UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Criptare si Decriptare RSA

Structura Sistemelor de Calcul

Nume: Ratiu Iulia-Miruna

Îndrumător: Andrei Mihai Sopterean

Grupa: 30238

Facultatea de Automatică
si Calculatoare

15 Ianuarie 2024

Cuprins

1 Rezumat

Criptarea este fundamentala in securizarea datelor in lumea reala, protejand informatiile sensibile in diverse domenii.

Decriptarea este procesul complementar criptarii si este esentiala pentru accesarea informatiilor protejate in lumea reala.

In acest proiect, am implementat algoritmii de criptare si decriptare RSA, utilizand o abordare secventiala pentru procesarea caracterelor. Implementarea este structurata in trei module esentiale, fiecare procesand cate un caracter pe rand si abordand un aspect cheie al procesului RSA. Primul modul este responsabil pentru calcularea cheii private, utilizand doua numere prime p si q, impreuna cu cheia publica. Acest proces este crucial, deoarece cheia privata este utilizata in decriptare si trebuie calculata conform unor proprietati matematice stricte pentru a asigura corectitudinea algoritmului.

Al doilea modul este dedicat criptarii mesajelor. In acest proces, fiecare caracter al mesajului este transformat intr-un text cifrat utilizand cheia publica, care este disponibila oricarui utilizator. Criptarea garanteaza ca mesajul nu poate fi citit decat de posesorul cheii private, oferind astfel securitate pentru transferurile de date.

Ultimul modul este responsabil de procesul de decriptare, in cadrul caruia textul cifrat este transformat inapoi in mesajul original, utilizand cheia privata. Aceasta etapa demonstreaza capacitatea algoritmului de a asigura confidentialitatea datelor, chiar si in prezenta unei chei publice vizibile.

Pentru a implementa operatiile matematice complexe implicate in RSA, s-au utilizat doua metode distincte: ridicarea la putere simpla, utilizata pentru gestionarea numerelor mai mici, si exponentierea modulara, care permite lucrul eficient cu numere de dimensiuni mari.

Testarea implementarii s-a realizat prin trei testbench-uri dedicate, fiecare axandu-se pe verificarea procesului de criptare, decriptare si generarea cheii private. Aceste testbench-uri au confirmat corectitudinea algoritmilor si au demonstrat functionarea acestora intr-un mediu simulat.

2 Introducere

In era digitala moderna, securitatea informatiei este o provocare majora, avand in vedere volumul imens de date transmise zilnic prin retele nesigure. De la comunicatii private la tranzactii financiare si aplicatii guvernamentale, protejarea informatiilor sensibile este esentiala pentru prevenirea accesului neautorizat si a atacurilor cibernetice. Criptografia joaca un rol central in acest context, oferind metode eficiente de protejare a datelor. Algoritmul RSA, dezvoltat in 1977 de Ron Rivest, Adi Shamir si Leonard Adleman la MIT, este unul dintre cei mai importanti si utilizati algoritmi de criptare asimetrica. RSA este remarcabil prin faptul ca a fost primul algoritm care permite atat criptarea, cat si semnatura electronica, bazandu-se pe proprietati matematice solide si pe dificultatea factorizarii numerelor mari[1][2][6].

RSA functioneaza pe baza unei perechi de chei – o cheie publica si una privata – legate matematic intre ele. Cheia publica este utilizata pentru criptarea mesajelor, in timp ce cheia privata este folosita pentru decriptare. Aceasta separare a cheilor ofera un avantaj major fata de metodele de criptare simetrica, permitand schimbul de informatii securizate fara a necesita un canal sigur pentru distribuirea cheilor. Puterea RSA deriva din dificultatea problemei de factorizare a unui numar mare n, obtinut ca produs al doua numere prime mari p si q. Algoritmul este

considerat sigur datorita complexitatii exponentiale a algoritmilor cunoscuti pentru factorizarea numerelor mari [2][3].

RSA este utilizat pe scara larga in aplicatii practice, inclusiv in protocoalele HTTPS, semnaturile electronice si schimbul securizat de chei. Cu toate acestea, datorita costurilor computationale ridicate, RSA este folosit adesea doar pentru criptarea initiala a unei chei simetrice (cum ar fi AES), care este ulterior utilizata pentru criptarea efectiva a datelor. Aceasta combinatie imbina securitatea RSA cu viteza algoritmilor simetrici [1][3].

Un aspect esential al proiectului de fata este analiza si implementarea algoritmului RSA in VHDL, punand accent pe performanta si corectitudine. Pentru a facilita procesul de criptare si decriptare, s-au utilizat doua metode distincte: ridicarea la putere simpla pentru numere mici si exponentierea modulara pentru gestionarea numerelor mari. Implementarea a fost validata prin testbench-uri dedicate, demonstrand functionalitatea algoritmului atat pentru criptare, cat si pentru decriptare.

Pe langa implementarea RSA, proiectul discuta aspecte legate de securitatea algoritmului si vulnerabilitatile posibile, inclusiv atacuri cu text cifrat ales si riscurile asociate unei configurari incorecte. De asemenea, sunt prezentate optimizari pentru imbunatatirea performantei, cum ar fi utilizarea exponentului public mic si aplicarea teoremei chinezesti a resturilor in procesul de decriptare [5].

3 Referinte

1. Rivest, R. L., Shamir, A., Adleman, L., "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, 1978.
2. "RSA Cryptography Specifications (PKCS #1 v2.2)," RFC 8017, 2016. Link oficial RFC 8017.
3. "Algoritmul de criptografie RSA," Securitatea Informatiei. Link articol.
4. Schneier, B., Applied Cryptography: Protocols, Algorithms, and Source Code in C, Wiley, 1996.
5. Stallings, W., Cryptography and Network Security: Principles and Practice, Prentice Hall, 2014.
6. "MIT lab for Computer Science" Link.

4 Proiectare și implementare

4.1 Generarea Cheii Private

4.1.1 Fundamentare teoretica

Modulul `private_key_RSA` calculeaza cheia privata d, utilizand numerele prime p si q si cheia publica e. Cheia privata este necesara pentru procesul de decriptare.

4.1.2 Implementare

- **Inputuri:**
  - `p_in`, `q_in`: Numere prime de 16 biti.
  - `public_key`: Cheia publica e.
- **Outputuri:**
  - `private_key`: Cheia privata d.

- `n`: Produsul $n = p \cdot q$.
- `both_keys`: Semnal care indica finalizarea calculului cheilor.

Metoda:

- Calculeaza $\Phi(n) = (p-1)(q-1)$.
- Determina d folosind algoritmul lui Euclid extins, astfel incat $e \cdot d \equiv 1 \pmod{\Phi(n)}$.

4.1.3 Codul VHDL

Listing 1: Codul pentru Modulul `private_key_RSA`

```
entity private_key_RSA is
    port (
        clk          : in  std_logic;
        en           : in  std_logic;
        p_in         : in  std_logic_vector(15 downto 0);
        q_in         : in  std_logic_vector(15 downto 0);
        public_key   : in  std_logic_vector(31 downto 0);
        private_key  : out std_logic_vector(31 downto 0);
        n: out std_logic_vector(31 downto 0);
        both_keys: out std_logic
    );
end private_key_RSA;

architecture Behavioral of private_key_RSA is
    signal mul_m   : unsigned(31 downto 0);
    signal acc3    : unsigned(31 downto 0) ;
    signal index   : std_logic_vector(31 downto 0) := x"00000001";
    signal found   : std_logic := '0';
    begin
        mul_m <=  (unsigned(p_in) - 1) * (unsigned(q_in) - 1);
        process(clk)
        begin
            if found = '1' then
                    found <= '1';
            else
                    acc3 <= (unsigned(index) * unsigned(public_key)) mod mul_m;
                    if rising_edge(clk)and en='1' then
                        if acc3 = 1 then
                            private_key <= index;
                            n<=std_logic_vector(unsigned(p_in)*unsigned(q_in));
                            found <= '1';
                            both_keys<='1';
                        else
                            index <= index + 1;
                        end if;
                    end if;
            end if;
end process;
end Behavioral;
```

4.2 Criptarea

4.2.1 Fundamentare teoretica

Modulul `rsa_encrypt` cripteaza un mesaj (reprezentat ca un caracter ASCII) utilizand cheia publica e si modulul n.

4.2.2 Implementare

- **Inputuri:**
  - `character_in`: Caracterul ASCII de criptat (8 biti).
  - `public_key_e`, `public_key_n`: Cheia publica e si modulul n.
- **Outputuri:**
  - `ciphertext`: Mesajul criptat.
  - `done`: Semnal de finalizare a criptarii.

Metoda:

- Transforma caracterul in format numeric.
- Utilizeaza formula $c = m^e \mod n$ pentru a genera textul cifrat.

4.2.3 Codul VHDL

Listing 2: Codul pentru Modulul `rsa_encrypt`

```
entity rsa_encrypt is
    port (
        clk           : in  std_logic;
        en            : in  std_logic;
        character_in  : in  std_logic_vector(7 downto 0);
        public_key_e  : in  std_logic_vector(31 downto 0);
        public_key_n  : in  std_logic_vector(31 downto 0);
        ciphertext    : out std_logic_vector(31 downto 0);
        done          : out std_logic
    );
end rsa_encrypt;

architecture Behavioral of rsa_encrypt is
    signal base        : unsigned(31 downto 0);
    signal exp         : unsigned(31 downto 0);
    signal modulus     : unsigned(31 downto 0);
    signal result      : unsigned(31 downto 0) := x"00000001";
    signal temp_base   : unsigned(31 downto 0);
    signal temp_exp    : unsigned(31 downto 0);
    signal calc_done   : std_logic := '0';
    signal active      : std_logic := '0';
begin
    base <= x"000000" & unsigned(character_in);
    exp <= unsigned(public_key_e);
    modulus <= unsigned(public_key_n);

    process(clk)
```

```
    begin
        if rising_edge(clk) then
            if en = '1' and active = '0' then
                -- Initializare pentru exponentiere modulara rapida
                temp_base <= base mod modulus; -- Calcul baza initiala
                temp_exp <= exp;
                result <= x"00000001"; -- Resetare rezultat la 1
                calc_done <= '0';
                active <= '1'; -- Activare calcul
            elsif active = '1' then
                if temp_exp /= 0 then
                    if temp_exp(0) = '1' then
                        result <= (result * temp_base) mod modulus;
                    end if;
                    -- Ridica la patrat baza
                    temp_base <= (temp_base * temp_base) mod modulus;
                    -- Shift dreapta exponenta
                    temp_exp <= "0" & temp_exp(31 downto 1);
                else
                    -- Calcul finalizat
                    calc_done <= '1';
                    active <= '0';
                end if;
            end if;

            -- Iesire rezultat
            if calc_done = '1' then
                ciphertext <= std_logic_vector(result); -- Rezultatul
                done <= '1';
            else
                done <= '0';
            end if;
        end if;
    end process;

end Behavioral;
```

4.3 Decriptarea

4.3.1 Fundamentare teoretica

Modulul `rsa_decrypt` recupereaza mesajul original din textul cifrat, utilizand cheia privata d si modulul n.

4.3.2 Implementare

- **Inputuri:**
  - `ciphertext_in`: Mesajul criptat.
  - `private_key_d`, `public_key_n`: Cheia privata d si modulul n.

- **Outputuri:**
  - `character_out`: Caracterul ASCII decriptat.
  - `done`: Semnal de finalizare a decriptarii.

Metoda:

- Utilizeaza formula $m = c^d \mod n$ pentru a reconstrui mesajul original.

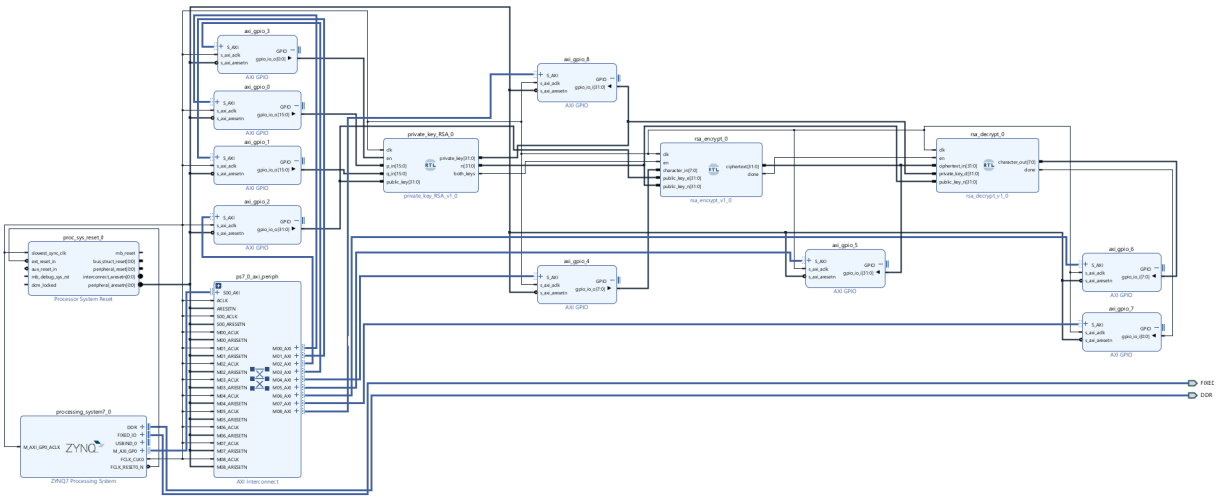
4.3.3 Codul VHDL

Listing 3: Codul pentru Modulul `rsa_decrypt`

```
entity rsa_decrypt is
    port (
        clk             : in  std_logic;
        en              : in  std_logic;
        ciphertext_in   : in  std_logic_vector(31 downto 0);
        private_key_d   : in  std_logic_vector(31 downto 0);
        public_key_n    : in  std_logic_vector(31 downto 0);
        character_out   : out std_logic_vector(7 downto 0);
        done            : out std_logic
    );
end rsa_decrypt;

architecture Behavioral of rsa_decrypt is
    signal base         : unsigned(31 downto 0);
    signal exp          : unsigned(31 downto 0);
    signal modulus      : unsigned(31 downto 0);
    signal result       : unsigned(31 downto 0) := x"00000001";
    signal temp_base    : unsigned(31 downto 0);
    signal temp_exp     : unsigned(31 downto 0);
    signal calc_done    : std_logic := '0';
    signal active       : std_logic := '0';
begin

    base <= unsigned(ciphertext_in);
    exp <= unsigned(private_key_d);
    modulus <= unsigned(public_key_n);

    process(clk)
    begin
        if rising_edge(clk) then
            if en = '1' and active = '0' then
                temp_base <= base mod modulus;
                temp_exp <= exp;
                result <= x"00000001";
                calc_done <= '0';
                active <= '1';
            elsif active = '1' then
                if temp_exp /= 0 then
                    if temp_exp(0) = '1' then
```

```
                    result <= (result * temp_base) mod modulus;
                end if;
                temp_base <= (temp_base * temp_base) mod modulus;
                temp_exp <= "0"&temp_exp(31 downto 1);
            else
                calc_done <= '1';
                active <= '0';
            end if;
        end if;

        if calc_done = '1' then
            character_out <= std_logic_vector(result(7 downto 0));
            done <= '1';
        else
            done <= '0';
        end if;
    end if;
  end process;

end Behavioral;
```

4.4 Schema



4.5 Cod Vitis

Listing 4: Codul pentru Modulul `vitis`

```
#include "xil_printf.h"
#include <stdio.h>
#include "platform.h"
#include "xparameters.h"
#include "sleep.h"
#include <stdlib.h>
```

```
#include "xgpio.h"

#define MAX_BUFFER_SIZE 100


void print_binary(uint32_t value, int bits) {
    for (int i = bits - 1; i >= 0; i--) {
        xil_printf("%d", (value >> i) & 1);
    }
}

int main() {
    XGpio pregatire_p_in, pregatire_q_in, pregatire_en_generare_prk, pregatire_pu
    int encrypt_character, decrypt_character;
    int en, done;
    uint16_t p_in, q_in;
    uint32_t public_key, private_key;
    int original_character; // Transmis pe 7 biti
    char buffer[MAX_BUFFER_SIZE];
    int encrypted_buffer[MAX_BUFFER_SIZE];
    int decrypted_buffer[MAX_BUFFER_SIZE];
    int index = 0;

    // Initializare GPIO
    XGpio_Initialize(&pregatire_p_in, XPAR_AXI_GPIO_0_BASEADDR);
    XGpio_Initialize(&pregatire_q_in, XPAR_AXI_GPIO_1_BASEADDR);
    XGpio_Initialize(&pregatire_puk, XPAR_AXI_GPIO_2_BASEADDR);
    XGpio_Initialize(&pregatire_en_generare_prk, XPAR_AXI_GPIO_3_BASEADDR);
    XGpio_Initialize(&pregatire_character, XPAR_AXI_GPIO_4_BASEADDR);
    XGpio_Initialize(&ciphertext, XPAR_AXI_GPIO_5_BASEADDR);
    XGpio_Initialize(&deciphertext, XPAR_AXI_GPIO_6_BASEADDR);
    XGpio_Initialize(&process_done, XPAR_AXI_GPIO_7_BASEADDR);
    XGpio_Initialize(&generated_private_key, XPAR_AXI_GPIO_8_BASEADDR);

    // Configurare directie GPIO
    XGpio_SetDataDirection(&pregatire_p_in, 1, 0x0);
    XGpio_SetDataDirection(&pregatire_q_in, 1, 0x0);
    XGpio_SetDataDirection(&pregatire_puk, 1, 0x0);
    XGpio_SetDataDirection(&pregatire_en_generare_prk, 1, 0x0);
    XGpio_SetDataDirection(&pregatire_character, 1, 0x0);
    XGpio_SetDataDirection(&ciphertext, 1, 0xF);
    XGpio_SetDataDirection(&deciphertext, 1, 0xF);
    XGpio_SetDataDirection(&process_done, 1, 0xF);
    XGpio_SetDataDirection(&generated_private_key, 1, 0xF);

    init_platform();

    xil_printf("Introdu caractere (maxim %d):\r\n", MAX_BUFFER_SIZE - 1);
```

```
while (index < MAX_BUFFER_SIZE - 1) {
    char c = inbyte();
    original_character = c & 0x7F;

    xil_printf("Original (8 bti binar): ");
    print_binary(original_character, 8);
    xil_printf(" ");
    xil_printf("Hexa: %02X\r\n", original_character);

    if (c == '\r') {
        char next = inbyte();
        if (next == '\n') break;
    } else if (c == '\n') {
        break;
    }


    XGpio_DiscreteWrite(&pregatire_p_in, 1, 0b0000000000000111);
    XGpio_DiscreteWrite(&pregatire_q_in, 1, 0b0000000000001011);
    XGpio_DiscreteWrite(&pregatire_en_generare_prk, 1, 1);
XGpio_DiscreteWrite(&pregatire_puk, 1, 0b00000000000000000000000000000111);

    XGpio_DiscreteWrite(&pregatire_character, 1, original_character);

    // Citire rezultate criptare si decriptare
    encrypt_character = XGpio_DiscreteRead(&ciphertext, 1) & 0xFFFFFFFF;
    decrypt_character = XGpio_DiscreteRead(&deciphertext, 1) & 0xFF;

    done = XGpio_DiscreteRead(&process_done, 1);

    // Salvare in buffere
    buffer[index] = original_character;
    encrypted_buffer[index] = encrypt_character;
    decrypted_buffer[index] = decrypt_character;

    // Afisare litera curenta in format binar si hexazecimal
    xil_printf("Criptata (32 biti binar): ");
    print_binary(encrypt_character, 32);
    xil_printf(" ");
    xil_printf("Hexa: %08X\r\n", encrypt_character);

    xil_printf("Decriptata (8 biti binar): ");
    print_binary(decrypt_character, 8);
    xil_printf(" ");
    xil_printf("Hexa: %02X\r\n", decrypt_character);

    index++;
```

```
        usleep(200000);
    }

    buffer[index] = '\0';

    // Citirea cheii private
    private_key = XGpio_DiscreteRead(&generated_private_key, 1);

    // Afisare rezultate finale
    xil_printf("\r\nSir complet citit: ");
    for (int i = 0; i < index; i++) {
        xil_printf("%c", buffer[i]);
    }
    xil_printf("\r\n");

    xil_printf("Sir criptat (32 biti binar):\r\n");
    for (int i = 0; i < index; i++) {
        print_binary(encrypted_buffer[i], 32);
        xil_printf(" ");
    }
    xil_printf("\r\nSir criptat (hexazecimal):\r\n");
    for (int i = 0; i < index; i++) {
        xil_printf("%08X ", encrypted_buffer[i]);
    }
    xil_printf("\r\n");

    xil_printf("Sir decriptat (8 biti binar):\r\n");
    for (int i = 0; i < index; i++) {
        print_binary(decrypted_buffer[i], 8);
        xil_printf(" ");
    }
    xil_printf("\r\nSir decriptat (hexazecimal):\r\n");
    for (int i = 0; i < index; i++) {
        xil_printf("%02X ", decrypted_buffer[i]);
    }
    xil_printf("\r\n");

    // Afisare cheia privata
    xil_printf("Cheia privata (binar): ");
    print_binary(private_key, 32);
    xil_printf(" ");
    xil_printf("Cheia privata (hexazecimal): %08X\r\n", private_key);

    xil_printf("Citirea s-a terminat.\r\n");

    cleanup_platform();
    return 0;
}
```

5 Rezultate experimentale

5.1 Testbench pentru Modulul de Criptare

5.1.1 Descriere

Testbench-ul pentru modulul `rsa_encrypt` este conceput pentru a valida functionalitatea acestuia. In cadrul testarii, sunt furnizate intrari specifice (un caracter ASCII, cheie publica e, modul n) si se verifica daca textul cifrat este corect generat conform formulei RSA: $c = m^e \mod n$.

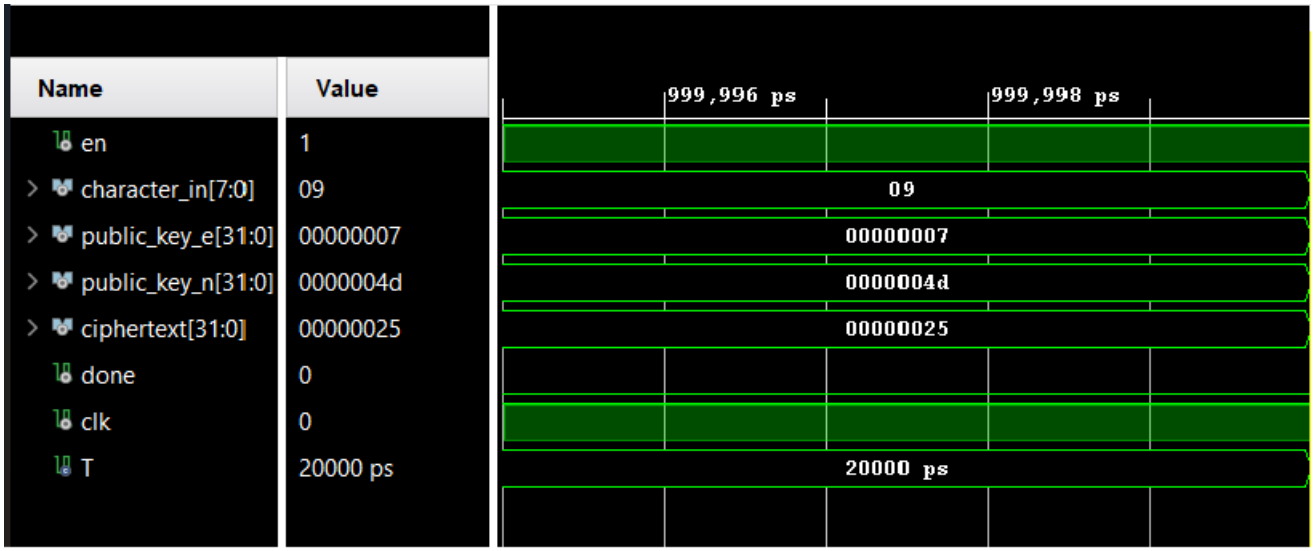
5.1.2 Calculul Manual

- Mesajul de intrare: $m = 9$ (caracter ASCII).
- Cheia publica: $e = 7$.
- Modulus: $n = 77$.
- Calculul textului cifrat:

$$c = m^e \mod n = 9^7 \mod 77 = 37. \tag{1}$$

5.1.3 Rezultate Asteptate

- Textul cifrat c generat de modul este consistent cu valoarea teoretica $c = 37$.
- Semnalul `done` este activat la finalizarea procesului de criptare.



5.2 Testbench pentru Modulul de Decriptare

5.2.1 Descriere

Testbench-ul pentru `rsa_decrypt` valideaza recuperarea corecta a mesajului original m dintr-un text cifrat c. Acesta testeaza formula $m = c^d \mod n$, folosind cheia privata d si modulul n.

5.2.2 Calculul Manual

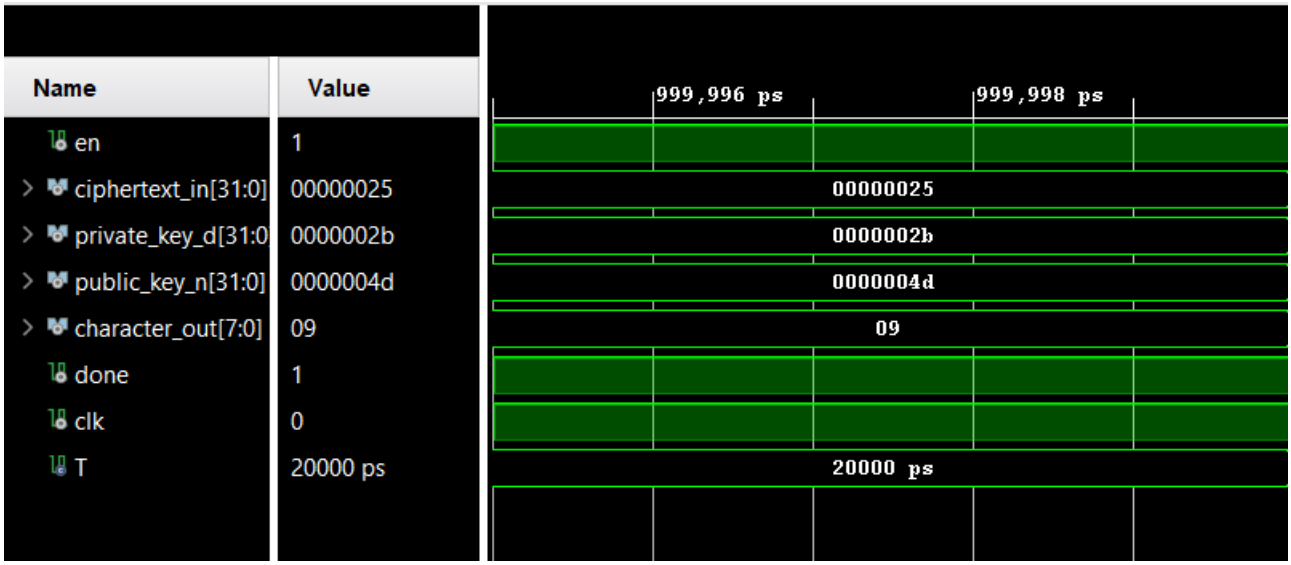
- Textul cifrat: $c = 37$.

- Cheia privata: $d = 43$.
- Modulus: $n = 77$.
- Calculul mesajului original:

$$m = c^d \mod n = 37^{43} \mod 77 = 9. \tag{2}$$

5.2.3 Rezultate Asteptate

- Mesajul m recuperat este identic cu valoarea originala utilizata in procesul de criptare: $m = 9$.
- Semnalul `done` este activat dupa finalizarea procesului.



5.3 Testbench pentru Generarea Cheii Private

5.3.1 Descriere

Testbench-ul pentru `private_key_RSA` valideaza generarea corecta a cheii private d.

5.3.2 Calculul Manual

- Numere prime: $p = 7, q = 11$.
- Calculul $\Phi(n) = (p-1)(q-1) = 60$.
- Cheia publica: $e = 7$.
- Calculul cheii private d:

$$d \text{ este unicul intreg pentru care } (e \cdot d) \mod \Phi(n) = 1. \tag{3}$$

- Rezultatul este $d = 43$.

5.3.3 Rezultate Asteptate

- Cheia privata generata $d = 43$ este consistenta cu calculul manual.
- Semnalul `both_keys` este activat la finalizarea generarii.

6 Observatii generale

- **Exponentiere Modulara Rapida:** Toate modulele folosesc aceasta tehnica pentru a gestiona eficient numere mari.
- **Optimizare:** Semnalele de control precum `done` si `both_keys` faciliteaza utilizarea in aplicatii practice.
- **Validare:** Modulele au fost testate folosind testbench-uri dedicate, confirmand functionalitatea acestora.

7 Concluzii

7.1 Implementarea Practica a RSA in VHDL

Proiectul demonstreaza implementarea cu succes a algoritmului RSA utilizand limbajul VHDL. Aceasta include generarea cheii private, criptarea mesajelor si decriptarea lor, oferind astfel un flux complet pentru procesarea datelor in sisteme digitale.

7.2 Utilizarea Exponentierii Modulare Rapide

Toate modulele dezvoltate folosesc tehnica de exponentiere modulara rapida, ceea ce asigura eficienta procesarii chiar si pentru numere mari. Aceasta metoda este cruciala pentru performanta sistemelor criptografice.

7.3 Validare si Testare

Testbench-urile dezvoltate au demonstrat corectitudinea implementarii prin comparatia rezultatelor generate cu calculele manuale teoretice. Aceasta abordare a confirmat ca modulele functioneaza conform specificatiilor RSA.

7.4 Limitari si Posibile Imbunatatiri

Desi implementarea este functionala, utilizarea unui FPGA ar putea beneficia de optimizari suplimentare, cum ar fi reducerea resurselor utilizate sau cresterea vitezei de procesare. De asemenea, daca reusesc sa fac integrarea proiectului pe Vitis, poate extinde aplicabilitatea.

7.5 Relevanta Proiectului

Algoritmul RSA este esential pentru securitatea digitala moderna, iar implementarea sa hardware reprezinta un pas important pentru integrarea sa in dispozitive embedded. Acest proiect poate servi ca punct de plecare pentru dezvoltari ulterioare in domeniul securitatii cibernetice.

7.6 Abordare Modulara

Proiectul a fost structurat modular, ceea ce permite reutilizarea componentelor, extinderea functionalitatilor si integrarea usoara cu alte sisteme.

8 Bibliografie

1. rsa-encryption algorithm
2. Online tool.
3. rsa-encryption algorithm.1.
4. RSA security solutions.
5. Git example.