ÉCOLE NATIONALE SUPÉRIEURE D'ARTS ET MÉTIERS
UNIVERSITÉ HASSAN II DE CASABLANCA

1ère année du cycle d'ingénieur

Management des systèmes électriques intelligents

Génie électromécanique

Composants Prog. VHDL

Rapport du Projet

Sous le thème :

Description Hardware de l'algorithme de chiffrement DES (Data Encryption Standard)

Fait par :

El Fahim Mohamed Gourtane Anouar Chami Salah

El Faqyr Soukaina

Encadré par :

M. El Moumni

2023 - 2024

Résumé

Dans ce projet, nous avons concrétisé l'algorithme de chiffrement DES (Data Encryptions Standard) en utilisant le langage VHDL. Le DES, un puissant algorithme de cryptage symétrique, est conçu pour sécuriser des blocs de données de 64 bits à l'aide d'une clé de 56 bits. Après une étude approfondie des principes fondamentaux du DES, nous avons traduit ces connaissances en un code VHDL robuste et efficace. Notre implémentation a été validée par des simulations détaillées, confirmant la fiabilité du cryptage des données. Cette expérience a enrichi notre compréhension du DES et a consolidé notre expertise dans le développement en VHDL, tout en mettant en lumière l'importance cruciale de la cryptographie matérielle dans la protection des informations sensibles.

Abstract

This project focuses on the implementation of the Data Encryption Standard (DES) algorithm using the VHDL (VHSIC Hardware Description Language). DES is a symmetric encryption algorithm designed to secure 64-bit data blocks with a 56-bit key. The project begins with an in-depth study of the DES principles, followed by the translation of these concepts into efficient VHDL code. The implementation aims to simulate the hardware required for reliable encryption and decryption processes. Through rigorous testing and validation, our VHDL code demonstrates the successful encryption of data, ensuring compliance with DES specifications. This project enhances our understanding of DES and strengthens our proficiency in VHDL development, underscoring the critical role of hardware cryptography in safeguarding sensitive information.

Sommaire

Liste des figures

Introduction générale.

➢ Les algorithmes de Chiffrement :

Les algorithmes de chiffrement sont des méthodes essentielles pour assurer la confidentialité et la sécurité des données dans le monde numérique. Ils transforment les données lisibles, appelées texte en clair, en données illisibles, appelées texte chiffré, en utilisant une clé de chiffrement. Cette transformation rend les informations inaccessibles aux personnes non autorisées. Il existe deux principales catégories d'algorithmes de chiffrement : symétriques et asymétriques. Les algorithmes symétriques utilisent la même clé pour le chiffrement et le déchiffrement, tandis que les algorithmes asymétriques utilisent des clés différentes pour ces opérations. Ces technologies sont fondamentales pour sécuriser les communications, les transactions financières et protéger les informations sensibles contre les cyberattaques.

➢ Langage VHDL :

VHDL (VHSIC Hardware Description Language) est un langage de description matériel utilisé pour modéliser, simuler et synthétiser des circuits électroniques. Il permet de décrire le comportement et la structure de systèmes numériques complexes, en fournissant un moyen précis et standardisé de spécifier les composants et leur interconnexion. Utilisé principalement dans le domaine de la conception de circuits intégrés et de systèmes sur puce, VHDL permet aux ingénieurs de développer des modèles fonctionnels avant la fabrication physique, facilitant ainsi la vérification et la validation des conceptions. Grâce à ses capacités de modélisation à différents niveaux d'abstraction, VHDL est un outil puissant pour la conception de matériel numérique, allant des simples portes logiques aux systèmes intégrés sophistiqués

➢ Objectif du Projet

L'objectif de notre projet est de décrire, simuler et implémenter en VHDL l'algorithme de chiffrement DES (Data Encryption Standard). Cet algorithme, bien que plus ancien, reste une référence classique dans le domaine du chiffrement symétrique. En utilisant VHDL, nous visons à modéliser le processus de chiffrement DES, en détaillant chaque étape, de la permutation initiale à la substitution finale à travers les itérations successives. Ce projet permettra de comprendre en profondeur les mécanismes internes de DES et de démontrer la capacité du langage VHDL à représenter et simuler des algorithmes de chiffrement complexes, renforçant ainsi notre expertise en conception de systèmes de sécurité numérique.

Chapitre I : Description de l'algorithme DES.

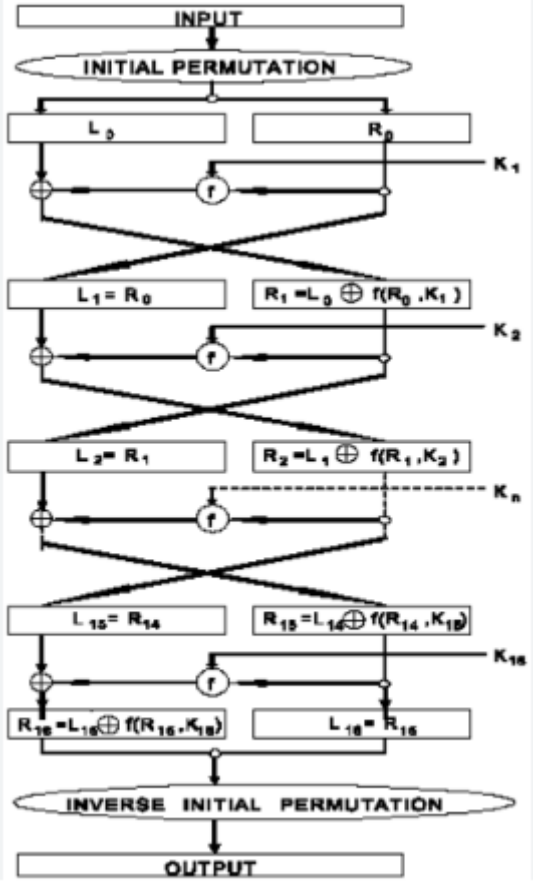


Figure 1: Illustration des instructions de l'algorithme DES.

Définition :

L'algorithme de chiffrement DES (Data Encryptions Standard) est l'un des premiers algorithmes de chiffrement symétrique largement utilisés dans le domaine de la cryptographie. Conçu dans les années 1970 par IBM, il est devenu un standard du gouvernement fédéral des États-Unis en 1977. Bien que désormais obsolète en raison de sa clé relativement courte de 56 bits, il reste une pierre angulaire importante de la cryptographie moderne.

L'algorithme DES opère sur des blocs de données de 64 bits, divisés en deux parties égales de 32 bits. Il utilise une clé de 56 bits, bien que seules 64 des 56 clés soient réellement utilisées pour le chiffrement. Le processus de chiffrement comporte 16 itérations principales, chacune utilisant une sous-clé de 48 bits dérivée de la clé principale.

Au cœur de l'algorithme DES se trouvent des opérations telles que la permutation, la substitution et le décalage, qui agissent de manière itérative sur les données et les sous-clés pour produire le texte chiffré.

En raison de son efficacité et de sa sécurité relative à l'époque de sa conception, l'algorithme DES a été largement utilisé dans divers domaines, allant des transactions financières aux communications gouvernementales. Cependant, avec l'avènement de la cryptographie moderne et des ordinateurs plus puissants, sa clé relativement courte est devenue vulnérable aux attaques par force brute, ce qui a conduit à son remplacement par des algorithmes de chiffrement plus robustes.

I. Description de l'algorithme.

L'algorithme de chiffrement **DES** est basé sur **16 itérations**, les instructions de chaque itération sont les mêmes ce qui change seulement C'est la clé K qu'on génère à chaque itération (k1, k2, k3……k16) Et donc par la suite on peut expliquer seulement une seule itération comme il est montré dans la figure ci-dessous.

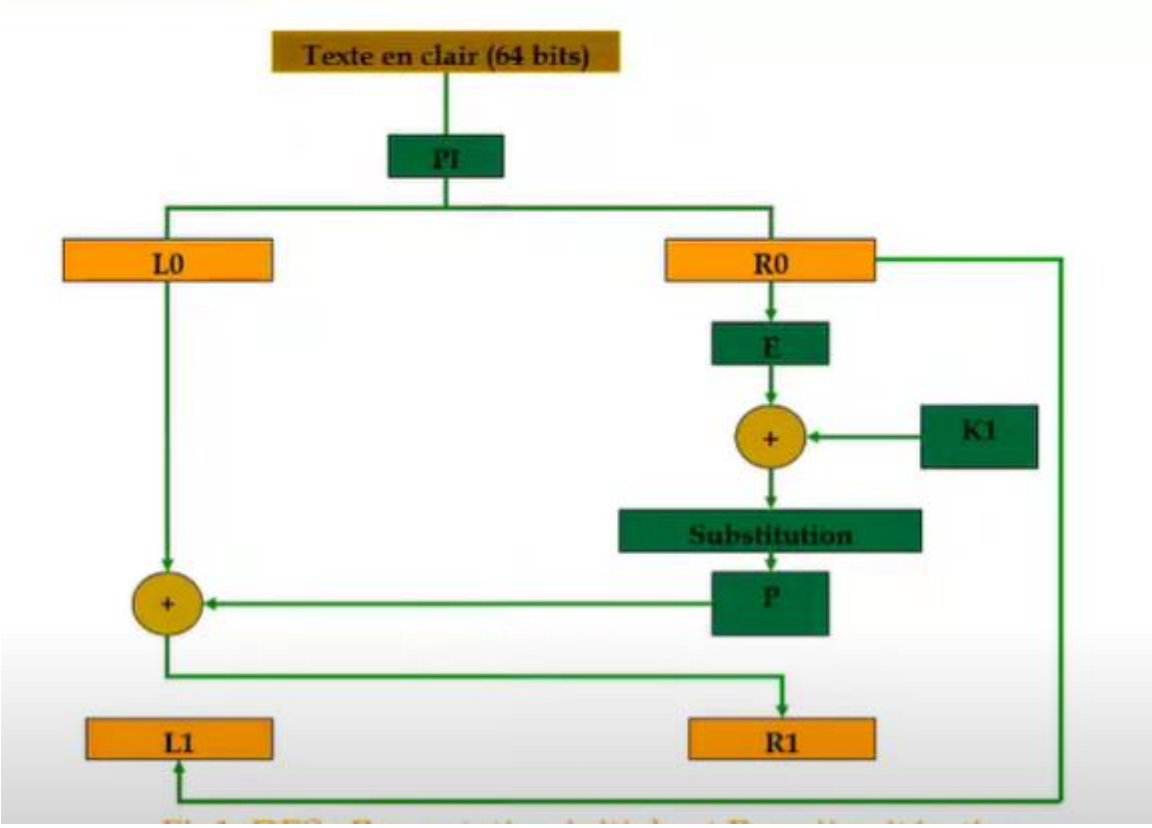


Figure 2: première itération de l'algorithme DES.

1. Avant itération :

Avant de commencer l'itération il faut passer notre donnée (**entrée**) de **64bits** par une **permutation initiale** suivant le tableau suivant :

Permutation initiale PI							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Figure 3 : Tableau de la première permutation.

Cela veut dire que le bit de **l'indice 58** de la donnée sera à **la première position**, le bit 50 à la deuxième position ainsi de suite jusqu'à permutation de tous les bits de la donnée, ensuite on devise notre trame après permutation sur deux **R0** et **L0** de taille **32bits** chacune. Et maintenant qu'on peut commencer **l'itération** en procédant sur **R0** et **L0** afin d'avoir **R1** et **L1**, ensuite pour la deuxième itération R1 et L1 prennent les places de R0 et L0 afin d'avoir **R2** et **L2** et ainsi de suite jusqu'à la 16ème itération (**R16**, **L16**).

2. Description de la 1er Itération :

Départ (L0, R0) >>>>>>>>>>>>>> arrivé (L1, R1)

✓ Pour L1 :

L1 est obtenu directement on **leur affecte R0**

✓ Pour R1 :

Pour arriver à R1d'après R0 il faut passer par plusieurs blocs,

➢ Bloc d'expansion :

Au début on passe la trame R0 (**32bits**) par **une expansion** afin que sa taille soit de **48bit** selon le tableau au-dessous :

Table de sélection de bits E					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	39
28	29	30	31	32	1

Figure 4 : Tableau de l'Expansion.

On procède de la même façon que le tableau de permutation, mais dans ce cas on remarque que certains bits (indice) sont **dupliqués** chose qui est normale parce que la trame R0 est de **32bits** alors que nous on désire une trame de **48bits** donc on va prendre certains bits (indice) deux fois pour compléter **les 48bits**.

➢ Bloc de l'opération logique XOR :

Au niveau de ce bloc on effectue un XOR avec la trame R0 **après Expansion** (***48bits***) et la première clé **K1(*48bits*).** A la fin de l'explication de cette première itération, on va expliquer comment on génère les clé k(i) à chaque itération. La taille des clés générées et la même de la trame R0 après expansion.

➢ Bloc de substitution :

Après avoir obtenir le résultat de l'XOR (48bits) on le passe ensuite par le bloc de substitution afin de **réduire sa taille à 32bits**.ce bloc est spécial n'est plus comme les blocs précédents qui utilisent des tableaux. Alors comment il **réduit** la taille de la trame XOR (48bits) ?

En fait ce bloc et constituer des **S-Boxes** (S1, S2, S3, S4 ……, S8), il possède **8** S-boxes, chaque box pend **6bits**, ce qui donne une taille de **48 bits** qui est identique à la taille de du résultat de l'XOR, donc dans un premier temp on substitue ce résultat XOR sur les 8 boxes, ensuite chaque box va donner à la sorite **4bits** ce qui donne une taille de **32bits** pour les 8-box et c'est le résultat qu'on souhaite avoir, comme il est montré sur les figure au-dessous :

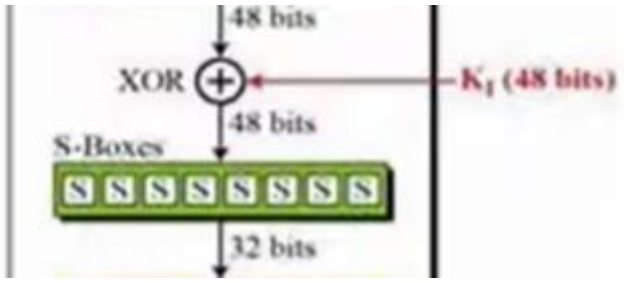


Figure 5 : Illustration du bloc de la substitution.

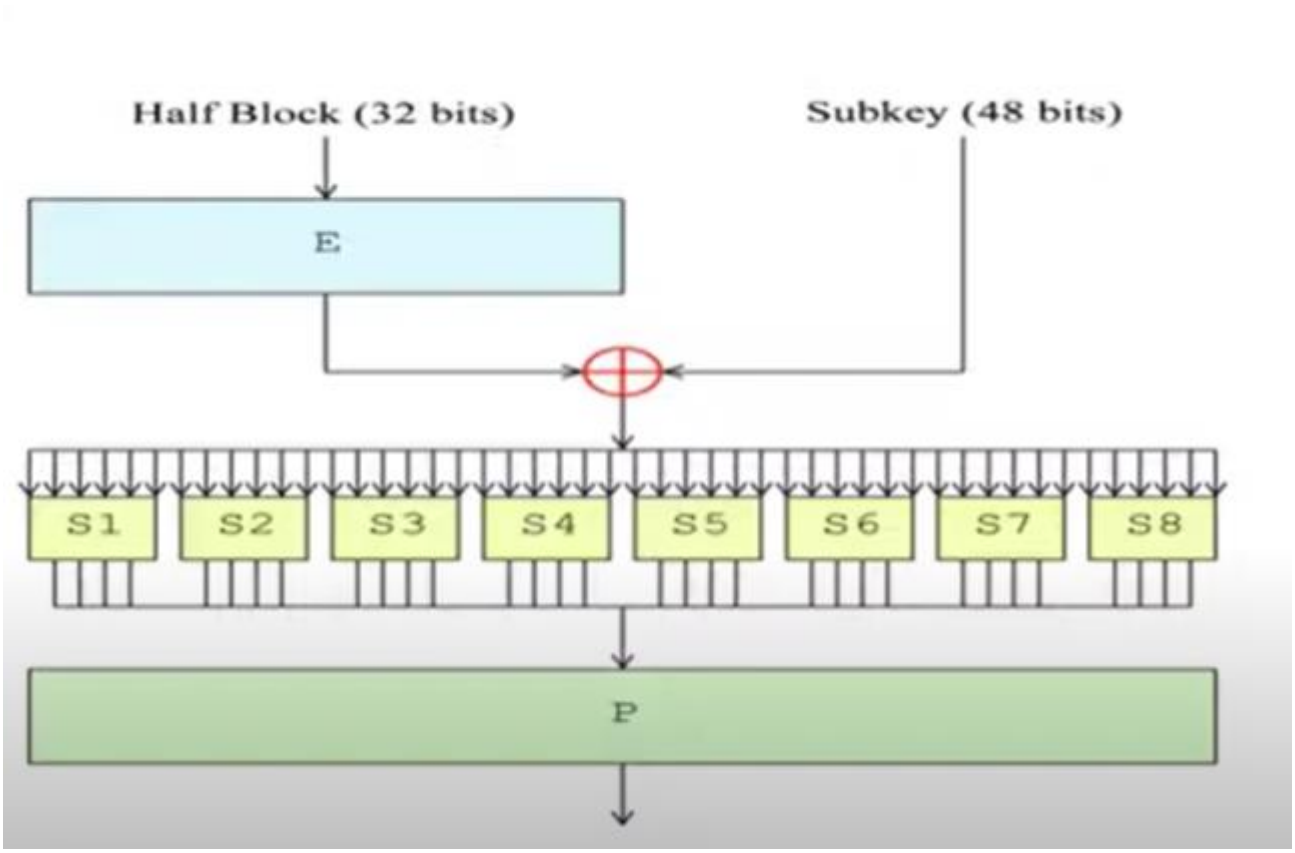
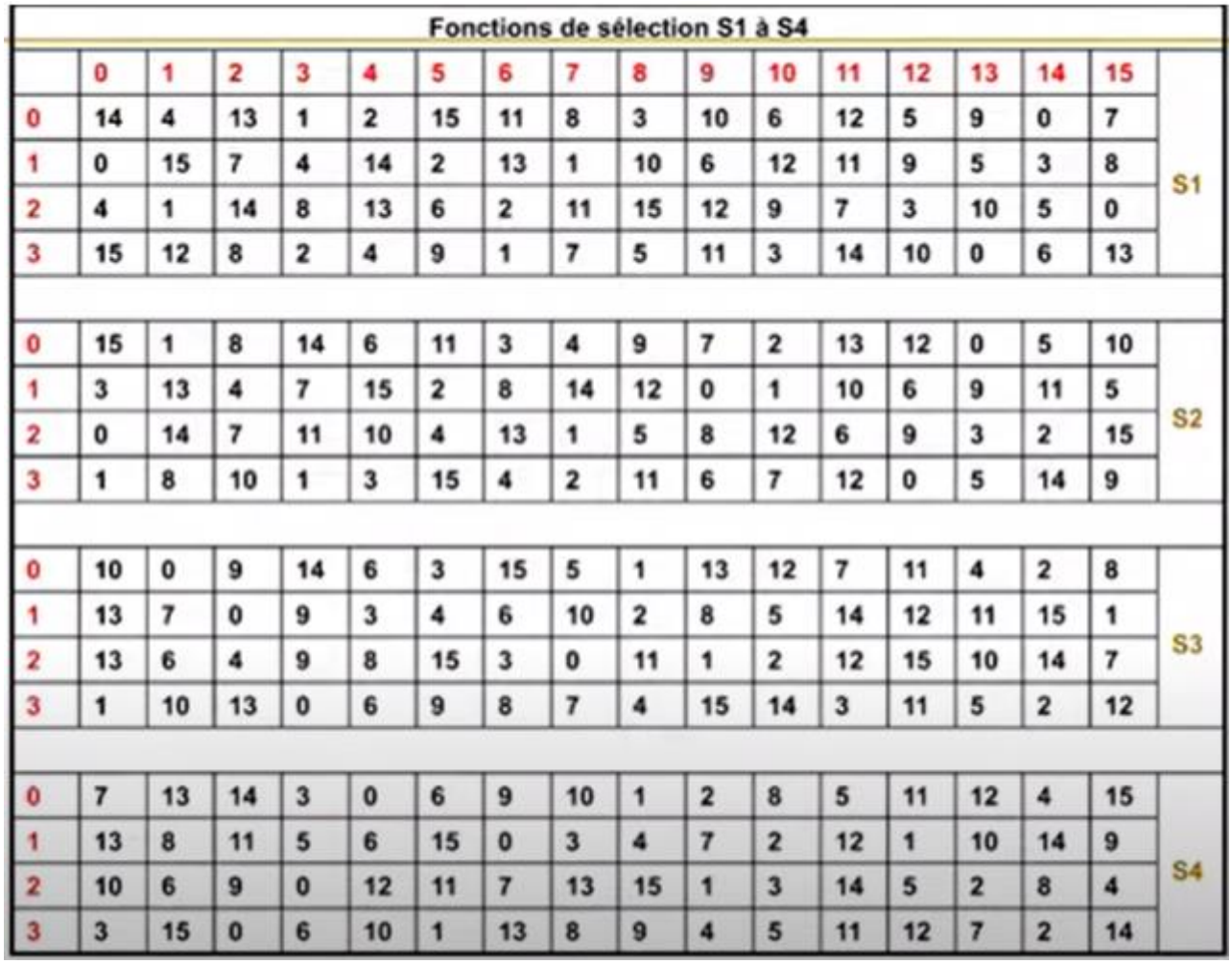


Figure 6 : Illustration de l'intérieur du bloc de substitution.

- Comment la box génère les 4bits de sorite ?

D'abord chaque box possède une matrice de **taille (4*16)** Les éléments de cette matrice sont des entiers naturels varient entre **0 et 15**, comment il est montré sur la figure au-dessous :

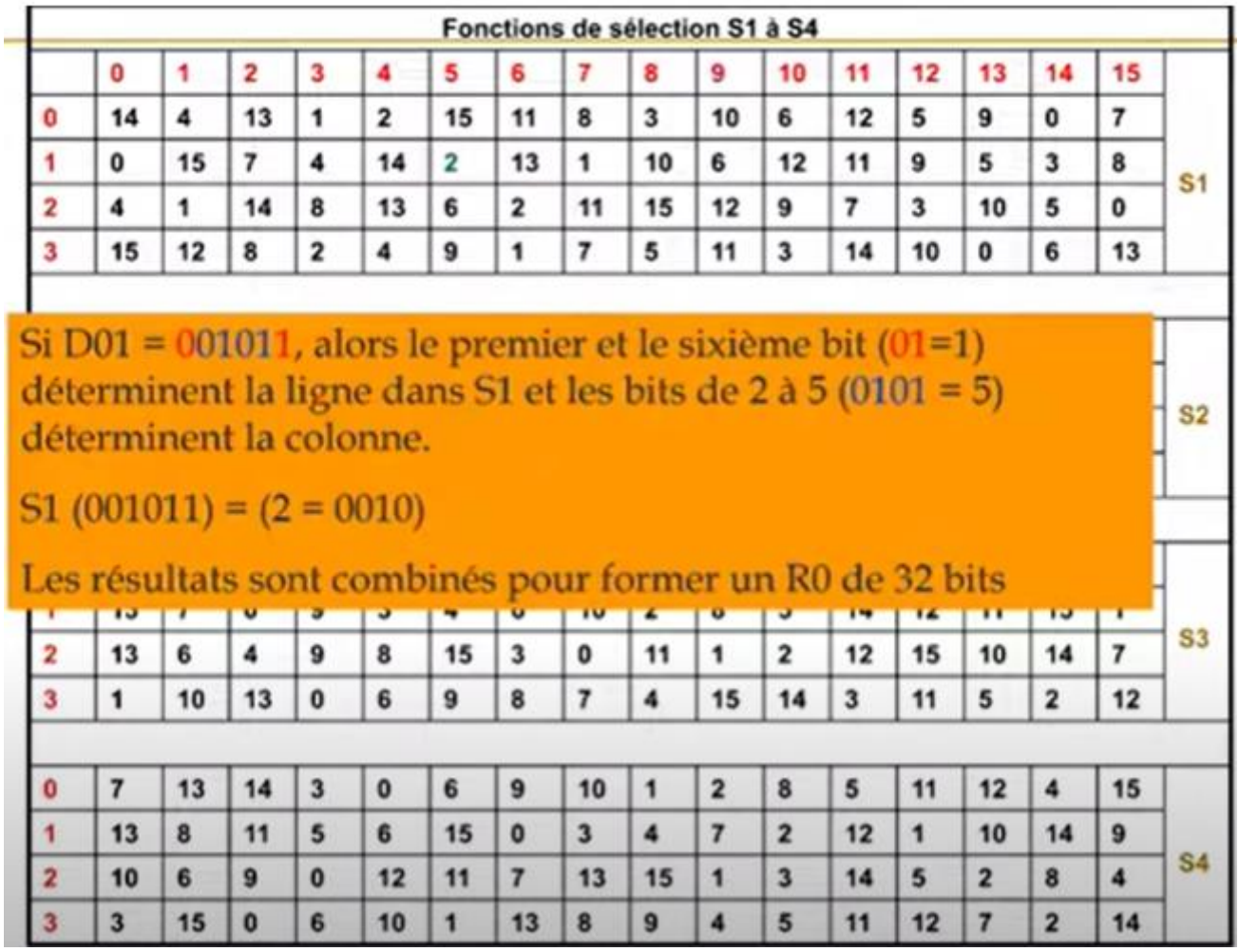


Fonctions de sélection S1 à S4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	S1
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	S2
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	
3	1	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	S3
2	13	6	4	9	8	15	3	0	11	1	2	12	15	10	14	7	
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	S4
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4	
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14	

Figure 7 : Illustration des Matrices des S-BOXES.

Donc l'objectif et de **pointer** sur un élément de la matrice d'après les **6bits** d'entrée de la box, une fois l'élément et pointé on le **convertis** en binaire sur **4bits** ce qui représente la sortie de la box et c'est pour cette raison que les entiers de la matrice **ne dépasse pas le 15** pour ne pas utiliser plus que **4bits** , pour le pointage et lorsque la box reçoit les **6bits** elle prend Le **premier** et le **sixième** bit pour pointé sur la **ligne**, et les **4bits** restant pour pointé sur la **colonne**, normalement le nombre de combinaison pour **deux bits** est **4** ce qui démontre le **nombre de lignes** de la matrice (**4 ligne**) ainsi que le nombre de combinaison pour **quatre bits** qui est **16** ce qui démontre **le nombre de colonnes** de la matrice (**16 colonnes**).la figurent au dessous represente un exemple de la procédure qu'on viennent d'expliquer :



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S1

2	13	6	4	9	8	15	3	0	11	1	2	12	15	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S3

0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S4

Figure 8 : Exemple de pointage sur un élément de la matrice de S1.

Pour toutes les boxes on suit la même procédure pour prélever les 4bits ce qui change c'est la matrice, chaque box elle a sont propre matrice. A la fin les résultats sont combinés pour former une trame (R0) de 32bits.

➢ Bloc de permutation :

Après combinaison des résultats des S-boxes, on obtient une trame de 32bits qu'on passe par une permutation selon le tableau suivant :

Permutation P			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Figure 9 : Tableau de la permutation P.

➢ Bloc de l'opération logique XOR :

On se retrouve encore devant **un 2ème opérateur XOR**, et c'est le dernier bloc pour l'itération, Donc on fait un XOR de la dernière trame obtenue après la passer par le bloc de la permutation et la trame **L0(32bits)** afin de trouver finalement **R1(32bits).**

Une fois l'itération et terminer, on reprend les mêmes instructions pour la 2ème itération mais cette fois-ci avec R1 et L1, cependant on doit générer une nouvelle clé K2, dans la partie suivante on va expliquer comment on génère les clé (k1, k2, k3, k4…….. K16) à chaque itération.

❖ Génération des clés k(i) :

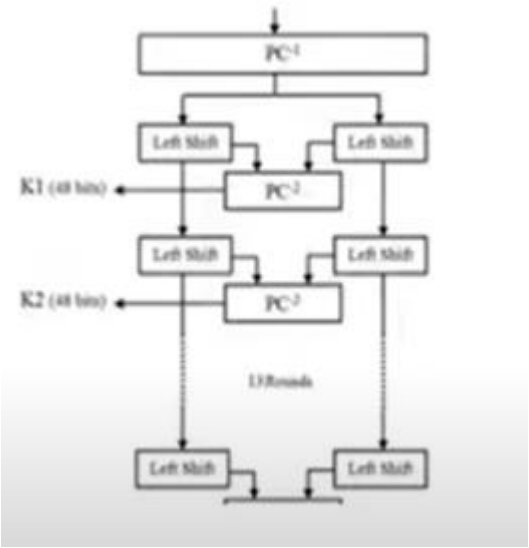


Figure 30 : Illustration des instructions de génération des clés K(i).

CP1						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	4	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Figure 41 : Tableau de la permutation CP1.

Au début on a la clé K principale (**64bits**) qu'on passe par une permutation **CP1** afin de rendre sa taille de **56bits** selon le tableau au-dessus, ensuite on devise la trame obtenue sur deux partie identique (**G0**, **D0**) de taille **28bits** chacune, maintenant et à chaque itération on décale d'un cran à gauche les parties (G0, D0), on les concatènent après décalage pour revenir à une nouvelle trame de **56 bits** qu'on passe par une permutation **CP2** selon le tableau au-dessous afin d'obtenir la première **clé K1**, de la même façon on décale une deuxième fois les parties (G0, D0) qui sont déjà décalées, on les concatènent ensuite on passe la nouvelle trame obtenue (56bits) par **la même** permutation **CP2** afin de générer **K2** ainsi de suite jusqu'à la 16ème itération.

CP2					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Figure 52 : Tableau de la permutation CP2.

3. Fin de la 16ème itération :

Après obtention des trames R16 et L16 on les **concatènes** et on fin on les passe par une **permutation initiale inverse** afin d'obtenir la donnée chiffré (S) selon le tableau au-dessus :

Permutation initiale inverse PI-1							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Figure 63 : Tableau de la permutation initiale inverse.

Chapitre II : Présentation du code VHDL.

Dans cette partie on va expliquer le code tous entier en insérant à chaque fois le screen de la partie du code concernée.

I. Bibliothèques est entité

La figure au-dessus montre la bibliothèque utilisé (IEEE) ainsi que les paquetages nécessaires pour spécifier les types des signaux et variables ainsi que la permission des opérations logiques et arithmétiques.

La figure montre également l'entité nommé « DES-ALGO », dans laquelle on a déclaré les signaux externes du circuit que sera dédier au chiffrement DES. En fait on a que trois signaux :

- **E** (entrée) : la donnée (64bits) **à chiffrer**.
- **K** (entrée) : la clé principale (64bits).
- **S** (sortie) : la donnée (64bits) **après chiffrement**.

```
1   Library IEEE;
2   use IEEE.std_logic_1164.all;
3   use IEEE.numeric_std.all;
4   use IEEE.std_logic_unsigned.all;
5
6   entity DES_ALGO is
7       PORT(E,K : in std_logic_vector(63 downto 0);
8            S: out std_logic_vector(63 downto 0));
9   end DES_ALGO ;
10
```

Figure 74 : Partie Code pour la déclaration des biblio et l'entity.

II. Architecture

Dans la architecture on a décrit ce qui ce passe à l'intérieur du circuit de chiffrement, autrement dite la description **du jeu d'instruction** de l'algorithme DES , et puisque on parle **d'instructions** et **d'ordre** à respecter donc on a décidé d'utiliser « ***process*** » dont **sa liste de sensibilité** contienne les signaux (**E,K**) qu'on viennent de déclarer dans l'entité ,cela signifie que chaque **changement** de la donnée (une nouvelle donnée à chiffrer) ou de la clé (nouvelle clé à utiliser) **provoque la reprise du process** tout entier.

1. Avant process :

Alors, avant de décrire l'intérieur du process on va parler tout d'abord des différentes **déclaration**s qu'on a effectué, donc comme il est montré dans les figures au-dessous où on a déclaré **un tableau de trois dimensions** pour les **S-BOXES** (**1èr** dimension => **nombres de boxes** ; **2ème** dimension => **nombre de lignes** de la matrice ; **3ème** dimension => **nombre de colonnes** de la matrice)

Ensuite on a déclaré une constante dans laquelle on a stocké les matrices des S-boxes

```
11  architecture DISCP of DES_ALGO is
12
13  -- Déclaration d'un tableau de trois dimensions pour les matrices de taille (4 * 16) des 8 S-boxes
14     type sbox_table is array (0 to 7, 0 to 3, 0 to 15) of std_logic_vector(3 downto 0);
15
16  -- Les martices des S-boxes stockées dans la constante "S_boxes"
17     constant S_Boxes : sbox_table := (
18         -- S1
19         (
20             ("1110", "0100", "1101", "0001", "0010", "1111", "1011", "1000", "0011", "1010", "0110", "1100", "0101", "1001", "0000", "0111"),
21             ("0000", "1111", "0111", "0100", "1110", "0010", "1101", "0001", "1010", "0110", "1100", "1011", "1001", "0101", "0011", "1000"),
22             ("0100", "0001", "1110", "1000", "1101", "0110", "0010", "1011", "1111", "1100", "1001", "0111", "0011", "1010", "0101", "0000"),
23             ("1111", "1100", "1000", "0010", "0100", "1001", "0001", "0111", "0101", "1011", "0011", "1110", "1010", "0000", "0110", "1101")
24         ),
25         -- S2
26         (
27             ("1111", "0001", "1000", "1110", "0110", "1011", "0011", "0100", "1001", "0111", "0010", "1101", "1100", "0000", "0101", "1010"),
28             ("0011", "1101", "0100", "0111", "1111", "0010", "1000", "1110", "1100", "0000", "0001", "1010", "0110", "1001", "1011", "0101"),
29             ("0000", "1110", "0111", "1011", "1010", "0100", "1101", "0001", "0101", "1000", "1100", "0110", "1001", "0011", "0010", "1111"),
30             ("1101", "1000", "1010", "0001", "0011", "1111", "0100", "0010", "1011", "0110", "0111", "1100", "0000", "0101", "1110", "1001")
31         ),
32         -- S3
33         (
34             ("1010", "0000", "1001", "1110", "0110", "0011", "1111", "0101", "0001", "1101", "1100", "0111", "1011", "0100", "0010", "1000"),
35             ("1101", "0111", "0000", "1001", "0011", "0100", "0110", "1010", "0010", "1000", "0101", "1110", "1100", "1011", "1111", "0001"),
36             ("1101", "0110", "0100", "1001", "1000", "1111", "0011", "0000", "1011", "0001", "0010", "1100", "0101", "1010", "1110", "0111"),
37             ("0001", "1010", "1101", "0000", "0110", "1001", "1000", "0111", "0100", "1111", "1110", "0011", "1011", "0101", "0010", "1100")
38         ),
39         -- S4
40         (
41             ("0111", "1101", "1110", "0011", "0000", "0110", "1001", "1010", "0001", "0010", "1000", "0101", "1011", "1100", "0100", "1111"),
42             ("1101", "1000", "1011", "0101", "0110", "1111", "0000", "0011", "0100", "0111", "0010", "1100", "0001", "1010", "1110", "1001"),
43             ("1010", "0110", "1001", "0000", "1100", "1011", "0111", "1101", "1111", "0001", "0011", "1110", "0101", "0010", "1000", "0100"),
44             ("0011", "1111", "0000", "0110", "1010", "0001", "1101", "1000", "1001", "0100", "0101", "1011", "1100", "0111", "0010", "1110")
45         ),
```

Figure 15 : Déclaration du tableau (3 dimensions) et remplissage des matrices.

```
        -- S5
        (
            ("0010", "1100", "0100", "0001", "0111", "1010", "1011", "0110", "1000", "0101", "0011", "1111", "1101", "0000", "1110", "1001"),
            ("1110", "1011", "0010", "1100", "0100", "0111", "1101", "0001", "0101", "0000", "1111", "1010", "0011", "1001", "1000", "0110"),
            ("0100", "0010", "0001", "1011", "1010", "1101", "0111", "1000", "1111", "1001", "1100", "0101", "0110", "0011", "0000", "1110"),
            ("1011", "1000", "1100", "0111", "0001", "1110", "0010", "1101", "0110", "1111", "0000", "1001", "1010", "0100", "0101", "0011")
        ),
        -- S6
        (
            ("1100", "0001", "1010", "1111", "1001", "0010", "0110", "1000", "0000", "1101", "0011", "0100", "1110", "0111", "0101", "1011"),
            ("1010", "1111", "0100", "0010", "0111", "1100", "1001", "0101", "0110", "0001", "1101", "1110", "0000", "1011", "0011", "1000"),
            ("1001", "1110", "1111", "0101", "0010", "1000", "1100", "0011", "0111", "0000", "0100", "1010", "0001", "1101", "1011", "0110"),
            ("0100", "0011", "0010", "1100", "1001", "0101", "1111", "1010", "1011", "1110", "0001", "0111", "0110", "0000", "1000", "1101")
        ),
        -- S7
        (
            ("0100", "1011", "0010", "1110", "1111", "0000", "1000", "1101", "0011", "1100", "1001", "0111", "0101", "1010", "0110", "0001"),
            ("1101", "0000", "1011", "0111", "0100", "1001", "0001", "1010", "1110", "0011", "0101", "1100", "0010", "1111", "1000", "0110"),
            ("0001", "0100", "1011", "1101", "1100", "0011", "0111", "1110", "1010", "1111", "0110", "1000", "0000", "0101", "1001", "0010"),
            ("0110", "1011", "1101", "1000", "0001", "0100", "1010", "0111", "1001", "0101", "0000", "1111", "1110", "0010", "0011", "1100")
        ),
        -- S8
        (
            ("1101", "0010", "1000", "0100", "0110", "1111", "1011", "0001", "1010", "1001", "0011", "1110", "0101", "0000", "1100", "0111"),
            ("0001", "1111", "1101", "1000", "1010", "0011", "0111", "0100", "1100", "0101", "0110", "1011", "0000", "1110", "1001", "0010"),
            ("0111", "1011", "0100", "0001", "1001", "1100", "1110", "0010", "0000", "0110", "1010", "1101", "1111", "0011", "0101", "1000"),
            ("0010", "0001", "1110", "0111", "0100", "1010", "1000", "1101", "1111", "1100", "1001", "0000", "0011", "0101", "0110", "1011")
        )
    );

begin
```

Figure 16 : Suite du remplissage des Matrices .

2. Intérieurs du process

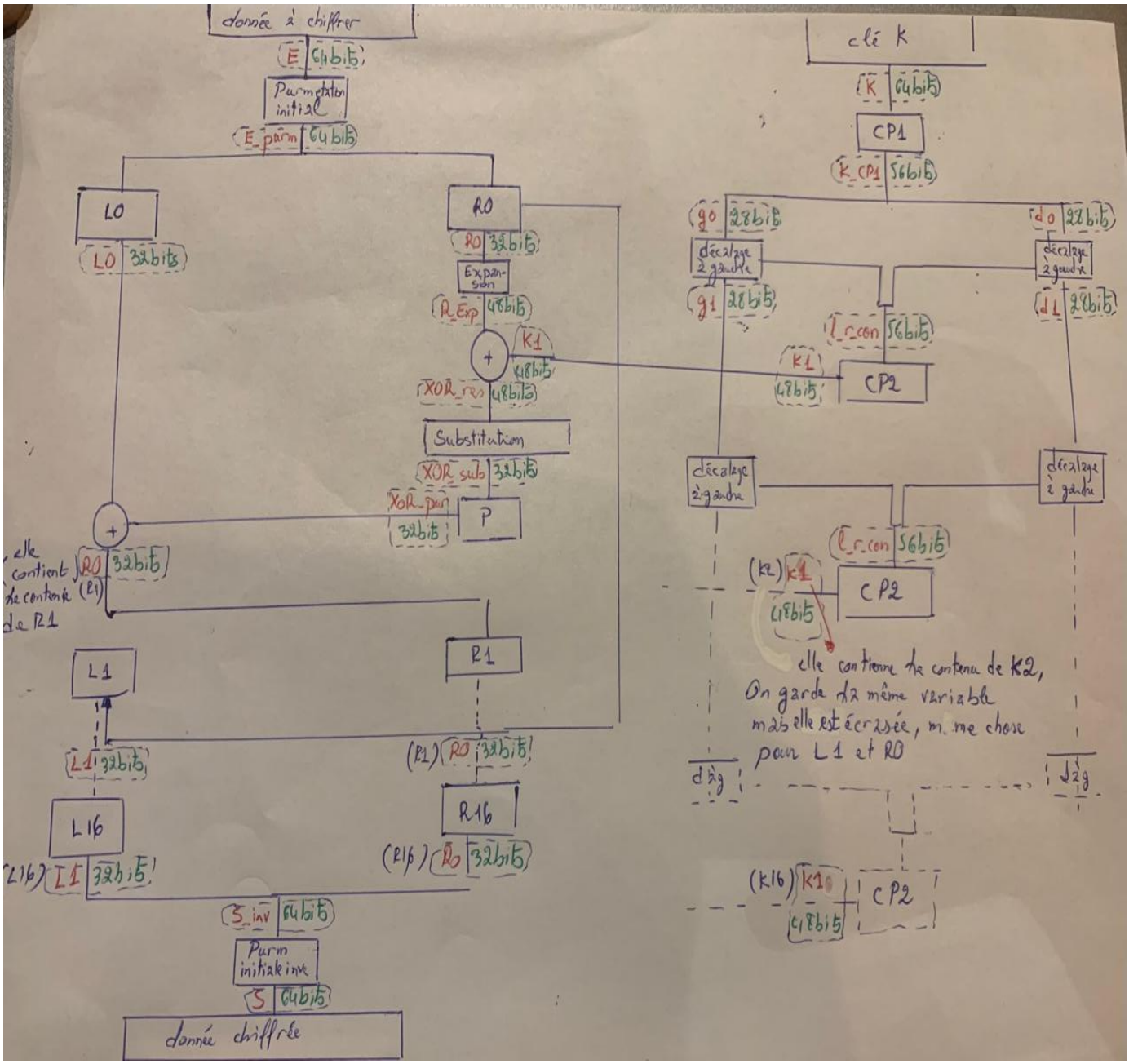
2.1. Variables déclarer :

```
begin
        process(E, K)

        -- Déclaration des variables necessaires utilisées dans le process
          variable E_purm : std_logic_vector(63 downto 0);
          variable L0 : std_logic_vector(31 downto 0);
          variable R0 : std_logic_vector(31 downto 0);
          variable R1 : std_logic_vector(31 downto 0);
          variable L1: std_logic_vector(31 downto 0);
          variable R_Exp : std_logic_vector(47 downto 0);
          variable K_CP1: std_logic_vector(55 downto 0);
          variable g0 : std_logic_vector(27 downto 0);
          variable d0 : std_logic_vector(27 downto 0);
          variable g1 : std_logic_vector(27 downto 0);
          variable d1 : std_logic_vector(27 downto 0);
          variable l_r_con : std_logic_vector(55 downto 0);
          variable K1 : std_logic_vector(47 downto 0);
          variable XOR_res : std_logic_vector(47 downto 0);
          variable XOR_sub : std_logic_vector(31 downto 0);
          variable XOR_pur : std_logic_vector(31 downto 0);
          variable S_inv : std_logic_vector(63 downto 0);

          -- Déclaration des variables  utilisées dans la boucle du bloc de substitution (S-boxes)
          variable temp : std_logic_vector(31 downto 0);
          variable groupe : std_logic_vector(5 downto 0);
          variable row : integer range 0 to 3;
          variable col : integer range 0 to 15;
          variable sbox_value : std_logic_vector(3 downto 0);


   begin
```

Figure 87 : Déclaration des variables.

On a **déclaré** également **différentes variables de déférentes tailles** qu'on aura besoin par la suite mais cette **fois-ci à l'intérieur du process**, parce qu'après passage par chaque bloc parmi ceux dont on a parlé auparavant (bloc permutation, bloc XOR etc.) on obtiendra à chaque fois une nouvelle trame qu'on affecte à la variable correspondante par exemple Pour la donnée E(entée) 64bits qu'on passe par la permutation initiale PI on l'affecte ensuite à une nouvelle variable nommé E_purm etc.

La figure au-dessous montre le schéma de la 1eme itération, entre bloc et bloc ils sont écrits les variables circulant dans la ligne on rouge et leurs tailles en vers :



2.2. Instructions dans process :

2.2.1. Avant la boucle principale de 16 itérations :

Au niveau de la partie I (description de l'algorithme DES) on a dit qu'il faut passer la donnée par une permutation initiale ensuite la divisé sur deux partie R0 L0. Et qu'on doit passer la clé principale K par une permutation CP1 afin de rendre sa taille de 56bits et la divisé aussi sur deux partie g0 et d0 Comme il est montré sur la figure au-dessus :

```
106     begin
107
108        -- Permutation de la donnée d'entrée E :
109        E_purm  :=  E(57) & E(49) & E(41) & E(33) & E(25) & E(17) & E(9)  & E(1) &
110                    E(59) & E(51) & E(43) & E(35) & E(27) & E(19) & E(11) & E(3) &
111                    E(61) & E(53) & E(45) & E(37) & E(29) & E(21) & E(13) & E(5) &
112                    E(63) & E(55) & E(47) & E(39) & E(31) & E(23) & E(15) & E(7) &
113                    E(56) & E(48) & E(40) & E(32) & E(24) & E(16) & E(8)  & E(0) &
114                    E(58) & E(50) & E(42) & E(34) & E(26) & E(18) & E(10) & E(2) &
115                    E(60) & E(52) & E(44) & E(36) & E(28) & E(20) & E(12) & E(4) &
116                    E(62) & E(54) & E(46) & E(38) & E(30) & E(22) & E(14) & E(6) ;
117
118        -- Division de la donnée après permutation en deux parties R0 et L0
119        L0 := E_purm(63 downto 32);
120        R0:= E_purm(31 downto 0);
121
122        -- Permutation de la clé principal
123        K_CP1 :=    K(56) & K(48) & K(40) & K(32) & K(24) & K(16) & K(8)  &
124                    K(0)  & K(57) & K(49) & K(41) & K(33) & K(25) & K(17) &
125                    K(9)  & K(1)  & K(58) & K(50) & K(42) & K(34) & K(26) &
126                    K(18) & K(10) & K(2)  & K(59) & K(51) & K(43) & K(35) &
127                    K(62) & K(54) & K(46) & K(38) & K(30) & K(22) & K(14) &
128                    K(6)  & K(61) & K(53) & K(45) & K(37) & K(29) & K(21) &
129                    K(13) & K(5)  & K(60) & K(52) & K(44) & K(36) & K(28) &
130                    K(20) & K(12) & K(4)  & K(27) & K(19) & K(11) & K(3);
131
132        ---Division de la clé  après permutation sur deux parties g0 et d0
133        g0 := K_CP1(55 downto 28);
134        d0 := K_CP1(27 downto 0);
135
136        -- La boucle principal de 16 itérations
137        for j in 0 to 15 loop
```

Figure 98 : Intérieur du process avant la boucle de 16 itérations.

2.2.2. L’intérieure de la boucle principale (16 itérations) :

```
136     -- La boucle principal de 16 itérations
137     for j in 0 to 15 loop
138
139
140     -- Obtention de la nouvelle partie left (L1 pour la première itération)
141     L1 := R0;
142
143     -- Expansion de R0
144     R_Exp :=    R0(31) & R0(0)  & R0(1 ) & R0(2 ) & R0(3)  & R0(4)  &
145                 R0(3)  & R0(4)  & R0(5)  & R0(6)  & R0(7)  & R0(8)  &
146                 R0(7)  & R0(8)  & R0(9)  & R0(10) & R0(11) & R0(12) &
147                 R0(11) & R0(12) & R0(13) & R0(14) & R0(15) & R0(16) &
148                 R0(15) & R0(16) & R0(17) & R0(18) & R0(19) & R0(20) &
149                 R0(19) & R0(20) & R0(21) & R0(22) & R0(23) & R0(24) &
150                 R0(23) & R0(24) & R0(25) & R0(26) & R0(27) & R0(28) &
151                 R0(27) & R0(28) & R0(29) & R0(30) & R0(31) & R0(0);
152
153     -- Décalage d'un cran à gauche des deux parties (g0 et d0 )
154     g1 := g0(26 downto 0) & g0(27) ;
155     d1 := d0(26 downto 0) & d0(27) ;
156
157     -- concaténation des deux partie g0 et d0 mais après decalage
158     l_r_con := g1 & d1 ;
159
160     -- generation de la clé (K1 pour la lèr itération) on passant les deux parties concatenées  par la matrice CP1
161     K1 :=       l_r_con(13) & l_r_con(16) & l_r_con(10) & l_r_con(23) & l_r_con(0)  & l_r_con(4)  &
162                 l_r_con(2)  & l_r_con(27) & l_r_con(14) & l_r_con(5)  & l_r_con(20) & l_r_con(9)  &
163                 l_r_con(22) & l_r_con(18) & l_r_con(11) & l_r_con(3)  & l_r_con(25) & l_r_con(7)  &
164                 l_r_con(15) & l_r_con(6)  & l_r_con(26) & l_r_con(19) & l_r_con(12) & l_r_con(1)  &
165                 l_r_con(40) & l_r_con(51) & l_r_con(30) & l_r_con(36) & l_r_con(46) & l_r_con(54) &
166                 l_r_con(29) & l_r_con(39) & l_r_con(50) & l_r_con(44) & l_r_con(32) & l_r_con(47) &
167                 l_r_con(43) & l_r_con(48) & l_r_con(38) & l_r_con(55) & l_r_con(33) & l_r_con(52) &
168                 l_r_con(45) & l_r_con(41) & l_r_con(49) & l_r_con(35) & l_r_con(28) & l_r_con(31);
169
170      -- un XOR de R0 après expansion et la clé générer precedement
171      XOR_res:= k1 xor R_Exp;
172
173
```

Figure 109 : Intérieur de la boucle de 16 itérations.

```
      -- passange du résultat de l'XOR par la substitution afin d'avoir une trame réduite de 32bits
      --donc substitution du résultat de l'XOR (48bits) sur les 8 S-box (8 itération)
         for i in 0 to 7 loop

           -- Extraire chaque groupe de 6 bits
           groupe := XOR_res(47 - i*6  downto 42 - 6*i);

           -- Utilisation des deux premiers bits comme index de ligne
           row := conv_integer(groupe(5) & groupe(0));

           -- Utilisation des quatre bits du milieu comme index de colonne
           col := conv_integer(groupe(4 downto 1));

           -- Accéder à la valeur dans la S-box correspondante
           sbox_value := S_Boxes(i, row, col) ;

           -- Concaténer les valeurs obtenues à partir des S-boxes
           temp(31 - 4*i downto 28 - 4*i) := sbox_value;

         end loop;

       -- Affecter le résultat à la sortie
       XOR_sub := temp;

      -- permutation du résultat de la sortie
      XOR_pur:= XOR_sub(15) & XOR_sub(6)  & XOR_sub(19) &  XOR_sub(20) &
                XOR_sub(28) & XOR_sub(11) & XOR_sub(27) &  XOR_sub(16) &
                XOR_sub(0)  & XOR_sub(14) & XOR_sub(22) &  XOR_sub(25) &
                XOR_sub(4)  & XOR_sub(17) & XOR_sub(30) &  XOR_sub(9)  &
                XOR_sub(1)  & XOR_sub(7)  & XOR_sub(23) &  XOR_sub(13) &
                XOR_sub(31) & XOR_sub(26) & XOR_sub(2)  &  XOR_sub(8)  &
                XOR_sub(18) & XOR_sub(12) & XOR_sub(29) &  XOR_sub(5)  &
                XOR_sub(21) & XOR_sub(10) & XOR_sub(3)  &  XOR_sub(24);

      -- XOR du résultat de la sortie après permutation  avec L0
      R0 := XOR_pur xor L0;
      -- le résultat ce l'XOR c'est donne R1,R2...R16 , il est affecter à R0 pour reprendre l'itération sur R1 ensuite R2 ect


      -- conservation de la valeur de L1 dans R0
      L0 := L1;
```

Figure 20 : Intérieur de la boucle de 16 itérations (Suite).

```
       -- conservation de la valeur de L1 dans R0
       L0 := L1;

       -- affectation des parties décalées pour remprendre le décalage encore afin de generer une nouvelle clé à chaque itération
       g0 := g1;
       d0 := d1;

       end loop;

        --concaténation de L16 et R16 après la fin des 16 itérations
       S_inv   := L1 & R0;


       --passage des parties concatener par une permutation initiale inverse et obtention de la donnée chiffré (S)
       S <= S_inv(47) & S_inv(7) & S_inv(47) & S_inv(15) & S_inv(55) & S_inv(23) & S_inv(63) & S_inv(31)&
            S_inv(38) & S_inv(6) & S_inv(46) & S_inv(14) & S_inv(54) & S_inv(22) & S_inv(62) & S_inv(30)&
            S_inv(37) & S_inv(5) & S_inv(45) & S_inv(13) & S_inv(53) & S_inv(21) & S_inv(61) & S_inv(29)&
            S_inv(36) & S_inv(4) & S_inv(44) & S_inv(12) & S_inv(52) & S_inv(20) & S_inv(60) & S_inv(28)&
            S_inv(35) & S_inv(3) & S_inv(43) & S_inv(11) & S_inv(51) & S_inv(19) & S_inv(59) & S_inv(27)&
            S_inv(34) & S_inv(2) & S_inv(42) & S_inv(10) & S_inv(50) & S_inv(18) & S_inv(58) & S_inv(26)&
            S_inv(33) & S_inv(1) & S_inv(41) & S_inv(9)  & S_inv(49) & S_inv(17) & S_inv(57) & S_inv(25)&
            S_inv(32) & S_inv(0) & S_inv(40) & S_inv(8)  & S_inv(48) & S_inv(16) & S_inv(56) & S_inv(24) ;

   end process;
end DISCP;
```

Figure 211: Intérieur de la boucle de 16 itérations et fin de la boucle.

Maintenant et **dans la boucle** on procède sur **R0** et **L0** obtenue précédemment (ailleurs la boucle) afin d'avoir **R1**et **L1** pour la 1ème itération ensuite **R2**et **L2** pour la **2ème itération** …etc. Dans un premier temp pour trouver **L1** on leur **affecte R0**, chose qui est faite dans la boucle comme il est montré dans la figure au-dessus (ligne 141) ensuite et pour avoir R1 il faut **passer** par **tous les blocs** qu'on a cité auparavant Jusqu'à le dernier bloc de l'XOR qui consiste en un XOR entre XOR_pur et L0 Normalement le résultat de ce bloc **représente** le R1 qu'on a **affecter à** R0 **directement** Pour reprendre la procédure Avec R0 (R0 contient le contenu de R1)Et pour **garder le contenu** de **L1** qu'on aura besoin dans **la 2ème itération** on **l'affecte à L0** à la fin de la boucle, on affecte également g1 et d1 à g0 et d0 pour décaler une deuxième fois les deux partie afin de générer la clé K2 dans la 2ème itération etc.

Chapitre III : Simulation.

La figure au-dessous montre la bonne compilation du Code VHDL tout entier expliqué dans le chapitre II :

La simulation est réalisée sur Quartus v9 ;

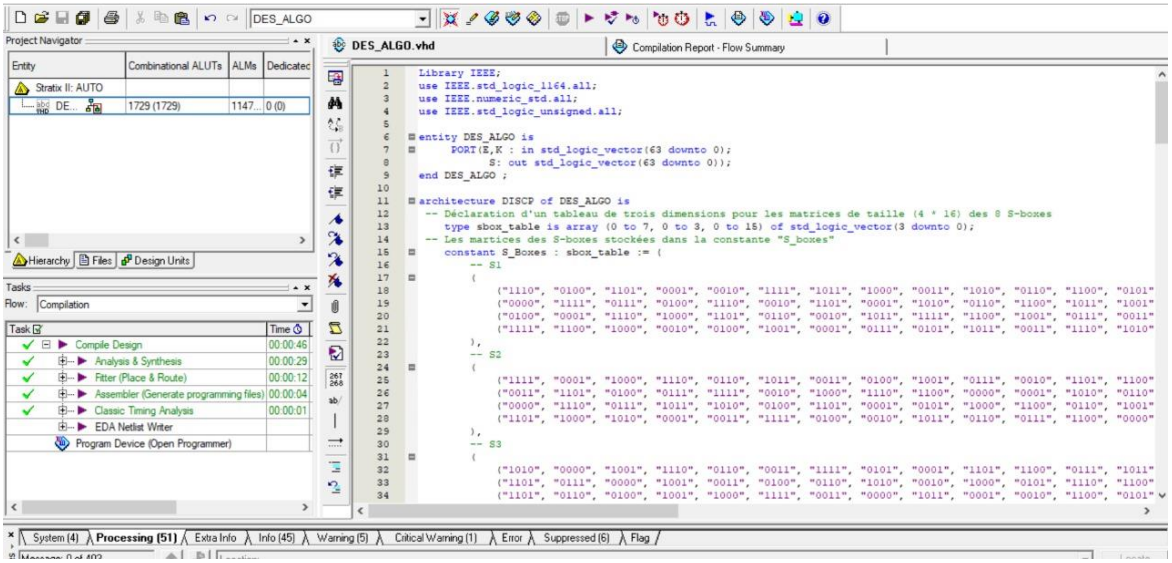


Figure 22 : Illustration de la compilation du programme.

La figure au-dessous représente la simulation du code avec des suites binaires de 64bits.

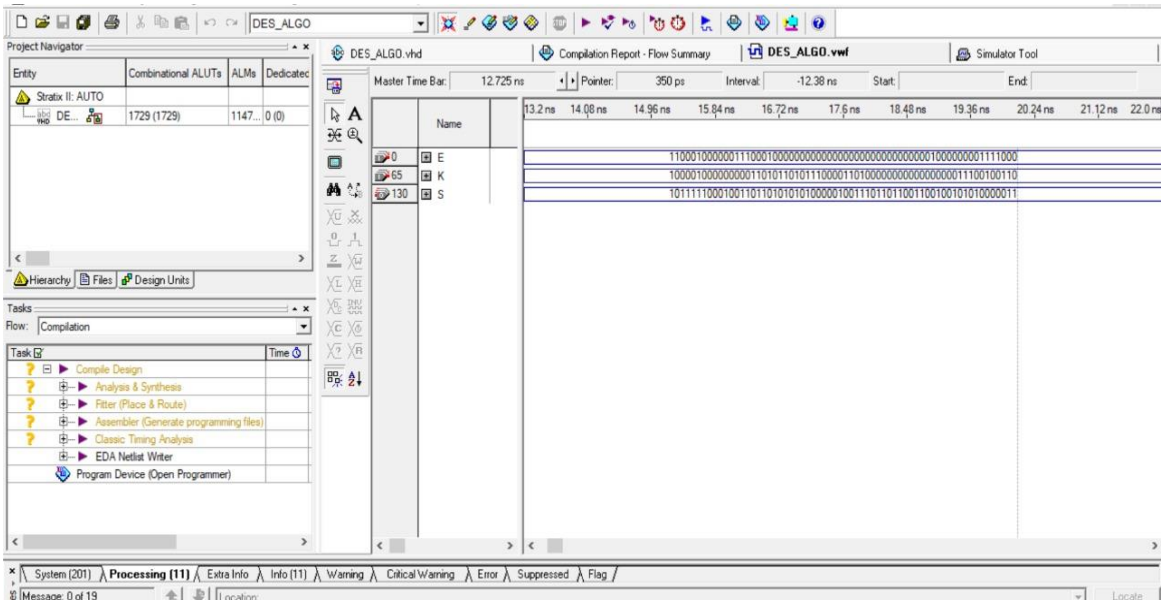


Figure 23 : Illustration de la simulation du code.

La figure au-dessous représente la simulation du code, la donnée et la clé cette fois-ci en Hexadécimale (16 symboles)

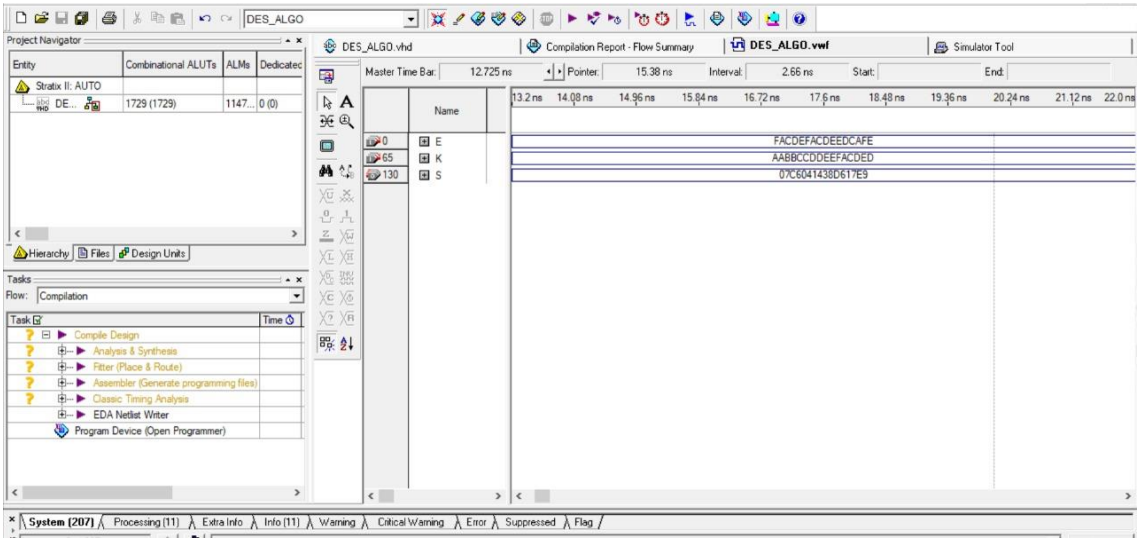


Figure 24 : Illustration de la simulation du code.

Figure 25 : Illustration de la simulation du code.

Conclusion :

Après l'étude et la réalisation du code VHDL pour l'algorithme DES, nous avons réussi à obtenir une implémentation fonctionnelle et efficace du cryptage. Ce projet nous a permis de comprendre en profondeur le fonctionnement du DES et d'acquérir une maîtrise solide du langage VHDL. Nous avons démontré la faisabilité et la performance de l'algorithme DES dans un environnement matériel. Cette expérience enrichissante a consolidé nos compétences en cryptographie et en conception matérielle, tout en soulignant l'importance de la sécurité dans le traitement des informations sensibles.