



ENSAM CASA

# CHIFFREMENT PAR INVERSION

Projet VHDL

Réalisé par :

**ABOURIZK BILAL**

**AMALOU HIBA**

**BAKOUCHE LINA**

**BEN EL AYFAR MAHA**

## Table des matières

Introduction générale .....	2
Le chiffrement par inversion.....	3
Définition .....	3
Avantages et Inconvénients.....	3
Objectif du projet : .....	4
Chiffrement par inversion (Reverse Cipher) .....	5
Implémentation sur Quartus .....	5
Explication du code .....	5
Simulation.....	8
Résumé : .....	9
Déchiffrement par inversion (ReverseDecryption) .....	10
Implémentation sur Quartus :.....	10
Explication :.....	11
Simulation.....	12
Résumé.....	13
Conclusion .....	14

# Introduction générale

## Contexte de la Sécurité de l'Information

Avec l'augmentation de notre dépendance aux technologies numériques, la cybersécurité est devenue une priorité majeure. L'essor d'Internet et la numérisation de nombreux aspects de la vie quotidienne et professionnelle ont fait de la protection des données une préoccupation cruciale. Les cyberattaques peuvent cibler des individus, des entreprises et des infrastructures nationales, entraînant des conséquences potentiellement catastrophiques.

## Dangers des cyberattaques

**Perte de données sensibles :** Vol ou corruption d'informations personnelles, financières ou commerciales.

**Conséquences financières :** Fraudes, restauration de systèmes compromis, amendes réglementaires.

**Atteintes à la sécurité nationale :** Infrastructures critiques ciblées, menaçant la sécurité publique et nationale.

## Importance de la cyber sécurité

La cybersécurité est devenue une pierre angulaire de notre ère numérique. Son importance réside dans sa capacité à protéger nos données, nos systèmes informatiques et nos infrastructures essentielles contre les menaces en ligne. Dans un monde où la connectivité est omniprésente, la cybersécurité est essentielle pour garantir la confidentialité, l'intégrité et la disponibilité des informations. Les attaques cybernétiques peuvent entraîner des conséquences dévastatrices, allant de la violation de la vie privée à la perturbation des services vitaux tels que l'énergie, les soins de santé et les services financiers.

# Le chiffrement par inversion

## Définition

Le chiffrement par inversion, également connu sous le nom de renversement de caractères, est une technique de cryptographie dans laquelle les caractères d'un message sont réarrangés dans l'ordre inverse. Cela signifie que le premier caractère du message original devient le dernier caractère du message chiffré, le deuxième devient l'avant-dernier, et ainsi de suite. Cette méthode est l'une des plus simples formes de chiffrement et offre un niveau minimal d'obscurcissement des données

- **Fonctionnement du Chiffrement par Inversion :**

Le chiffrement par inversion est une méthode de cryptage qui consiste à inverser l'ordre des caractères dans un message pour le rendre illisible sans la clé de déchiffrement correspondante. Pour illustrer le fonctionnement du chiffrement par inversion, prenons un exemple simple :

Message original : "HELLO" => Message chiffré : "OLLEH"

- **Processus de cryptage :**

**Processus de chiffrement :** On prend le message original et on inverse l'ordre des caractères du message, et voilà le résultat est le message chiffré.

**Processus de déchiffrement :** Pour déchiffrer le message, on prend le message chiffré et on inverse à nouveau l'ordre des caractères. Cela nous donne le message original.

- **Applications du Chiffrement par Inversion :**

Le chiffrement par inversion, bien que simple et offrant une sécurité limitée, trouve des applications variées dans différents contextes. Voici quelques exemples plus détaillés où cette technique peut être utile :

- Obscurcissement Basique des Données : Puzzles et Jeux, Révisions Scolaires .
- Masquage Temporaires des Données : Journaux de Débogage.
- Prétraitement dans des Algorithmes Complexes : Encodage de Données.
- Techniques de Compression et d'Encodage : Encodage d'URL.

## Avantages et Inconvénients

- **Avantages :**

*Simplicité :* Algorithme très simple à comprendre et à implémenter.

*Rapidité :* Processus est rapide à exécuter, même pour des messages longs.

*Faible coût en ressources* : Nécessite peu de ressources matérielles ou computationnelles.

- **Inconvénients :**

*Sécurité très faible* : Ce chiffrement ne fournit qu'un minimum de sécurité. Il est trivial à déchiffrer sans aucune clé ou information supplémentaire.

*Pas de clé de chiffrement* : L'absence de clé rend ce chiffrement non sécurisé pour toute application nécessitant une confidentialité réelle.

*Limitation d'usage* : Utilisé principalement pour des besoins d'obscurcissement minimal ou comme composant dans des systèmes de chiffrement plus complexes.

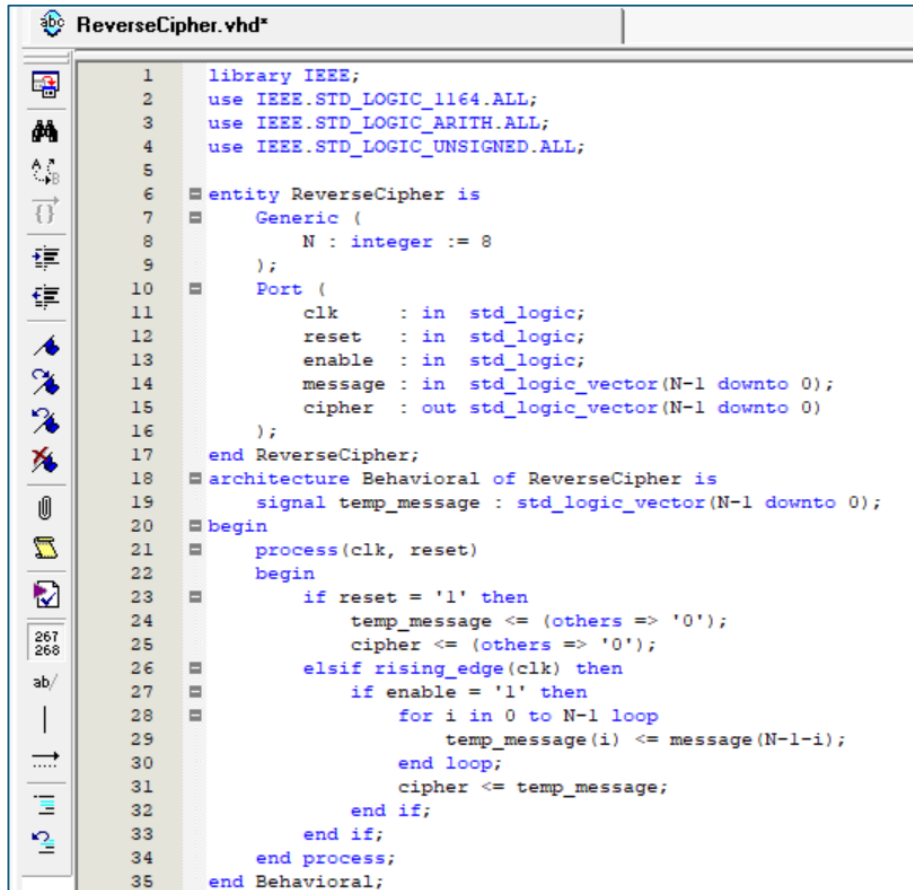
## Objectif du projet :

L'objectif de ce rapport est d'explorer le chiffrement par inversion en utilisant VHDL, en se concentrant sur ses principes de base, ses avantages et ses inconvénients, ainsi que sur la mise en œuvre pratique via codage et simulation sur Quartus. Le chiffrement par inversion repose sur des transformations mathématiques simples pour masquer les données, offrant l'avantage d'une faible complexité de calcul et une mise en œuvre matérielle efficace, mais présentant des faiblesses potentielles en termes de sécurité face à des attaques sophistiquées. J'ai développé et testé le code VHDL pour le chiffrement, expliquant chaque segment clé du code et démontrant comment il réalise l'inversion des données.

La simulation effectuée sur Quartus a confirmé le bon fonctionnement de l'algorithme, avec des résultats détaillés illustrant chaque étape du processus de chiffrement. De même, le déchiffrement a été codé et simulé, montrant comment les données peuvent être récupérées fidèlement à partir de leur forme chiffrée, validant ainsi l'efficacité et la robustesse de l'implémentation.

# Chiffrement par inversion (Reverse Cipher)

## Implémentation sur Quartus



```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity ReverseCipher is
7  generic (
8      N : integer := 8
9  );
10 port (
11     clk      : in  std_logic;
12     reset    : in  std_logic;
13     enable    : in  std_logic;
14     message   : in  std_logic_vector(N-1 downto 0);
15     cipher    : out std_logic_vector(N-1 downto 0)
16 );
17 end ReverseCipher;
18 architecture Behavioral of ReverseCipher is
19     signal temp_message : std_logic_vector(N-1 downto 0);
20 begin
21     process(clk, reset)
22     begin
23         if reset = '1' then
24             temp_message <= (others => '0');
25             cipher <= (others => '0');
26         elsif rising_edge(clk) then
27             if enable = '1' then
28                 for i in 0 to N-1 loop
29                     temp_message(i) <= message(N-1-i);
30                 end loop;
31                 cipher <= temp_message;
32             end if;
33         end if;
34     end process;
35 end Behavioral;
```

## Explication du code

### 1- Bibliothèques :

**Library IEEE :** Cette ligne inclut la bibliothèque IEEE, qui est une collection standard de paquets VHDL.

**use IEEE.STD\_LOGIC\_1164.ALL :** Cette ligne inclut les définitions des types standard `std_logic` et `std_logic_vector`, qui sont des types de données utilisés pour représenter des signaux logiques.

**use IEEE.STD\_LOGIC\_ARITH.ALL et use IEEE.STD\_LOGIC\_UNSIGNED.ALL :** opérations sur des vecteurs de bits non signés. Cependant, elles ne sont pas nécessaires dans ce code, car il n'y a pas d'opérations arithmétiques

## 2- Entité

```
entity ReverseCipher is
  Generic (
    N : integer := 8
  );
  Port (
    clk      : in  std_logic;
    reset    : in  std_logic;
    enable    : in  std_logic;
    message  : in  std_logic_vector(N-1 downto 0);
    cipher   : out std_logic_vector(N-1 downto 0)
  );
end ReverseCipher;
```

**Generic (N : integer := 8) :** Définit un paramètre générique N, qui représente la longueur du message. Par défaut, N est fixé à 8.

**Port :** Cette section définit les ports de l'entité :

- **clk** : Signal d'horloge (std\_logic) utilisé pour synchroniser les opérations.
- **reset** : Signal de réinitialisation (std\_logic) utilisé pour réinitialiser l'état du circuit.
- **enable** : Signal d'activation (std\_logic) pour activer le chiffrement.
- **message** : Entrée du message à chiffrer, représenté par un vecteur de std\_logic de longueur N.
- **cipher** : Sortie du message chiffré, également représentée par un vecteur de std\_logic de longueur N.

## 3- Architecture

### Déclaration des signaux internes :

**signal temp\_message : std\_logic\_vector(N-1 downto 0) :** Déclare un signal interne temp\_message qui est un vecteur de std\_logic de longueur N. Ce signal est utilisé pour stocker temporairement le message inversé.

```
architecture Behavioral of ReverseCipher is
  signal temp_message : std_logic_vector(N-1 downto 0);
```

## Processus principal ; process :

```
process(clk, reset)
begin
    if reset = '1' then
        temp_message <= (others => '0');
        cipher <= (others => '0');
    elsif rising_edge(clk) then
        if enable = '1' then
            for i in 0 to N-1 loop
                temp_message(i) <= message(N-1-i);
            end loop;
            cipher <= temp_message;
        end if;
    end if;
end process;
```

**process(clk, reset)** : Déclare un processus sensible aux signaux clk et reset. Un processus en VHDL est un bloc de code qui s'exécute séquentiellement.

**if reset = '1' then** : Si le signal de réinitialisation est actif (reset = '1'), alors :

**temp\_message <= (others => '0')** : Le signal temp\_message est réinitialisé à zéro.

**cipher <= (others => '0')** : Le signal de sortie cipher est réinitialisé à zéro.

**elsif rising\_edge(clk) then** : Si un front montant de l'horloge (clk) est détecté :

**if enable = '1' then** : Si le signal d'activation est actif (enable = '1'), alors :

**for i in 0 to N-1 loop** : Une boucle 'for' pour parcourir chaque bit du message.

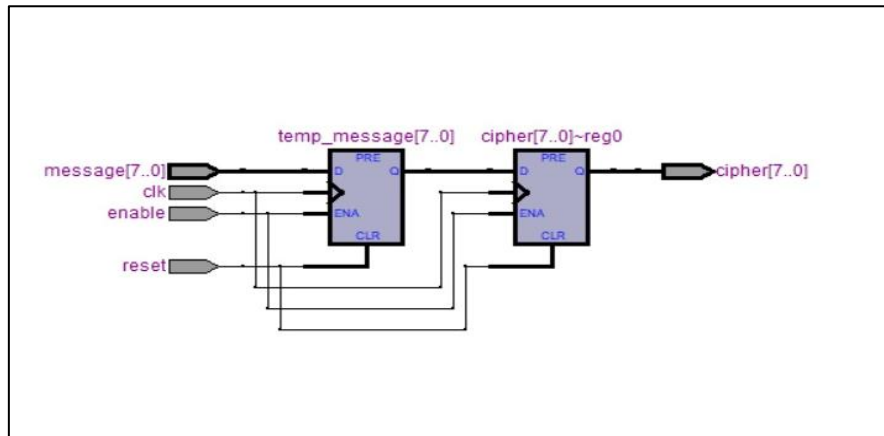
**temp\_message(i) <= message(N-1-i)** : L'élément i de temp\_message est assigné à l'élément N-1-i de message, ce qui inverse l'ordre des bits.

**cipher <= temp\_message** : Après la boucle, le signal de sortie cipher est mis à jour avec la valeur de temp\_message.



## Simulation

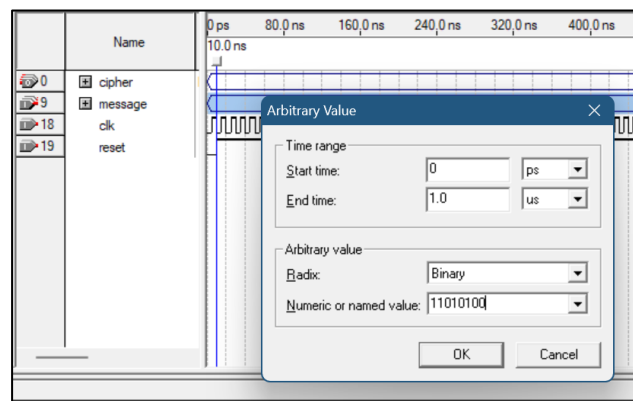
### Schéma équivalent :



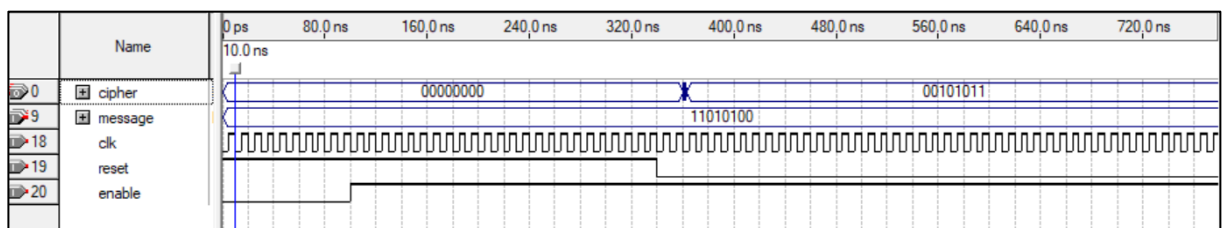
### Simulation :

**Etape 1 :** donner au message une valeur arbitraire

Dans l'exemple Message = '11010100'



**Etape 2 :** simuler sur quartus



L'image de simulation montre que le cipher est mis à jour avec la version inversée du message lorsque l'horloge clk monte et que enable est actif. Les étapes de simulations sont :

1. *Initialisation*: Au début, le signal `reset` est actif (haut), ce qui réinitialise la sortie `cipher` à zéro (`00000000`).

2. *Changement du message*: le `message` change et passe à `11010100`.

3. *Production du chiffrement* : Peu de temps après, le `cipher` affiche la valeur inversée du `message`. Si le `message` est `11010100`, après inversion, il devient `00101011`.

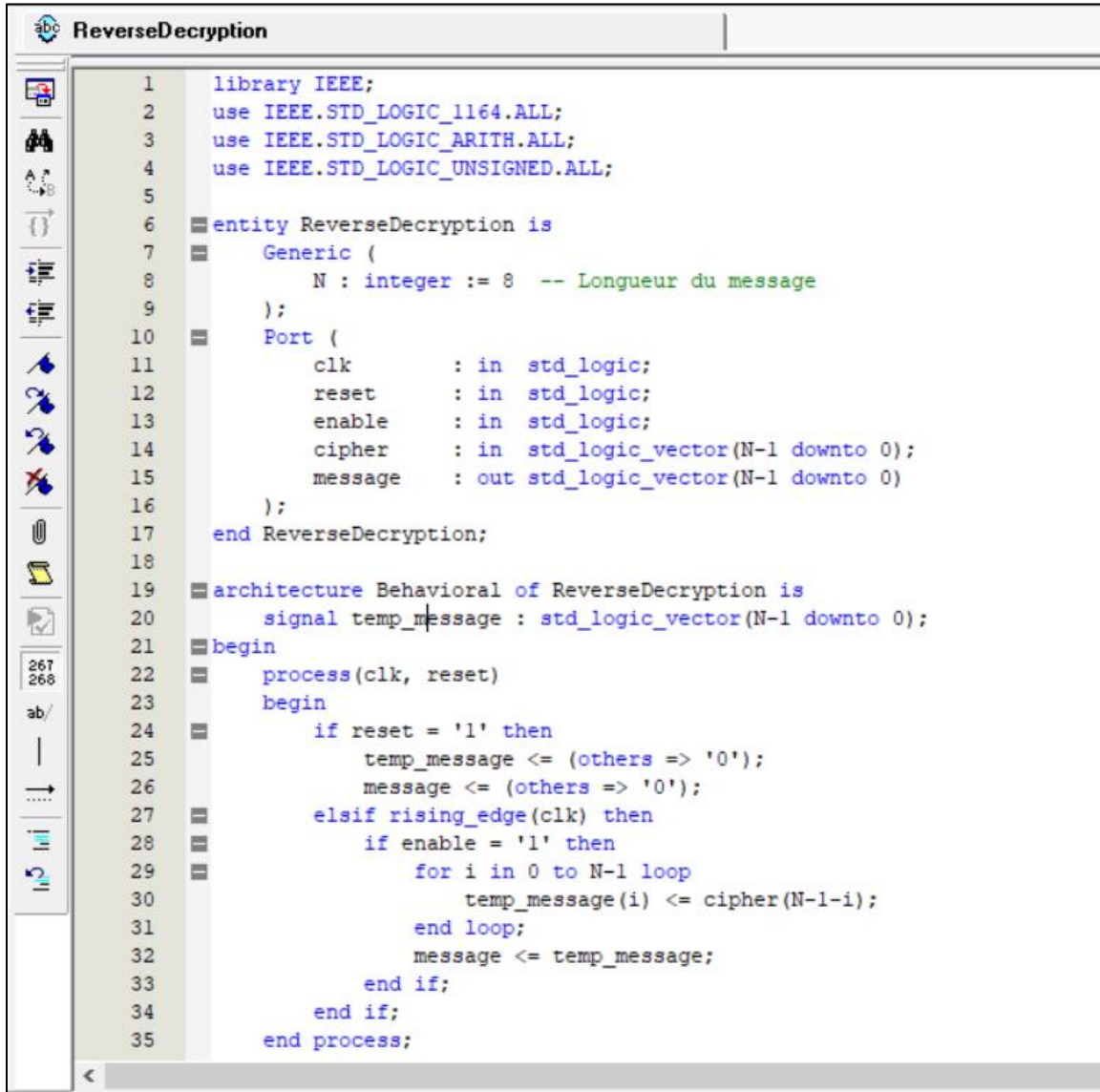
Cela montre que le module prend un message en entrée, inverse ses bits, et produit le résultat chiffré en sortie.

## Résumé :

Le code VHDL implémente un chiffrement par inversion de manière suivante : lorsque le signal de réinitialisation est actif, les sorties sont réinitialisées à zéro. À chaque front montant du signal d'horloge, si le signal d'activation est actif, le message d'entrée est parcouru, et chaque bit est assigné à la position inverse dans le signal temporaire temp\_message. Le chiffrement par inversion est une méthode extrêmement simple de réarrangement des caractères d'un message pour fournir un niveau minimal d'obscurcissement. Son implémentation en VHDL montre comment cette technique peut être appliquée dans des systèmes matériels, bien qu'elle ne soit pas appropriée pour des applications nécessitant une véritable sécurité cryptographique.

# Déchiffrement par inversion (ReverseDecryption)

Implémentation sur Quartus :



```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity ReverseDecryption is
7      Generic (
8          N : integer := 8  -- Longueur du message
9      );
10     Port (
11         clk      : in  std_logic;
12         reset    : in  std_logic;
13         enable    : in  std_logic;
14         cipher    : in  std_logic_vector(N-1 downto 0);
15         message   : out std_logic_vector(N-1 downto 0)
16     );
17 end ReverseDecryption;
18
19 architecture Behavioral of ReverseDecryption is
20     signal temp_message : std_logic_vector(N-1 downto 0);
21 begin
22     process(clk, reset)
23     begin
24         if reset = '1' then
25             temp_message <= (others => '0');
26             message <= (others => '0');
27         elsif rising_edge(clk) then
28             if enable = '1' then
29                 for i in 0 to N-1 loop
30                     temp_message(i) <= cipher(N-1-i);
31                 end loop;
32                 message <= temp_message;
33             end if;
34         end if;
35     end process;
```

## Explication :

Ce code VHDL décrit une entité appelée **ReverseDecryption** qui consiste à inverser les bits d'un vecteur de bits en utilisant une architecture comportementale un style en décrivant comportement d'une entité sans détailler l'implémentation structurelle.

### 1- Les bibliothèques utilisées :

- « STD\_LOGIC\_1164 » est utilisé pour définir les types de signaux logiques.
- « STD\_LOGIC\_ARITH » et « STD\_LOGIC\_UNSIGNED » fournissent des fonctions arithmétiques et de manipulation de bits.

### 2- L'entité :

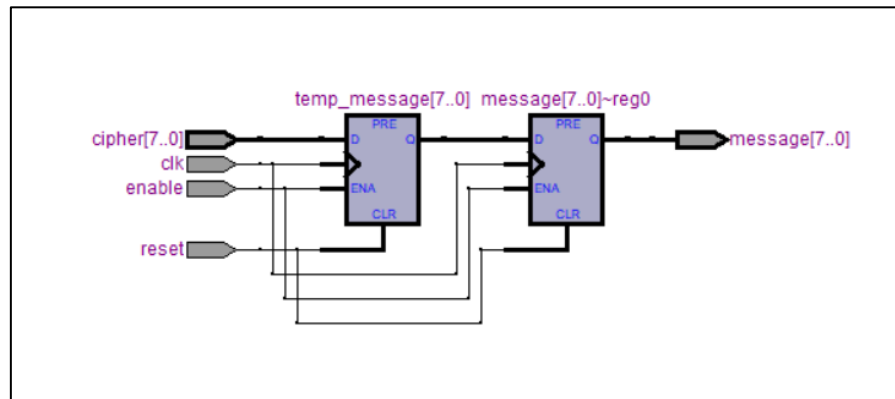
- 'N : integer := 8' Définit la longueur du vecteur 'cipher' et 'message' avec une valeur par défaut de 8.
- 'clk' : Signal d'horloge.
- 'Reset' : Signal de réinitialisation d'entrée.
- 'Enable' : Signal d'activation.
- 'cipher' : Vecteur d'entrée de type std\_logic\_vector de longueur N, représentant le message chiffré.
- 'message' : Vecteur de sortie de type std\_logic\_vector de longueur N, représentant le message déchiffré (inversé).

### 3- L'architecture :

- Déclaration de signal 'temp\_message' : Signal interne de type 'std\_logic\_vector(N-1 downto 0)' utilisé pour stocker temporairement le message inversé.
- Processus : car le processus est sensible aux changements des signaux 'clk' et 'reset'.
- Réinitialisation : Si le signal (reset = '1'), les vecteurs 'temp\_message' et 'message' sont remis à 0.
- Rising Edge de l'horloge : Si un front montant (rising\_edge) est détecté sur le signal d'horloge clk, le processus vérifie si le signal enable (actif = '1').
- Inversion des bits : Si enable est actif, une boucle for parcourt les indices de 0 à N-1. Pour chaque index i, le bit 'N-1-i' du vecteur 'cipher' est assigné au bit i du vecteur temp\_message.
- Ensuite, 'temp\_message' est assigné à 'message', ce qui permet le déchiffrement du message pour obtenir le texte original en inversant l'ordre des bits du vecteur 'cipher'.

## Simulation

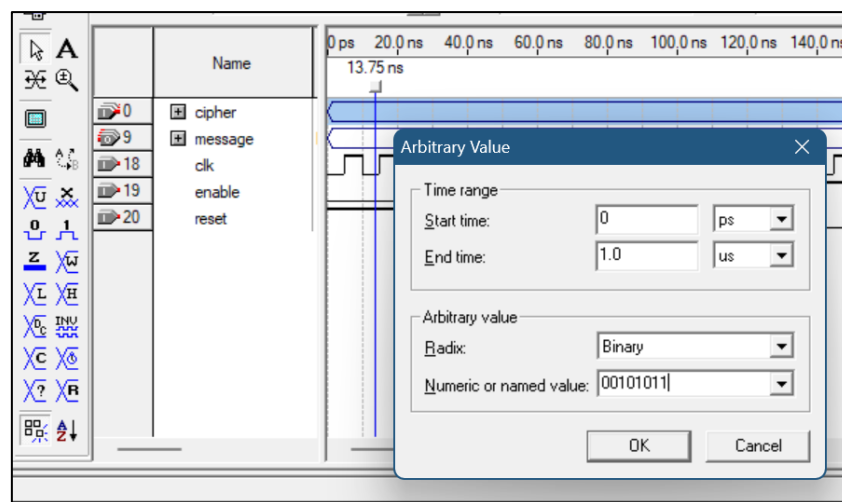
### Le schéma bloc :



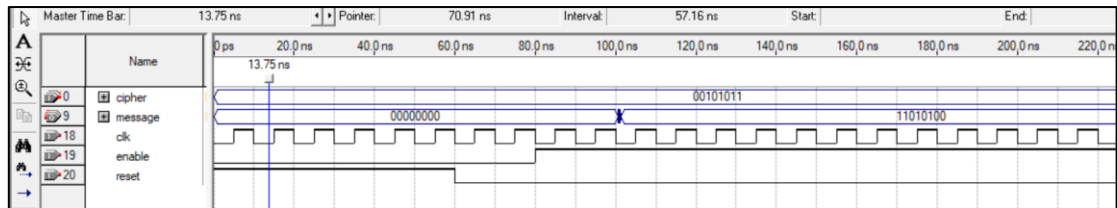
### Simulation :

**ETAPE1 :** donner la valeur du cipher crypter dans la simulation du ReverseCipher

CIPHER='00101011'



## Etape 2 : simuler



Cette simulation montre le processus de décryptage où un signal "cipher" est inversé pour obtenir le signal "message". Voici les points clés de la simulation :

1. Initialisation : Les signaux "message" et "cipher" commencent à zéro.
2. Cycle d'horloge : Le signal "clk" oscille régulièrement.
3. Activation : Le signal "enable" est activé (haut) après un certain temps.
4. Décryptage : À chaque front montant du signal "clk" lorsque "enable" est haut, le contenu du signal "cipher" est inversé et transféré dans "message".
5. Résultat : On observe que le contenu de "cipher" est transféré inversé dans "message". Par exemple, si "cipher" est "00101011", alors "message" devient "11010100".

L'image montre clairement ces étapes avec les transitions des signaux au fil du temps.

## Résumé

La conclusion du rapport sur le chiffrement par inversion en VHDL souligne la simplicité de cette méthode pour réorganiser les caractères d'un message, offrant un niveau de sécurité minimale. Cependant, il est clair que cette approche n'est pas adaptée aux applications nécessitant une sécurité cryptographique robuste. En effet, bien que le code VHDL implémente efficacement le chiffrement par inversion, cette technique ne fournit qu'un obscurcissement minimal et peut être facilement déchiffrée par quiconque a connaissance de l'algorithme utilisé.

En conclusion, bien que le chiffrement par inversion en VHDL soit une démonstration intéressante de la manière dont des techniques de cryptage peuvent être mises en œuvre dans des systèmes matériels, il est important de reconnaître ses limites en termes de sécurité. Pour des applications nécessitant une véritable confidentialité et protection des données, des méthodes de chiffrement plus robustes et éprouvées doivent être privilégiées.

# Conclusion

Le projet de chiffrement par inversion réalisé en VHDL démontre avec succès l'application d'un algorithme simple mais efficace de transformation des données. À travers le développement de l'entité ReverseDecryption et son architecture comportementale, nous avons mis en place un système capable de renverser l'ordre des bits d'un message chiffré pour en obtenir le message original.

Les simulations effectuées confirment le bon fonctionnement du décryptage. Les transitions observées dans les signaux de simulation illustrent clairement le processus de renversement des bits. À chaque activation du signal enable lors d'un front montant du signal d'horloge clk, le contenu de cipher est correctement inversé et stocké dans message.

En outre, ce projet met en lumière l'importance de la conception efficace des algorithmes de traitement des données dans le contexte des systèmes matériels. Bien que le chiffrement par inversion soit une méthode de sécurité minimale, il permet de comprendre les bases de la manipulation des bits et de la gestion des signaux dans un environnement VHDL.

L'application pratique de ce projet peut également servir de fondation pour développer des méthodes de chiffrement plus complexes. En intégrant des techniques de sécurité supplémentaires, telles que des permutations plus sophistiquées ou des algorithmes de chiffrement bien établis comme AES ou DES, les concepts explorés dans ce projet pourraient être étendus pour créer des systèmes de cryptographie plus robustes.

En conclusion, ce travail présente une approche solide pour le chiffrement et le décryptage de messages par inversion des bits, démontrant à la fois la simplicité et l'efficacité de cette méthode. Les résultats obtenus par simulation soutiennent pleinement la fonctionnalité attendue, faisant de ce projet une base fiable pour des applications de cryptographie légère ou des études plus approfondies dans le domaine des systèmes numériques sécurisés.