Chiffremment des messages textuelles en XOR :

- Introduction:

Contexte et Importance du Chiffrement

Dans le monde moderne, la sécurité des données est une priorité essentielle. Avec l'explosion des échanges d'informations via les réseaux informatiques, protéger les données contre les accès non autorisés et les attaques malveillantes est devenu crucial. La cryptographie, qui est l'art de coder les informations pour les rendre inintelligibles à toute personne non autorisée, joue un rôle vital dans cette protection. Elle est utilisée dans diverses applications, telles que les communications sécurisées, les transactions financières en ligne, et le stockage des données sensibles.

Le chiffrement est l'un des principaux outils de la cryptographie. Il transforme les données en clair, compréhensibles, en un format chiffré, illisible sans une clé de déchiffrement appropriée. Parmi les nombreuses méthodes de chiffrement existantes, certaines sont très simples et d'autres extrêmement complexes. Le choix de la méthode dépend souvent du niveau de sécurité requis et des ressources disponibles.

Objectif du Projet

L'objectif principal de ce projet est de concevoir, implémenter et tester un système de chiffrement et de déchiffrement en utilisant une technique de base : le chiffrement XOR (exclusive OR). Bien que le chiffrement XOR soit simple, il est utile pour comprendre les principes fondamentaux de la cryptographie et pour démontrer comment les opérations logiques peuvent être utilisées pour sécuriser les données.

Le projet se concentre sur les aspects suivants :

1. Conception du système : Création de modules VHDL pour le chiffrement et le déchiffrement XOR.

2. Implémentation en VHDL: Codage des modules de chiffrement et de déchiffrement en utilisant le langage de description matérielle VHDL.

3. Test et validation : Développement d'un testbench pour simuler le comportement du système et vérifier son bon fonctionnement.

Aperçu du Chiffrement XOR

Le chiffrement XOR est basé sur l'opération logique XOR, qui est une opération binaire simple mais puissante. L'opération XOR compare deux bits et retourne 1 si et seulement si les bits comparés sont différents (0 XOR 1 = 1, 1 XOR 0 = 1), et retourne 0 s'ils sont identiques (0 XOR 0 = 0, 1 XOR 1 = 0).

Pour chiffrer un message, chaque bit du texte en clair est XOR avec un bit de la clé. Le résultat est un texte chiffré. Pour déchiffrer le message, le texte chiffré est de nouveau XOR avec la même clé, ce qui restitue le texte en clair d'origine en raison des propriétés de l'opération XOR.

Plan du Rapport

Ce rapport détaille chaque étape du projet, depuis la conception initiale jusqu'à la validation finale. Il est structuré comme suit :

1. Théorie du Chiffrement XOR: Explication du principe de l'opération XOR et son application au chiffrement.

2. Conception du Système: Description des modules de chiffrement et de déchiffrement, accompagnée de diagrammes de bloc.

3. Implémentation en VHDL : Présentation du code VHDL et explication détaillée.

4. Test et Validation : Description du testbench, scénarios de test, et analyse des résultats de simulation.

Théorie du chiffrement XOR :

Le chiffrement XOR est basé sur l'opération logique « exclusive OR » (XOR), une opération fondamentale en informatique et en électronique.

Principe de l'Opération XOR

L'opération XOR prend deux bits en entrée et produit un seul bit en sortie. La règle de l'opération XOR est la suivante :

0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0

Propriétés Importantes de XOR

- Commutativité : A XOR B = B XOR A
- Associativité : (A XOR B) XOR C = A XOR (B XOR C)
- Identité : A XOR 0 = A
- Involution : A XOR A = 0

Application dans le Chiffrement

Le chiffrement XOR fonctionne en appliquant l'opération XOR entre chaque bit du texte en clair et un bit de la clé. Voici comment cela fonctionne :

Chiffrement

- Texte en clair (P) : La donnée originale que vous souhaitez chiffrer.
- Clé (K) : Une séquence de bits utilisée pour chiffrer et déchiffrer le texte. Pour une sécurité maximale, la clé doit être de la même longueur que le texte en clair et aléatoire (dans le cas d'un « one-time pad »).
  Pour chaque bit du texte en clair
  P et de la clé K :
  $C=P \oplus K$
  C=P⊕K

Où C C est le texte chiffré.

Déchiffrement

Le processus de déchiffrement utilise la même opération XOR :

Pour chaque bit du texte chiffré C et de la clé K:

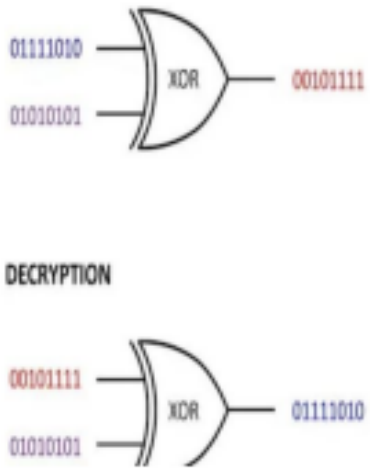
$P=C \oplus K$

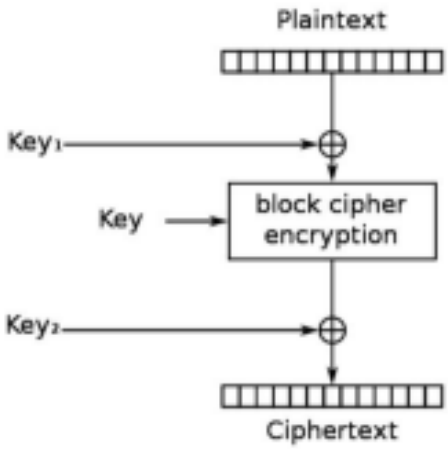
P=C⊕K

Puisque

$P \oplus K$ donne C lors du chiffrement, et $C \oplus K$ donne P lors du déchiffrement grâce à la propriété d'involution de XOR.

- ## Conception du système :





- ## Implémentation en VHDL :

```vhdl
-- XorCipher.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity XorCipher is
    generic (
        N : integer := 4   -- Number of characters in the block
    );
    Port (
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        text_in : in STD_LOGIC_VECTOR(8*N-1 downto 0);
        key : in STD_LOGIC_VECTOR(8*N-1 downto 0);
        text_out : out STD_LOGIC_VECTOR(8*N-1 downto 0)
    );
end XorCipher;

architecture Behavioral of XorCipher is
begin
    process(clk, reset)
    begin
        if reset = '1' then
            text_out <= (others => '0');
        elsif rising_edge(clk) then
            text_out <= text_in XOR key;
        end if;
    end process;
end Behavioral;
```

```vhdl
-- XorDecipher.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity XorDecipher is
    generic (
        N : integer := 4   -- Number of characters in the block
    );
    Port (
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        cipher_text : in STD_LOGIC_VECTOR(8*N-1 downto 0);
        key : in STD_LOGIC_VECTOR(8*N-1 downto 0);
        text_out : out STD_LOGIC_VECTOR(8*N-1 downto 0)
    );
end XorDecipher;

architecture Behavioral of XorDecipher is
begin
    process(clk, reset)
    begin
        if reset = '1' then
            text_out <= (others => '0');
        elsif rising_edge(clk) then
            text_out <= cipher_text XOR key;
        end if;
    end process;
end Behavioral;
```

Explication CipherXOR :

- Entité XorCipher : Déclare les ports d'entrée et de sortie ainsi qu'un paramètre générique N pour le nombre de caractères à chiffrer.
- Architecture Behavioral : Contient un processus sensible au signal d'horloge (clk) et de reset (reset).
- Lorsque le signal reset est actif ('1'), la sortie (text_out) est mise à zéro.
- À chaque front montant de l'horloge (rising_edge(clk)), l'opération XOR est effectuée entre text_in et key, et le résultat est assigné à text_out

Explication DecipherXOR :

- Entité XorDecipher : Déclare les ports d'entrée et de sortie ainsi qu'un paramètre générique N pour le nombre de caractères à déchiffrer.

- Architecture Behavioral : Contient un processus sensible au signal d'horloge (clk) et de reset (reset).

- Lorsque le signal reset est actif (‘1’), la sortie (text_out) est mise à zéro.
- À chaque front montant de l’horloge (rising_edge(clk)), l’opération XOR est effectuée entre cipher_text et key, et le résultat est assigné à text_out.

Explication du testbench :

- Entité XorCipher_Decipher_tb : Déclare une entité vide, car il s’agit d’un testbench.
- Architecture Behavioral :
- Déclare les signaux et constantes nécessaires pour la simulation, y compris les signaux d’entrée et de sortie pour les modules de chiffrement et de déchiffrement.

  Fonction to_hex_string : Convertit un vecteur de bits en une chaîne de caractères hexadécimale.

  Instanciation des modules XorCipher et XorDecipher : Relie les signaux de testbench aux ports des modules.

  Processus clk_process : Génère un signal d’horloge périodique.

  Processus stimulus_process : Applique des stimuli aux modules pour tester leur comportement.

  Applique un reset initial.

  Définit un cas de test où le texte « ABCD » est chiffré avec la clé « 1234 ».

Attend quelques cycles d'horloge pour observer les résultats.

Affiche les textes chiffré et déchiffré.

- **Conclusion :**

Le projet de chiffrement et déchiffrement basé sur l'opération XOR présente une implémentation simple mais efficace pour comprendre les principes fondamentaux de la cryptographie symétrique. L'utilisation de VHDL pour modéliser et simuler ce système permet de démontrer plusieurs concepts clés de la conception numérique et de la sécurité des données.

L'utilisation d'un testbench ('XorCipher_Decipher_tb.vhd') pour simuler les modules de chiffrement et de déchiffrement est essentielle pour valider le fonctionnement correct du système. Les tests incluent des scénarios réalistes avec des textes d'entrée et des clés spécifiques, permettant de vérifier que le chiffrement et le déchiffrement sont corrects.

Limitations :

Bien que l'opération XOR soit efficace et simple, elle n'est pas sécurisée pour des applications réelles sans des techniques complémentaires. La réutilisation de la même clé pour différents messages peut conduire à des vulnérabilités. Pour des applications nécessitant une sécurité élevée, des algorithmes de chiffrement plus complexes comme AES (Advanced Encryption Standard) sont recommandés.