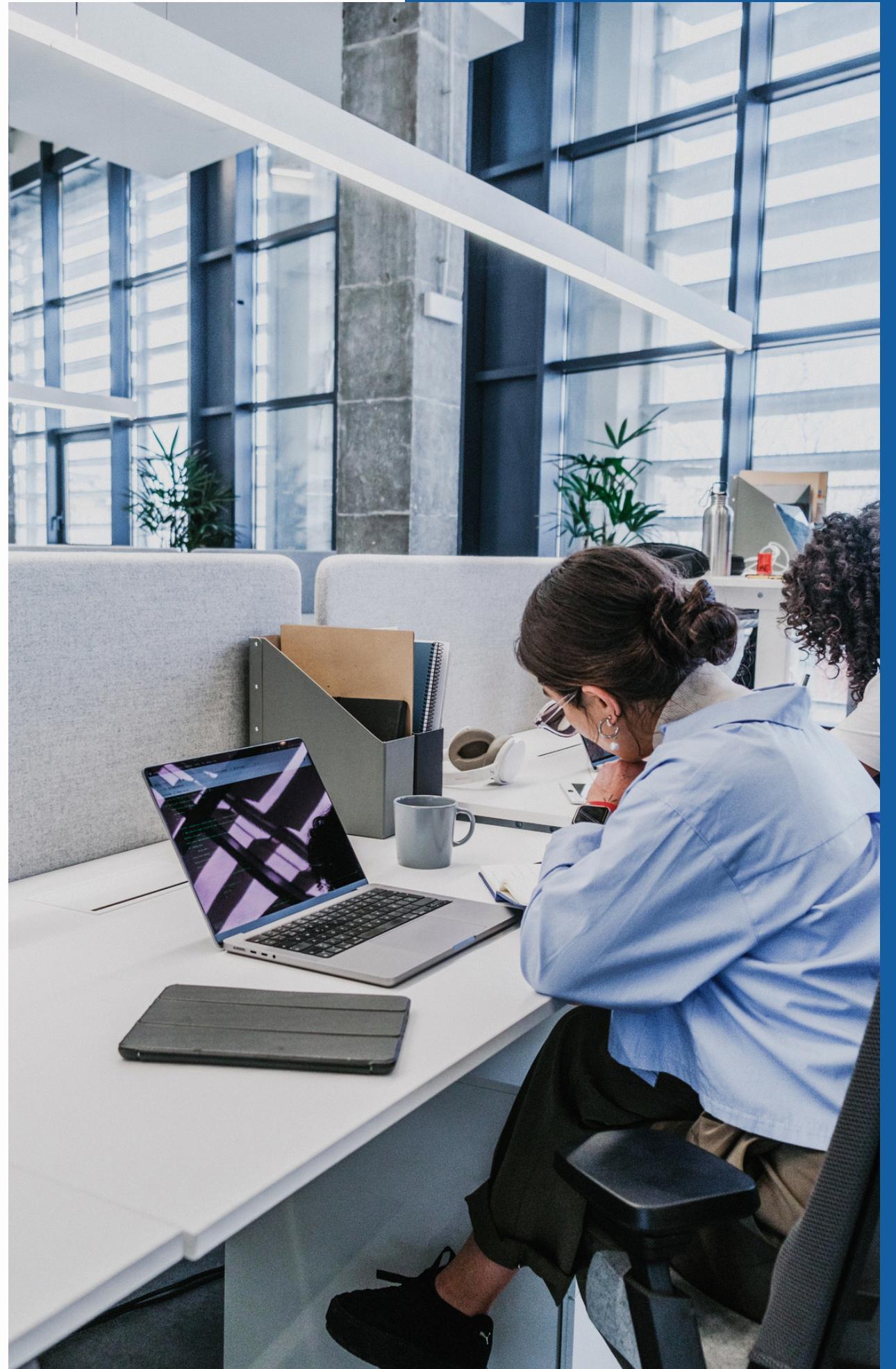
```
if ($window.scrollTop() > header0_initialDistance) {
    if (parseInt(header0.css('padding-top'), 10) > header0_initialPadding) {
        header0.css('padding-top', '' + $window.scrollTop() - header0_initialDistance);
    }
} else {
    header0.css('padding-top', '' + header0_initialPadding + 'px');
}

if ($window.scrollTop() > header1_initialDistance) {
    if (parseInt(header1.css('padding-top'), 10) > header1_initialPadding) {
        header1.css('padding-top', '' + $window.scrollTop() - header1_initialDistance);
    }
} else {
    header1.css('padding-top', '' + header1_initialPadding + 'px');
}

if ($window.scrollTop() > header2_initialDistance) {
    if (parseInt(header2.css('padding-top'), 10) > header2_initialPadding) {
        header2.css('padding-top', '' + $window.scrollTop() - header2_initialDistance);
    }
} else {
    header2.css('padding-top', '' + header2_initialPadding + 'px');
}
```

# Overview

- ▶ Goal
- ▶ Background
- ▶ How it Works
- ▶ State Diagram
- ▶ Block Diagram
- ▶ VHDL Code
- ▶ Result and Testing
- ▶ Our Team





# Goal

- Merancang metode enkripsi RSA untuk mengenkripsi pesan berbentuk string
- Mengimplementasikan pemrograman menggunakan bahasa VHDL,
- Memenuhi nilai praktikum Perancangan Sistem Digital.



# Background

Keamanan data sangat penting dalam komunikasi digital, karena data sering kali rentan terhadap serangan di jaringan yang tidak aman. RSA (Rivest-Shamir-Adleman) adalah algoritma kriptografi yang menggunakan kunci publik untuk enkripsi dan kunci privat untuk dekripsi, dengan keamanan berdasarkan faktorisasi bilangan besar. Meskipun aman, RSA dalam perangkat lunak cenderung lambat.

Implementasi di perangkat keras dengan VHDL dapat meningkatkan kecepatan pemrosesan dan keamanan. Proyek ini bertujuan mengimplementasikan RSA menggunakan VHDL dengan generator bilangan prima acak untuk menghasilkan kunci publik dan privat serta proses enkripsi-dekripsi yang cepat dan aman.





# How it Works

Entitas Finpro\_tb berperan sebagai pengirim pesan. Pada awalnya, Finpro\_tb mengirimkan pesan ke entitas DecryptorUnit untuk diproses dengan enkripsi dan dekripsi menggunakan algoritma RSA. Proses ini melibatkan komponen DecryptorUnit sebagai modul utama yang berinteraksi dengan entitas lain untuk melakukan enkripsi dan dekripsi pesan.

Entitas DecryptorUnit sebagai modul utama menerima pesan dari Finpro\_tb, kemudian melakukan enkripsi dan dekripsi dengan menggunakan kunci publik dan kunci privat yang dihasilkan oleh entitas KeyGenUnit. Enkripsi dilakukan dengan mengubah karakter pesan menjadi bilangan bulat, lalu menjalankan operasi eksponensial dan modulo menggunakan kunci publik. Pesan yang terenkripsi kemudian dikirim kembali ke Finpro\_tb, dan proses dekripsi dilakukan di DecryptorUnit menggunakan kunci privat yang sesuai.

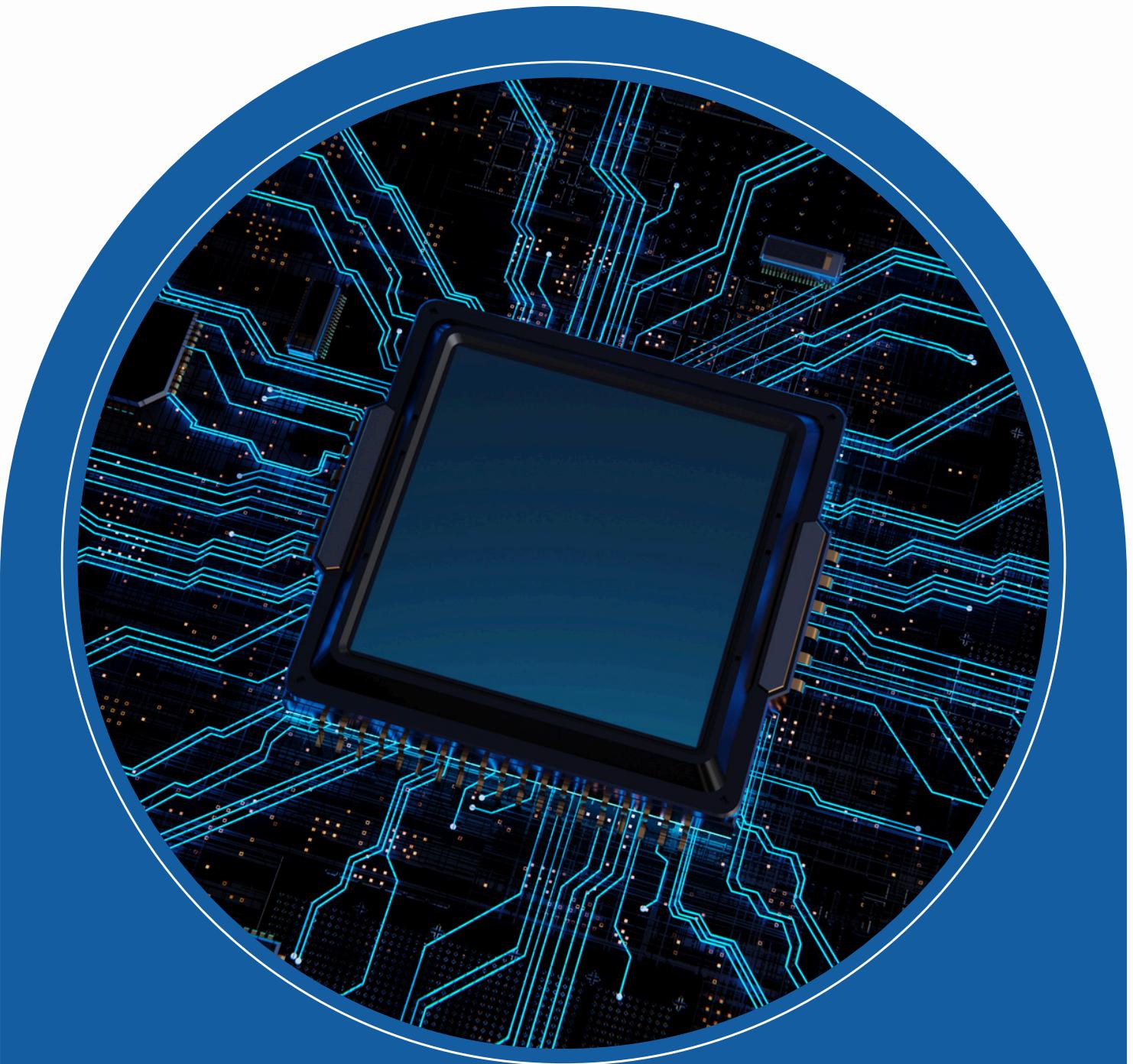




## How it Works

Entitas KeyGenUnit berfungsi untuk menghasilkan kunci publik dan kunci privat yang digunakan dalam proses enkripsi dan dekripsi pesan. Komponen PrimeGen pada KeyGenUnit menghasilkan bilangan prima secara acak yang menjadi dasar untuk pembuatan kunci publik dan kunci privat dalam algoritma RSA.

Proses algoritma RSA dijalankan dalam entitas DecryptorUnit. Ketika pesan diterima, karakter pesan diubah menjadi bilangan bulat, kemudian dilakukan operasi matematika sesuai langkah-langkah enkripsi RSA. Hasil enkripsi dikirim kembali ke Finpro\_tb untuk ditampilkan, sementara dekripsi dilakukan di DecryptorUnit menggunakan kunci privat yang sesuai, dan hasil dekripsi ditampilkan.



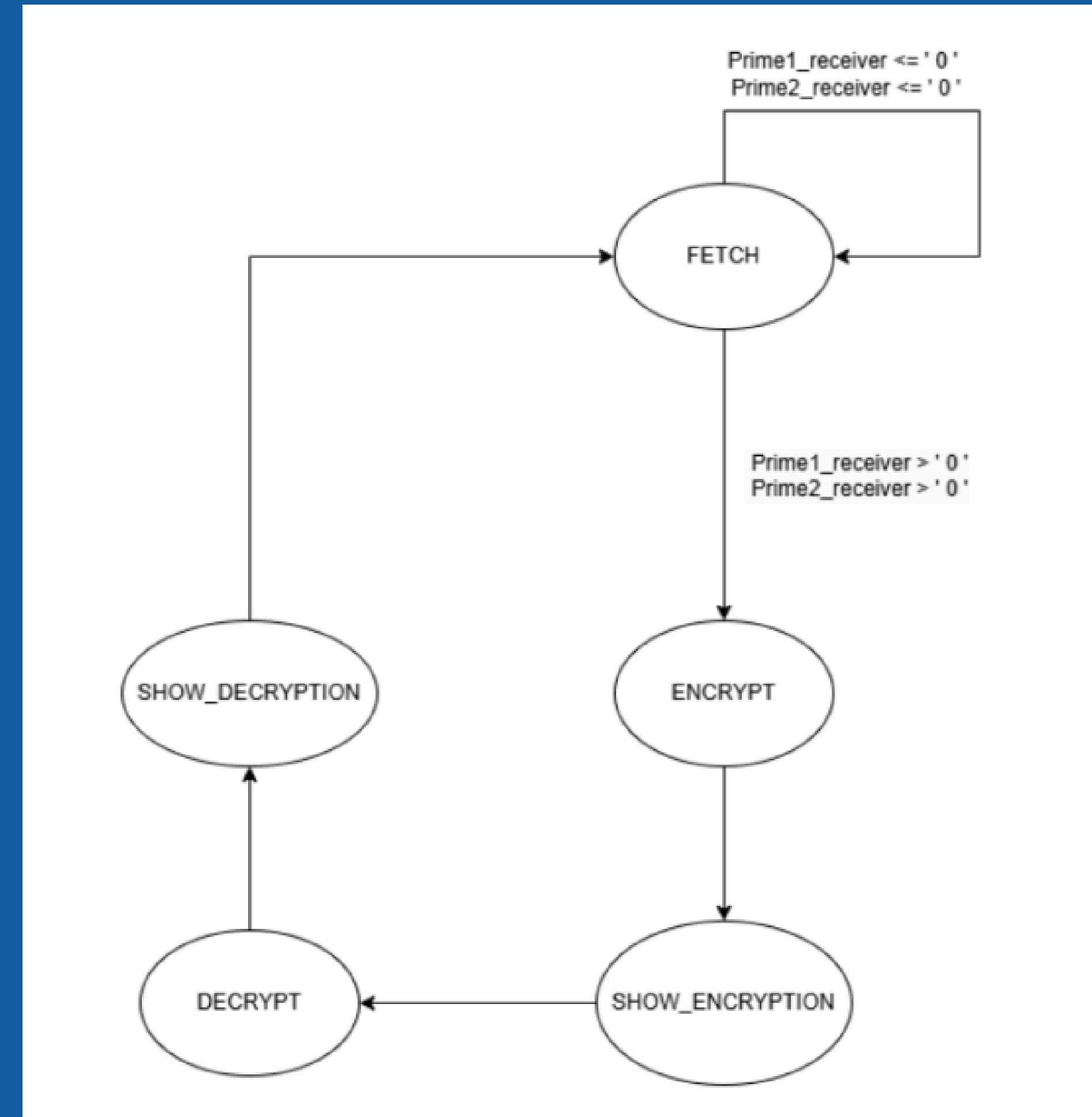


# State Diagram

## Proyek Akhir PSD

Diagram ini menunjukkan siklus lengkap dari pengambilan data, enkripsi, dan dekripsi, dengan kondisi berikut:

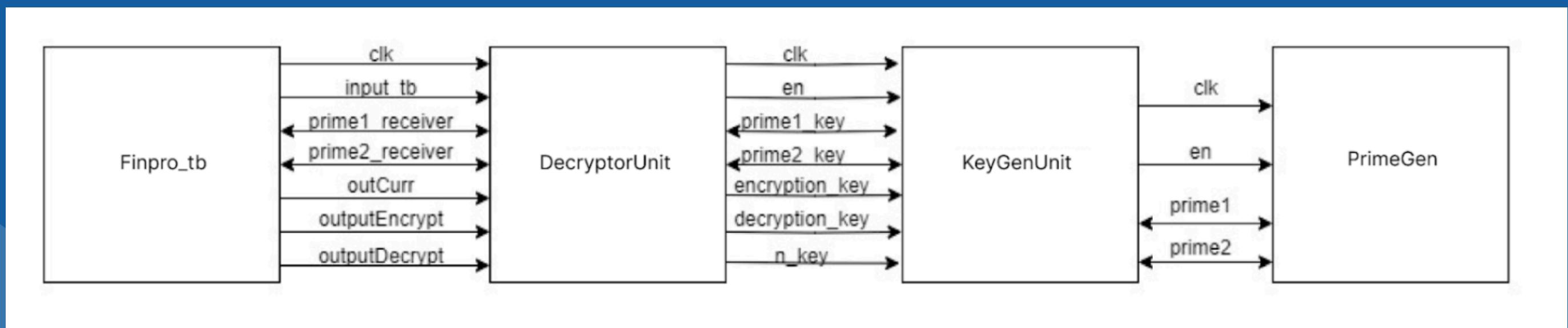
- FETCH adalah tempat sistem menunggu input valid. Jika nilai Prime1\_receiver dan Prime2\_receiver > 0, sistem melanjutkan ke tahap berikutnya. Jika tidak, sistem tetap di sini.
- Pada ENCRYPT, data yang valid diproses untuk dienkripsi.
- Setelah enkripsi selesai, hasilnya ditampilkan di SHOW\_ENCRYPTION, memastikan pengguna dapat melihat data yang telah dienkripsi.
- Proses berlanjut ke DECRYPT, di mana data dienkripsi kembali ke bentuk aslinya.
- Hasil dekripsi ditampilkan di SHOW\_DECRYPTION untuk konfirmasi bahwa data telah berhasil dipulihkan.
- Setelah itu, sistem kembali ke FETCH, mempersiapkan siklus baru jika diperlukan.





# Block Diagram

- Finpro\_tb: Testbench untuk menguji sistem dengan input clock, data, dan menerima output seperti hasil enkripsi dan dekripsi.
- DecryptorUnit: Memproses enkripsi dan dekripsi menggunakan kunci yang dihasilkan.
- KeyGenUnit: Menghasilkan kunci enkripsi, dekripsi, dan nilai n\_key berdasarkan bilangan prima.
- PrimeGen: Membuat bilangan prima (prime1 dan prime2) sebagai dasar untuk menghasilkan kunci.



```
- Finpro_tb.vhd
1 LIBRARY IEEE;
2 USE IEEE.std_logic_1164.ALL;
3 USE IEEE.numeric_std.ALL;
4
5 ENTITY Finpro_tb IS
6 PORT (
7     clk : IN STD_LOGIC;
8     prime1_receiver, prime2_receiver : BUFFER INTEGER;
9     outCurr : OUT CHARACTER;
10    outputEncrypt : OUT CHARACTER;
11    outputDecrypt : OUT CHARACTER
12 );
13 END ENTITY Finpro_tb;
14
15 ARCHITECTURE rtl OF Finpro_tb IS
16 COMPONENT DecryptorUnit IS
17 PORT (
18     clk : IN STD_LOGIC;
19     input : IN STRING (1 TO 19);
20     prime1_receiver, prime2_receiver : BUFFER INTEGER;
21     outCurr : OUT CHARACTER;
22     outputEncrypt : OUT CHARACTER;
23     outputDecrypt : OUT CHARACTER
24 );
25 END COMPONENT DecryptorUnit;
26
27 SIGNAL input_tb : STRING(1 TO 19) := "B C D E F G H I J K";
28 BEGIN
29     Receiver1 : DecryptorUnit PORT MAP(
30         clk => clk,
31         input => input_tb,
32         prime1_receiver => prime1_receiver,
33         prime2_receiver => prime2_receiver,
34         outCurr => outCurr,
35         outputEncrypt => outputEncrypt,
36         outputDecrypt => outputDecrypt
37 );
38 END ARCHITECTURE rtl;
```

# Testbench

- Deklarasi Entity Finpro\_tb: Mendefinisikan port untuk clock (clk), dua bilangan prima (prime1\_receiver dan prime2\_receiver), serta output karakter untuk proses enkripsi dan dekripsi (outCurr, outputEncrypt, outputDecrypt).
- Instansiasi Komponen DecryptorUnit: Menghubungkan komponen DecryptorUnit yang berfungsi untuk mengenkripsi dan mendekripsi data dengan port-port yang telah didefinisikan di Finpro\_tb.
- Input Testbench: Kode ini juga berfungsi sebagai pengirim, karena input\_tb berisi string yang akan diteruskan ke DecryptorUnit untuk diproses.
- Pengujian dengan Testbench: Menyediakan sinyal input yang akan diproses oleh DecryptorUnit, sambil mengamati output dari proses enkripsi dan dekripsi untuk memastikan kinerja komponen tersebut.



Pr

## Entity Finpro\_tb

```
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;
```

**Entity Finpro\_tb mendefinisikan interface dari testbench.**

- **clk:** Sinyal clock untuk sinkronisasi.
- **prime1\_receiver, prime2\_receiver:** Buffer integer untuk bilangan prima yang diterima.
- **outCurr:** Karakter yang sedang diproses (input karakter).
- **outputEncrypt:** Karakter hasil enkripsi.
- **outputDecrypt:** Karakter hasil dekripsi.

**BUFFER** digunakan untuk sinyal yang perlu dibaca dan ditulis dalam Testbench. Dalam hal ini, bilangan prima yang diterima akan dimodifikasi oleh komponen lain dan akan dibaca kembali.

## Architecture rtl

```
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;
```

**Arsitektur rtl:** Bagian ini mendefinisikan komponen yang ada dalam testbench. Di sini, DecryptorUnit adalah komponen yang diuji.

- **DecryptorUnit memiliki port yang mencakup input clock, input string, dua bilangan prima yang diterima, serta output untuk karakter yang sedang diproses, hasil enkripsi, dan hasil dekripsi.**

## Mapping

```
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;
```

- **Signal input\_tb:** Mendefinisikan string yang berisi karakter-karakter yang akan dienkripsi dan didekripsi oleh sistem. Karakter-karakter ini adalah data yang akan diproses oleh DecryptorUnit.
- **Receiver1:** Menyambungkan (port map) komponen DecryptorUnit dengan sinyal-sinyal yang ada di Finpro\_tb. Di sini, input karakter (input\_tb), bilangan prima (prime1\_receiver, prime2\_receiver), dan output karakter yang dihasilkan dari enkripsi dan dekripsi akan dipetakan ke DecryptorUnit.

- **Testbench ini menguji DecryptorUnit yang melakukan proses enkripsi dan dekripsi dengan input karakter dari input\_tb.**
- **prime1\_receiver dan prime2\_receiver adalah bilangan prima yang digunakan untuk enkripsi dan dekripsi.**
- **DecryptorUnit akan mengenkripsi dan mendekripsi string karakter satu per satu, dan hasilnya akan dipetakan ke outputEncrypt dan outputDecrypt.**

Testbench ini menyimulasikan proses enkripsi dan dekripsi untuk string "B C D E F G H I J K" dengan bilangan prima yang diterima melalui prime1\_receiver dan prime2\_receiver. Kode ini berfungsi untuk memverifikasi bahwa sistem RSA yang diimplementasikan dapat memproses data dengan benar, dengan menghasilkan output enkripsi dan dekripsi yang sesuai.

## Proyek Akhir PSD

```
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";
    currChar : 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;
```

# DecryptorUnit

- Menginisialisasi Input: Membaca karakter dari string input satu per satu dan mengonversinya menjadi nilai numerik.
- Enkripsi: Setiap karakter dikonversi dan dienkripsi menggunakan formula  $(i_{int}^{**} e) \text{ MOD } n$ .
- Dekripsi: Hasil enkripsi didekripsi kembali menggunakan formula  $(e_{int} d) \text{ MOD } n$ .
- Menampilkan Hasil Dekripsi: Hasil dekripsi dikonversi kembali menjadi karakter dan disimpan di outputDecrypt.
- - Mengulang Proses: Proses diulangi untuk semua karakter dalam string input hingga selesai.

## Deklarasi State Machine

```

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;

```

**Mendefinisikan tipe state (FETCH, ENCRYPT, dll.) yang digunakan untuk mengontrol proses enkripsi dan dekripsi.**

## Proses Transisi State:

```

PROCESS (clk)
BEGIN
  IF rising_edge(clk) THEN
    currstate <= nextstate;
  END IF;
END PROCESS;

```

**Mengatur transisi dari satu state ke state berikutnya berdasarkan hasil pemrosesan.**

## Proses Enkripsi dan Dekripsi

```

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;

```

**Menangani logika enkripsi dan dekripsi karakter satu per satu.**

**Conclusion :** ini merupakan main code mengimplementasikan unit dekripsi dan enkripsi berbasis RSA, yang mengenkripsi dan mendekripsi karakter dari string input menggunakan kunci yang dihasilkan dari dua bilangan prima.

```
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: out integer;
        prime2: out 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;
    begin
        --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(rround(randomValue * real(max - min + 1) + real(min) - .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;
```

# PrimeGen

- Menghasilkan Angka Acak: Angka acak dihasilkan menggunakan fungsi rand\_int\_gen.
- Pengecekan Keprimaan: Setiap angka acak diuji untuk keprimaannya menggunakan metode trial division.
- Mengulang Proses: Jika salah satu angka bukan bilangan prima, maka proses pengacakan dan pengecekan akan diulang.
- Menghasilkan Dua Bilangan Prima: Setelah ditemukan dua bilangan prima yang valid, mereka diteruskan ke komponen lain untuk digunakan dalam pembuatan kunci RSA.

## Inisialisasi Variabel:

```
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;
```

- **seed1 dan seed2:** untuk inisialisasi generator angka acak.
- **rand\_int1 dan rand\_int2:** menyimpan angka acak yang diuji untuk keprimaannya.
- **is\_prime1 dan is\_prime2:** menandakan apakah angka tersebut merupakan bilangan prima.

## Pengecekan Prima:

```
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;
```

Melakukan pengecekan apakah **rand\_int1** bisa dibagi habis oleh angka lain selain 1 dan dirinya sendiri. Jika bisa, maka **is\_prime1** di-set ke **false**.

## Fungsi Generator Angka Acak:

```
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;
```

**Fungsi rand\_int\_gen** menghasilkan angka acak antara **min** dan **max** menggunakan teknik **uniform random number generation**.

## Proses Pencarian Bilangan Prima:

- Loop pertama mencari dua bilangan acak (**rand\_int1** dan **rand\_int2**).
- Loop kedua melakukan pengecekan keprimaan untuk kedua bilangan acak yang dihasilkan (**rand\_int1** dan **rand\_int2**).
- Jika keduanya bilangan prima, maka proses berhenti dan kedua angka tersebut digunakan sebagai bilangan prima dalam pembuatan kunci RSA.

## Proyek Akhir PSD

```
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
    );

```

```
        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;

        -- Output the generated keys
        encryption_key <= e;
        decryption_key <= d;
        n_key <= N;
    end process;
end architecture rtl;
```

# KeyGenUnit

- Mengambil Bilangan Prima: Mengambil sejumlah dua bilangan prima dari komponen PrimeGen untuk digunakan dalam perhitungan kunci.
- Perhitungan Kunci Enkripsi dan Dekripsi: Menghitung kunci enkripsi (e) dan dekripsi (d) dengan menerapkan algoritma RSA, memastikan bahwa nilai-nilai tersebut adalah coprime.
- Menghasilkan Nilai N: Menhitung nilai N sebagai hasil perkalian dari kedua bilangan prima, yang berfungsi dalam proses enkripsi dan dekripsi.

## Perhitungan Nilai N dan T

```
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);
```

Menghitung nilai N dan T berdasarkan dua bilangan prima yang dihasilkan oleh komponen PrimeGen.

## Pencarian Nilai d (Kunci Dekripsi)

```
-- 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;
```

Menentukan e sebagai coprime dengan T dan N menggunakan IsCoprime. (memastikan tidak memiliki faktor pembagi yang sama)

## Fungsi Pencarian Nilai e (Kunci Enkripsi)

```
-- 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;
```

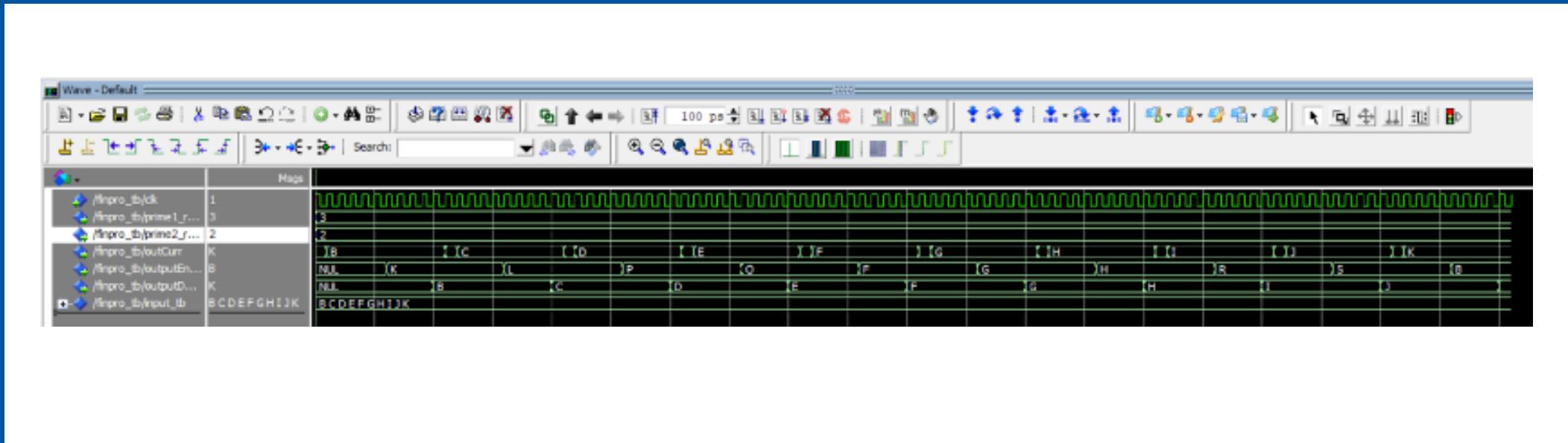
Mencari e yang coprime dengan T dan N menggunakan IsCoprime.

### Conclusion:

Kode ini dirancang untuk menghasilkan kunci enkripsi dan dekripsi menggunakan algoritma RSA. Prosesnya mencakup perhitungan modulus (N) dan totient (T), pemilihan kunci enkripsi (e) yang coprime dengan T dan N, serta penentuan kunci dekripsi (d) melalui penggunaan persamaan modular.

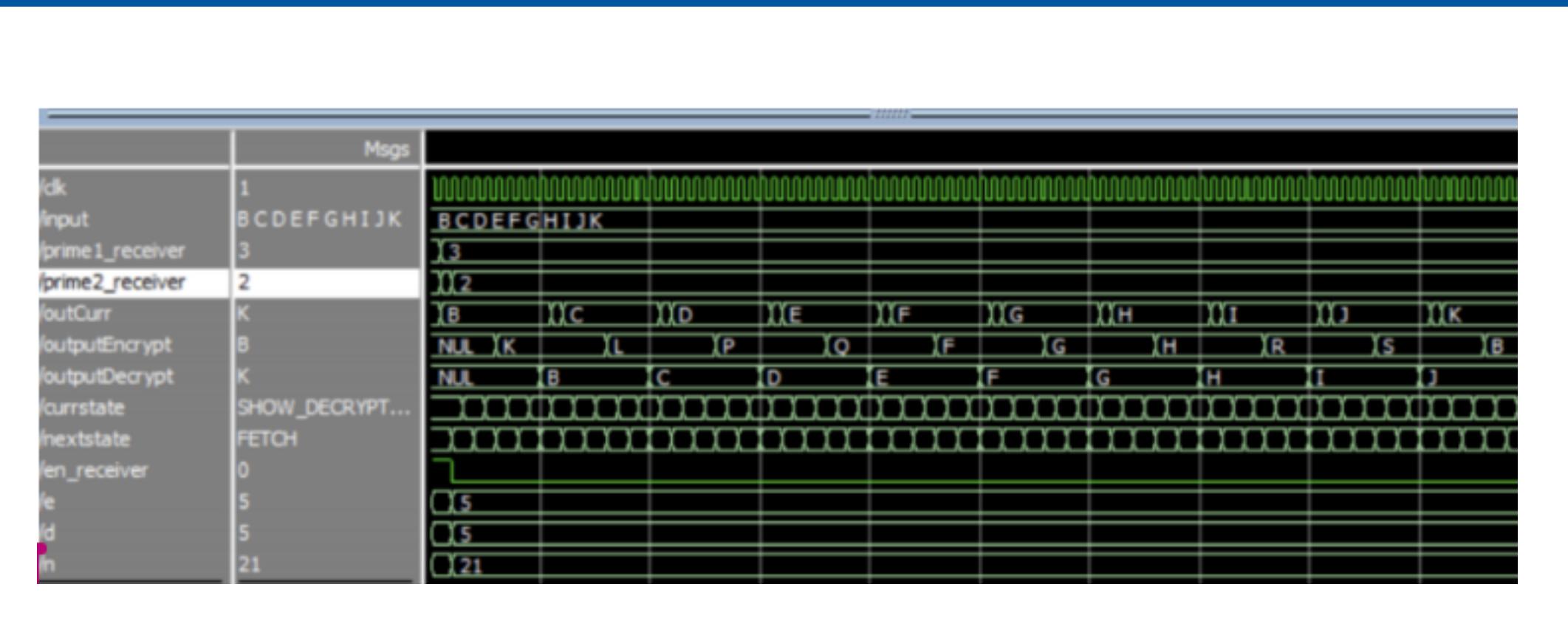


## Testing



### Waveform (Testbench)

- 1. Input (A, B):** Sinyal Input\_A dan Input\_B bervariasi untuk menguji logika desain.
- 2. Output (Result):** Output berubah sesuai input, menunjukkan operasi sesuai logika.
- 3. Validasi:** Output konsisten dengan input, menunjukkan modul bekerja sesuai spesifikasi.

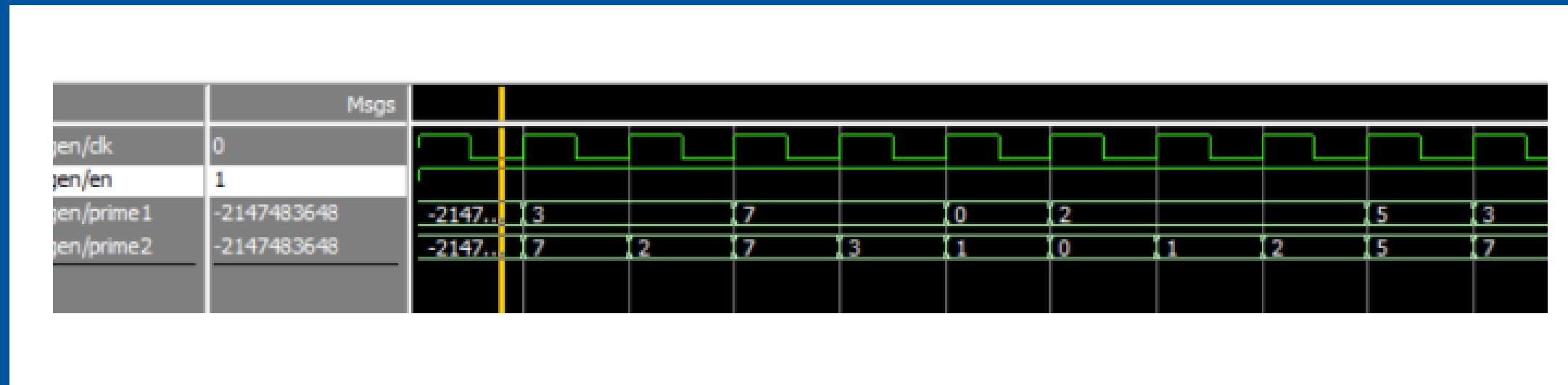


### Waveform (DecryptorUnit)

- 1. Input (input, en\_receiver):** Menunjukkan data masuk dan sinyal aktif untuk penerimaan data.
- 2. Sinyal Internal (prime1\_receiver, prime2\_receiver):** Menampilkan status proses pengolahan data.
- 3. Output (outputDecrypt, outputEncrypt):** Menunjukkan hasil enkripsi atau dekripsi sesuai input.

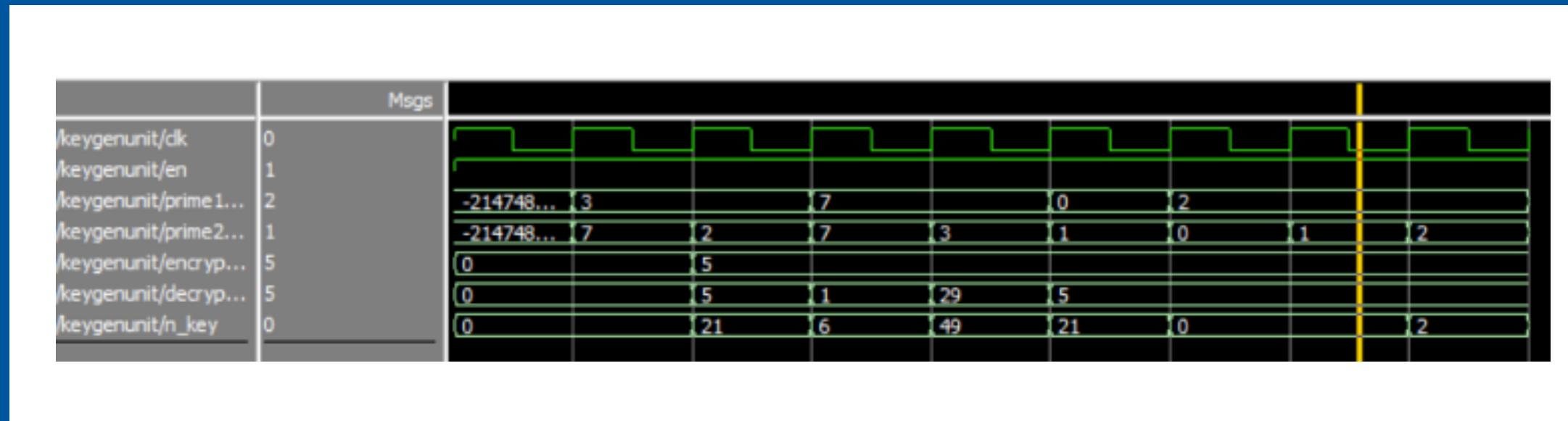


## Testing



### Waveform (PrimeGen)

- 1. Clock (clk):** Menunjukkan siklus clock yang teratur.
- 2. Enable (en):** Mengaktifkan proses pada sinyal lainnya.
- 3. Prime (prime1, prime2):** Menunjukkan keluaran yang bervariasi setiap siklus.



### Waveform (KeyGenUnit)

- 1. Input:** Sinyal `/keygenunit/clk` sebagai clock dan `/keygenunit/en` mengaktifkan modul dengan input bilangan prima `/prime1` dan `/prime2`.
- 2. Output:** Modul menghasilkan `/encrypt_key`, `/decrypt_key`, dan `/n_key` berdasarkan input.
- 3. Proses:** Saat `en` aktif, output berubah mengikuti clock sesuai kalkulasi internal.



## Proyek Akhir PSD

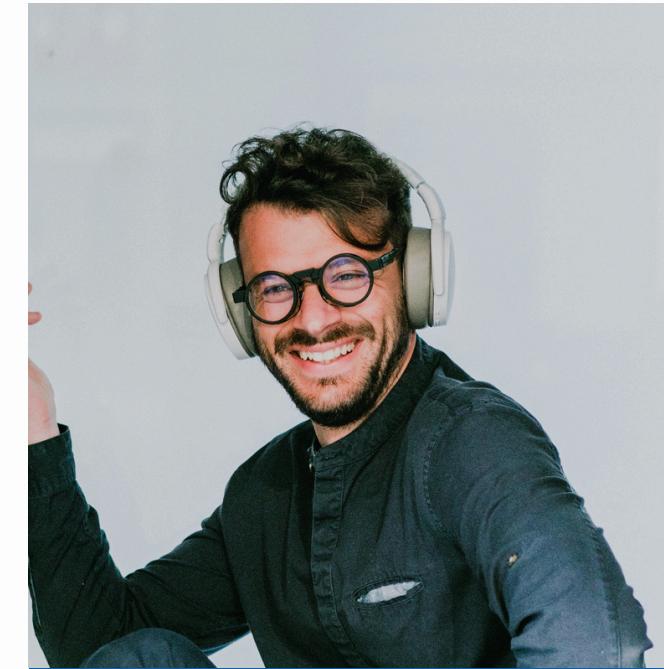
# Our Team



**Maharaka Fadhilah**



**R. Aisha Syauqi**



**Tri Yoga**



Proyek Akhir PSD

# THANK YOU!

