L'Algorithme de Cryptographie de César :

Membre du groupe :

GOUMBRI Ratoudg Wende Landry

MBAYE Ndeye Maguette

DIBA Alhassane Souleymane

MWEEMBA Mukwamataba

I) Introduction:

L'algorithme de cryptographie de César, également connu sous le nom de chiffre de César, est l'un des plus anciens et des plus simples algorithmes de chiffrement. Il a été utilisé par Jules César pour sécuriser ses communications militaires. Cet algorithme appartient à la famille des chiffres de substitution, où chaque lettre du texte clair est remplacée par une lettre située un certain nombre de positions plus loin dans l'alphabet.

II) Principe du cryptage de César:

Le cryptage de César fonctionne en décalant chaque lettre d'un message d'un certain nombre de positions dans l'alphabet. Le nombre de positions de décalage est appelé la clé du cryptage.

1) Formule mathématique :

Pour une lettre donnée, la transformation peut être représentée par l'équation suivante :

$C = (P + K) \bmod 26$

Où :

- C est la lettre chiffrée.

- P est la lettre en clair (avec A = 0, B = 1, ..., Z = 25).

- K est la clé (le nombre de positions de décalage).

- mod 26 indique que l'opération se fait en modulo 26, c'est-à-dire que l'alphabet est cyclique.

2) Exemple de cryptage:

Supposons que la clé soit 3 et que le texte à chiffrer soit "HELLO".

1. H (7) + 3 = K (10)

2. E (4) + 3 = H (7)

3. L (11) + 3 = O (14)

4. L (11) + 3 = O (14)

5. O (14) + 3 = R (17)

Le texte chiffré est "KHOOR".

III) <u>Principe du décryptage de César:</u>

Pour déchiffrer un message chiffré avec le chiffre de César, il suffit d'inverser le processus en décalant chaque lettre du texte chiffré de la clé en arrière dans l'alphabet.

1) Formule mathématique :

Pour une lettre chiffrée, la transformation inverse est :

$P = (C - K + 26) \bmod 26$

L'ajout de 26 garantit que le résultat est toujours positif avant l'application du modulo.

2) Exemple de Principe du décryptage :

Reprenons notre exemple avec le texte chiffré "KHOOR" et la clé 3.

1. K (10) - 3 = H (7)
2. H (7) - 3 = E (4)
3. O (14) - 3 = L (11)
4. O (14) - 3 = L (11)
5. R (17) - 3 = O (14)

Le texte déchiffré est "HELLO".

IV) <u>Avantages et Inconvénients :</u>

1) Avantages :

a. Simplicité : L'algorithme est facile à comprendre et à implémenter.

b. Rapidité : Le chiffrement et le déchiffrement sont rapides.

2) Inconvénients :

a. Faible sécurité : Le chiffre de César est vulnérable aux attaques par force brute (il n'y a que 25 clés possibles) et aux analyses de fréquence.

b. Prédictibilité : Connaître une partie du texte chiffré et son équivalent en clair permet de déterminer la clé et de déchiffrer tout le message.

V) Conclusion

Le chiffrement de César, malgré sa simplicité et sa faible sécurité, représente une étape importante dans l'histoire de la cryptographie. Comprendre son fonctionnement aide à apprécier les avancées dans les méthodes de chiffrement modernes et à reconnaître l'importance de la sécurité des données dans notre monde numérique actuel.

Références

[Wikipedia: Caesar Cipher] (https://fr.wikipedia.org/wiki/Chiffre_de_C%C3%A9sar)

[Cryptography and Network Security Principles and Practices] (https://www.pearson.com/us/higher-education/program/Stallings-

Cryptography-and-Network-Security-Principles-and-Practice-7th-Edition/PGM240378.html) de William Stallings

Implémentation de l'algorithme de cryptographie César en VHDL

Introduction

L'algorithme de chiffrement César, un des plus anciens et des plus simples algorithmes de cryptographie, a été utilisé par Jules César pour sécuriser ses communications militaires. Son principe est de décaler chaque lettre d'un message d'un certain nombre de positions dans l'alphabet. Dans cet exposé, nous allons présenter une implémentation de cet algorithme en VHDL, un langage de description matériel.

Fonctionnement de l'algorithme

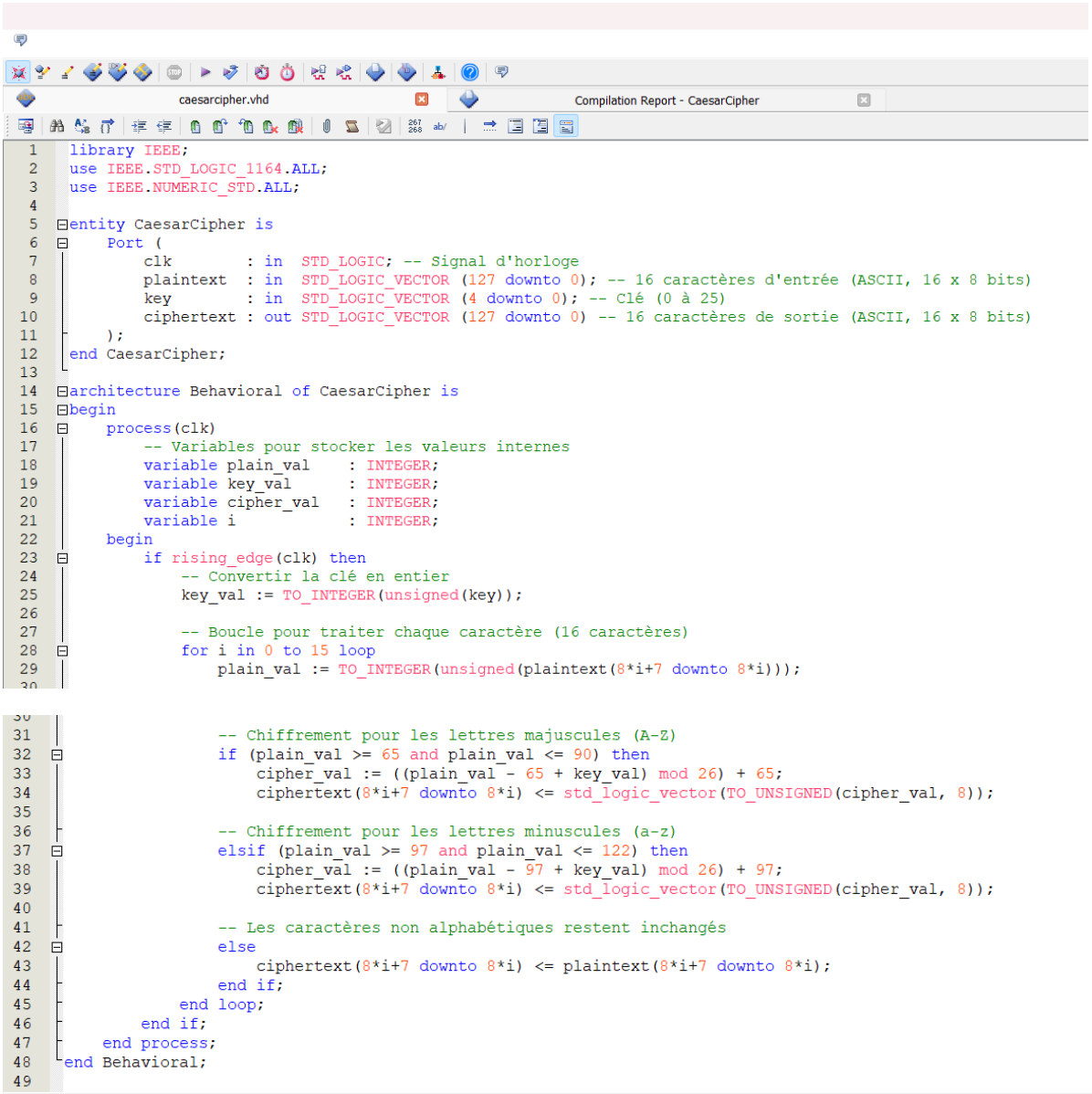
L'algorithme de chiffrement César fonctionne par les étapes suivantes :

1. Une clé de chiffrement est choisie, représentant le nombre de positions à décaler dans l'alphabet.
2. Chaque lettre du message est décalée vers la droite de ce nombre de positions.
3. Si le décalage dépasse 'Z', on revient au début de l'alphabet.
4. Les caractères non alphabétiques restent inchangés.

Implémentation en VHDL

L'implémentation en VHDL repose sur la description d'un circuit numérique qui prend en entrée un flux de données ASCII représentant le message et la clé de chiffrement, et produit en sortie le message chiffré.

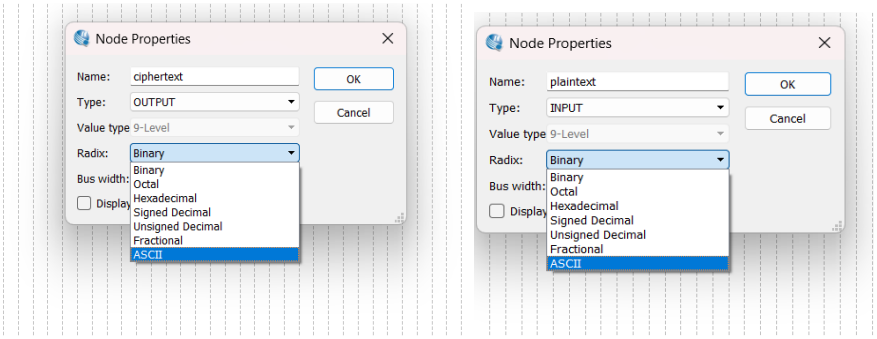
Code VHDL pour le cryptage en utilisant l'algorithme CESAR




```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity CaesarCipher is
    Port (
        clk        : in  STD_LOGIC; -- Signal d'horloge
        plaintext  : in  STD_LOGIC_VECTOR (127 downto 0); -- 16 caractères d'entrée (ASCII, 16 x 8 bits)
        key        : in  STD_LOGIC_VECTOR (4 downto 0); -- Clé (0 à 25)
        ciphertext : out STD_LOGIC_VECTOR (127 downto 0) -- 16 caractères de sortie (ASCII, 16 x 8 bits)
    );
end CaesarCipher;

architecture Behavioral of CaesarCipher is
begin
    process(clk)
        -- Variables pour stocker les valeurs internes
        variable plain_val    : INTEGER;
        variable key_val      : INTEGER;
        variable cipher_val   : INTEGER;
        variable i            : INTEGER;
    begin
        if rising_edge(clk) then
            -- Convertir la clé en entier
            key_val := TO_INTEGER(unsigned(key));

            -- Boucle pour traiter chaque caractère (16 caractères)
            for i in 0 to 15 loop
                plain_val := TO_INTEGER(unsigned(plaintext(8*i+7 downto 8*i)));

                -- Chiffrement pour les lettres majuscules (A-Z)
                if (plain_val >= 65 and plain_val <= 90) then
                    cipher_val := ((plain_val - 65 + key_val) mod 26) + 65;
                    ciphertext(8*i+7 downto 8*i) <= std_logic_vector(TO_UNSIGNED(cipher_val, 8));

                -- Chiffrement pour les lettres minuscules (a-z)
                elsif (plain_val >= 97 and plain_val <= 122) then
                    cipher_val := ((plain_val - 97 + key_val) mod 26) + 97;
                    ciphertext(8*i+7 downto 8*i) <= std_logic_vector(TO_UNSIGNED(cipher_val, 8));

                -- Les caractères non alphabétiques restent inchangés
                else
                    ciphertext(8*i+7 downto 8*i) <= plaintext(8*i+7 downto 8*i);
                end if;
            end loop;
        end if;
    end process;
end Behavioral;
```

L'exécutable se trouve dans le fichier text joint

NB : il faudra changer la sortie ascii dans la partie simulation



- **<u>Exemple de simulation 1 :</u>**

-Key = 2 ou 00010 en binaire sur 5 bits

-Notre text entrée (plaintext) = JE SUIS MALADIVE

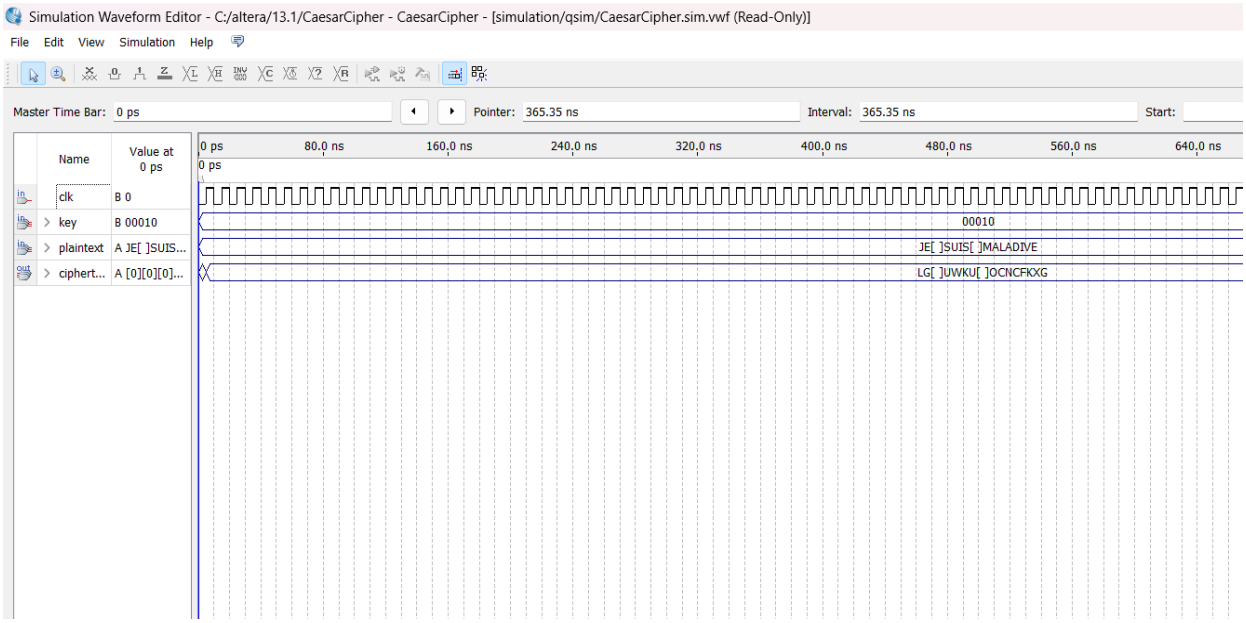


Figure : Résultat de la simulation pour le cryptage en VHDL

Nous obtenons comme sortie (ciphertext) : LG UWKU OCNCFKXG qui est notre message crypté

On constate que : - J devient L et E devient G, ...chaque caractère subit un décalage de 2 correspondant à la clé

- l'espace ne faisant pas parti de l'alphabet n'a pas été modifié

Notre cryptage de César a bien réussi.

➢ Exemple de simulation 2 :

- Key = 2 ou 00010 en binaire sur 5 bits

-Notre text entrée (plaintext) = Hello Al1n&@ Oui

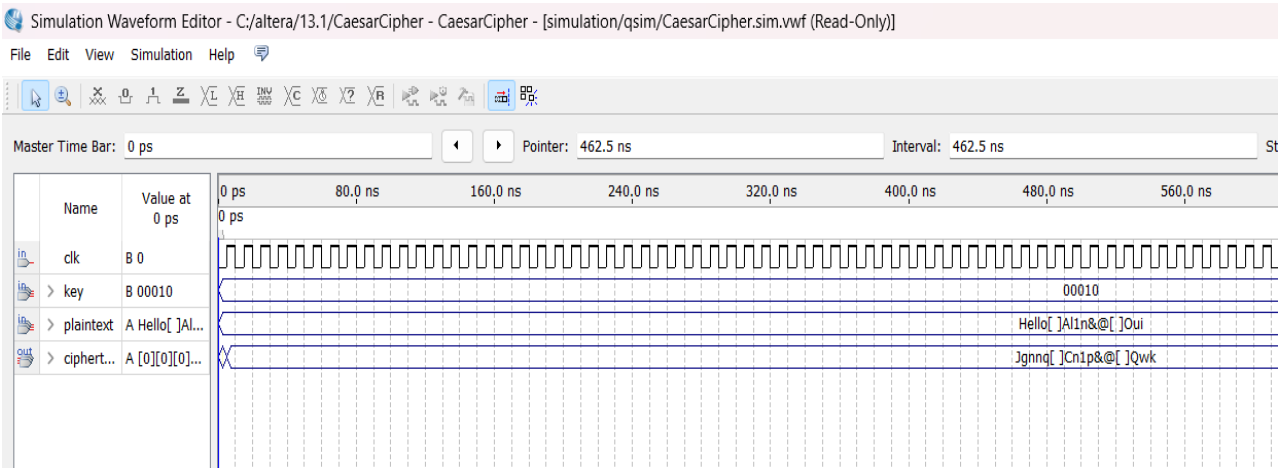


Figure : Résultat de la simulation pour le cryptage en VHDL

Nous obtenons un message crypté : Jgnnq Cn1p&@ Qwk

Cela nous atteste la crédibilité du cryptage car toutes les lettres alphabétiques (majuscules et minuscules) sont décalées et seulement les caractères non alphabétiques sont inchangés

Code VHDL pour le décryptage en utilisant l'algorithme CESAR

caesarcipher.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity CaesarCipher is
    Port (
        clk        : in  STD_LOGIC;
        ciphertext : in  STD_LOGIC_VECTOR (127 downto 0); -- 16 caractères d'entrée (ASCII, 16 x 8 bits)
        key        : in  STD_LOGIC_VECTOR (4 downto 0); -- Clé (0 à 25)
        decrypted  : out STD_LOGIC_VECTOR (127 downto 0) -- 16 caractères de sortie (ASCII, 16 x 8 bits)
    );
end CaesarCipher;

architecture Behavioral of CaesarCipher is

begin
    process(clk)
        variable cipher_val    : INTEGER;
        variable key_val       : INTEGER;
        variable decrypted_val : INTEGER;
        variable i             : INTEGER;
    begin
        if rising_edge(clk) then
            -- Convertir la clé en entier
            key_val := TO_INTEGER(unsigned(key));

            -- Boucle pour traiter chaque caractère (16 caractères)
            for i in 0 to 15 loop
                cipher_val := TO_INTEGER(unsigned(ciphertext(8*i+7 downto 8*i)));

                -- Déchiffrement pour les lettres majuscules (A-Z)
                if (cipher_val >= 65 and cipher_val <= 90) then
                    decrypted_val := ((cipher_val - 65 - key_val) mod 26) + 65;
                    decrypted(8*i+7 downto 8*i) <= std_logic_vector(TO_UNSIGNED(decrypted_val, 8));

                -- Déchiffrement pour les lettres minuscules (a-z)
                elsif (cipher_val >= 97 and cipher_val <= 122) then
                    decrypted_val := ((cipher_val - 97 - key_val)mod 26) + 97;
                    decrypted(8*i+7 downto 8*i) <= std_logic_vector(TO_UNSIGNED(decrypted_val, 8));

                -- Les caractères non alphabétiques restent inchangés
                else
                    decrypted(8*i+7 downto 8*i) <= ciphertext(8*i+7 downto 8*i);
                end if;
            end loop;
        end if;
    end process;
end Behavioral;
```

L'exécutable se trouve dans le fichier text joint

- **<u>Exemple de simulation 1 :</u>**

-Key = 2 ou 00010 en binaire sur 5 bits

-Notre text entrée (ciphertext) = LG UWKU OCNCFKXG (sortie 1 précédente cryptée)

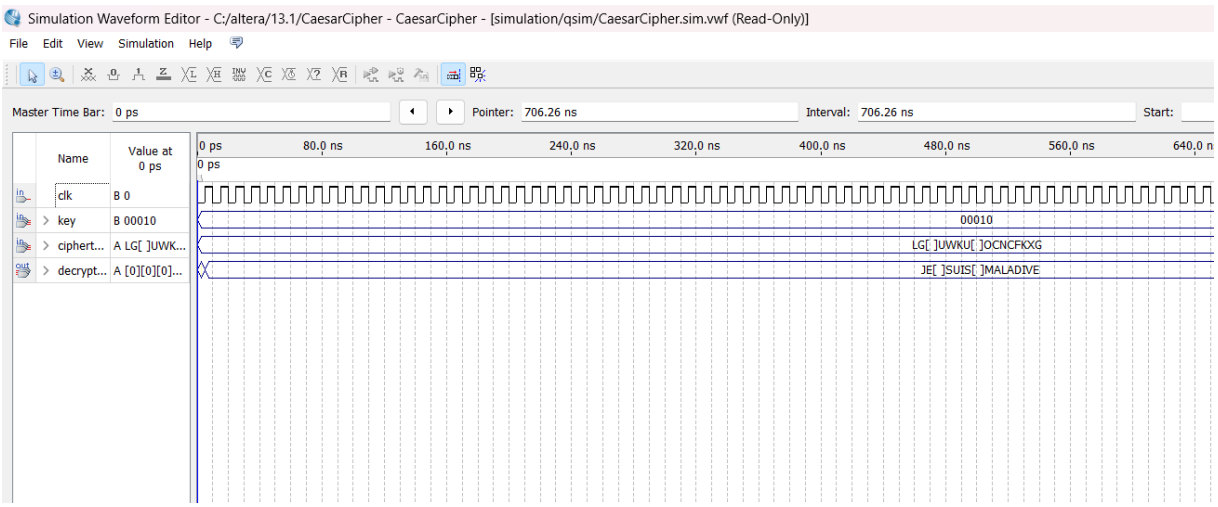


Figure : Résultat de la simulation pour le décryptage en VHDL

Nous obtenons comme sortie (decrypted) : JE SUIS MALADIVE qui est le message décrypté et correspond au message appliqué comme entrée de cryptage précédemment

➢ **Exemple de simulation 2 :**

- Key = 2 ou 00010 en binaire sur 5 bits

-Notre text entrée (ciphertext) = Jgnnq Cn1p&@ Qwk (sortie 2 précédente cryptée)

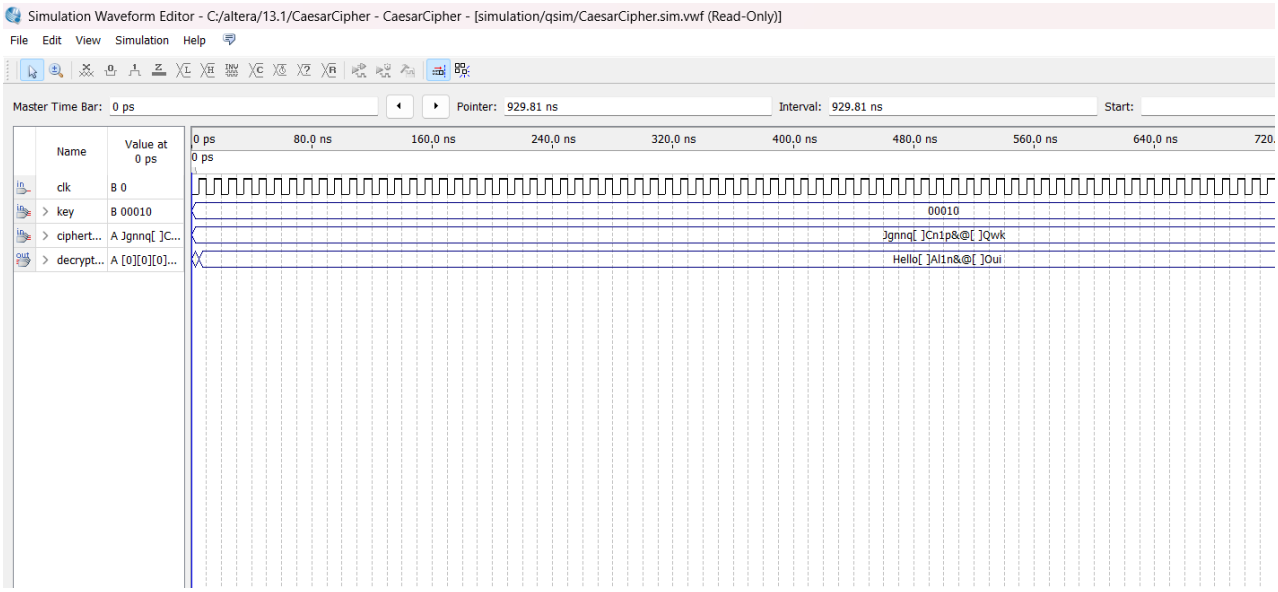


Figure : Résultat de la simulation pour le décryptage en VHDL

Nous obtenons comme sortie (decrypted) : Hello Al1n&@ Oui correspondant aussi à l'entrée appliquée précédemment pour le cryptage

Notre décryptage est alors réussi