

# Projet PCSN:

# Chiffrement AES



BOUAGOUN Abdellah

Promo EI18



[1]

# Sommaire.

I.	Introduction: .....	4
1.	C'est quoi AES: .....	4
2.	Fonctionnement de AES : .....	4
3.	Structure de l'implémentation du AES en VHDL : .....	4
II.	Implémentation du AES en VHDL : .....	6
1.	SubBytes : .....	6
2.	ShiftRows : .....	7
3.	MixColumns: .....	8
a.	MultiplyMatrix.....	8
b.	Mixcolumns.....	9
4.	AddRoundKey.....	10
5.	KeyExpansion : .....	11
6.	La machine d'état (FSM_AES): .....	12
7.	AESRound : .....	14
8.	AES Complet : .....	15
III.	Conclusion : .....	17
IV.	Références : .....	18

## Liste des Figures:

Figure 1 Chiffrement AES 128bits .....	5
Figure 2: La table S-Box de substitution.....	6
Figure 3: Simulation de la SBox .....	6
Figure 4: Simulation de SubBytes .....	7
Figure 5: Rotation des lignes de la matrice .....	7
Figure 6: Simulation de ShiftRows .....	8
Figure 7: Transformation linéaire sur une colonne .....	8
Figure 8 : Architecture de MultiplyMatrix.....	9
Figure 9: Simulation de MultiplyMatrix .....	9
Figure 10: Simulation de MixColumns .....	10
Figure 11: Simulation de AddRoundKey.....	10
Figure 12: la machine d'état de Moore.....	12
Figure 13: la machine d'état implémentée .....	12
Figure 14: Simulation de la FSM .....	13
Figure 15: Schéma de AESRound .....	14
Figure 16: Simulation du passage de la Ronde_0 à la Ronde_1 .....	14
Figure 17: Schéma de AES complet .....	15
Figure 18: Simulation du AES complet .....	16

# I. Introduction:

## 1. C'est quoi AES:

Advanced Encryption Standard ou AES est un algorithme de chiffrement symétrique. Il a gagné en octobre 2000 le concours AES, lancé par le NIST pour choisir le nouveau standard de chiffrement pour les organisations du gouvernement des États-Unis. Actuellement, il est le plus utilisé et le plus sûr algorithme cryptographique.

## 2. Fonctionnement de AES :

L'algorithme prend en entrée un bloc de 128 bits (16 octets), la clé peut être 128, 192 ou 256 bits.

Les opérations majeures effectuées par l'AES sont les suivantes :

- 1- Substituer les 16 octets en entrée selon une table définie au préalable.
- 2- Placer les octets obtenus dans une matrice de 4x4 éléments et appliquer une rotation vers la droite sur ses lignes. L'incrément pour la rotation varie selon le numéro de la ligne.
- 3- Appliquer sur la matrice Une transformation linéaire qui consiste en la multiplication binaire de chaque élément de la matrice avec des polynômes issus d'une matrice auxiliaire, cette multiplication est soumise à des règles spéciales selon GF(28) (groupe de Galois ou corps fini). L'avantage de cette transformation linéaire est qu'elle garantit une meilleure diffusion (propagation des bits dans la structure) sur plusieurs tours.
- 4- un OU exclusif XOR entre la matrice et une autre matrice (clés générées à partir de la clé initiale) permet d'obtenir une matrice intermédiaire.

Ces différentes opérations sont répétées plusieurs fois et définissent un « tour ». Pour une clé de 128, 192 ou 256, AES nécessite respectivement 10, 12 ou 14 tours.

## 3. Structure de l'implémentation du AES en VHDL :

Pour implémenter AES, nous allons décrire en VHDL 4 composants électroniques tel que chacun d'eux représente une des opérations présentées dans la partie précédente. Ces composants sont :

SubBytes : il permet la substitution des octets d'entrée selon une table déjà définie.

ShiftRows : il permet d'appliquer des rotations sur les lignes de la matrice créée à partir les octets obtenus de la substitution.

AddRoundKey : Ce composant fait le XOR entre la matrice et la matrice représentante de la clé intermédiaire.

MixColumns : Ce composant applique sur la matrice la transformation linéaire déjà mentionnée dans l'opération 3.

L'algorithme va utiliser ces composants répétitivement dans les rondes suivantes :

- Une ronde initiale : Elle utilise seulement la fonction AddRoundKey.
- 9 rondes intermédiaires qui utilisent toutes les composants développées.
- Une ronde finale : Elle utilise toutes les fonctions sauf MixColumns.

En sortie de la ronde 10 (finale), on obtient le texte crypté par AES.

Voilà un schéma qui représente le chiffrement AES 128 que nous allons implémenter en VHDL :

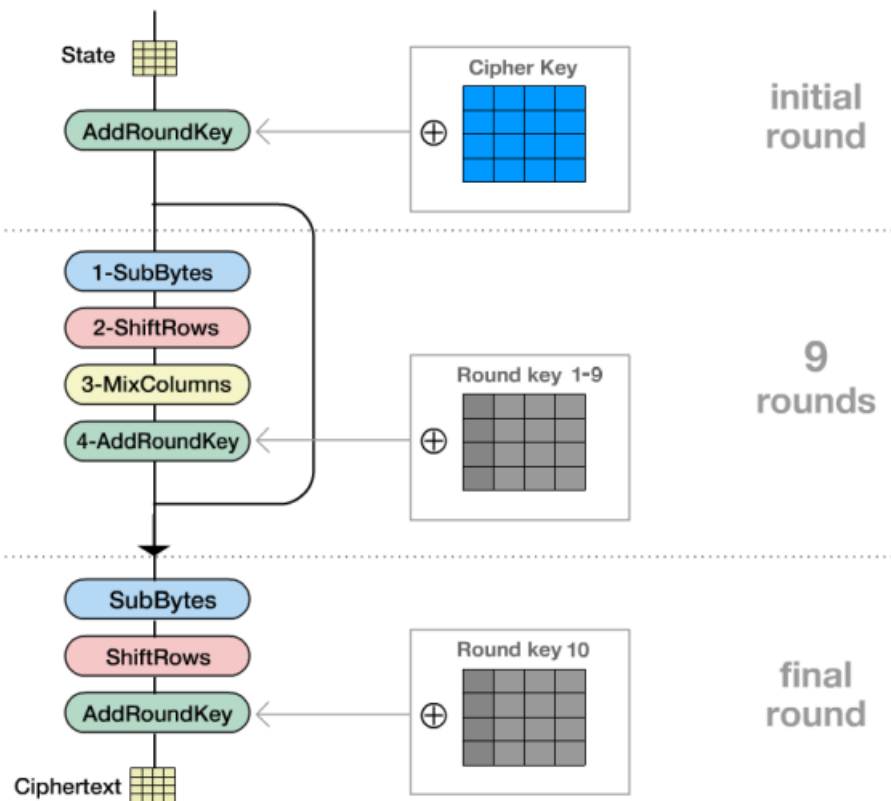


Figure 1 Chiffrement AES 128bits

## II. Implémentation du AES en VHDL :

### 1. SubBytes :

Cette fonction consiste en une transformation non linéaire appliquée à tous les octets de l'état en utilisant une table de substitution appelée S-Box (figure 2).

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Figure 2: La table S-Box de substitution

Concernant l'implémentation, ce composant a en entrée une matrice 4x4 et donne aussi en sortie une matrice de même taille .

Voila les résultats de simulation pour le composant Sbox crée pour faciliter l'implémentation de SubBytes, et puis pour le composant SubBytes :

Wave		Msgs			
+	/sbox_tb/input	AB	01	EF	AB
	/sbox_tb/output	0E	09	61	0E

Figure 3: Simulation de la SBox

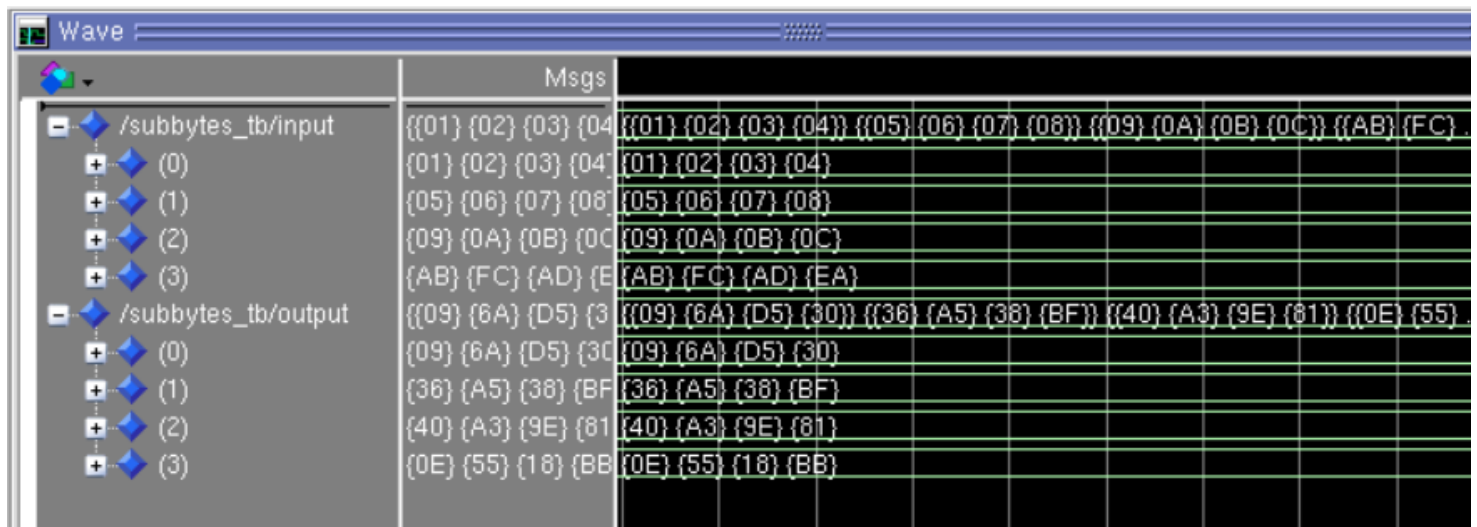


Figure 4: Simulation de SubBytes

## 2. ShiftRows :

La fonction ShiftRows applique des rotations des octets des lignes de l'état. Le décalage des octets correspond à l'indice de la ligne considérée ( $0 \leq r \leq 3$ ).

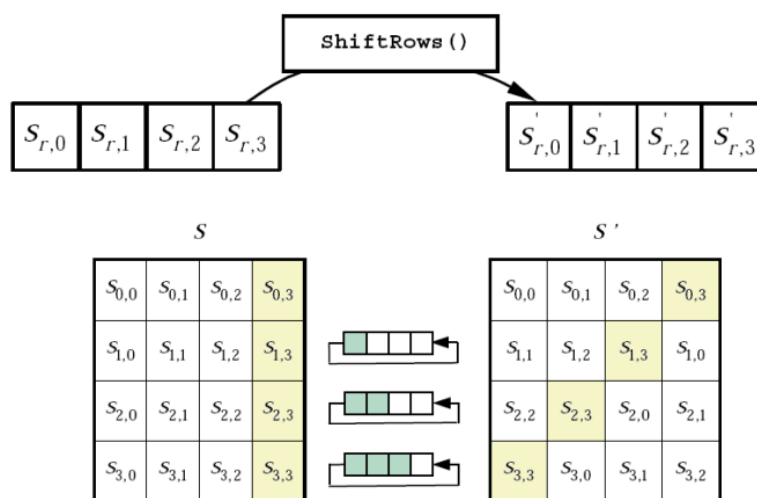


Figure 5: Rotation des lignes de la matrice

Voila les résultats de simulation du composant :

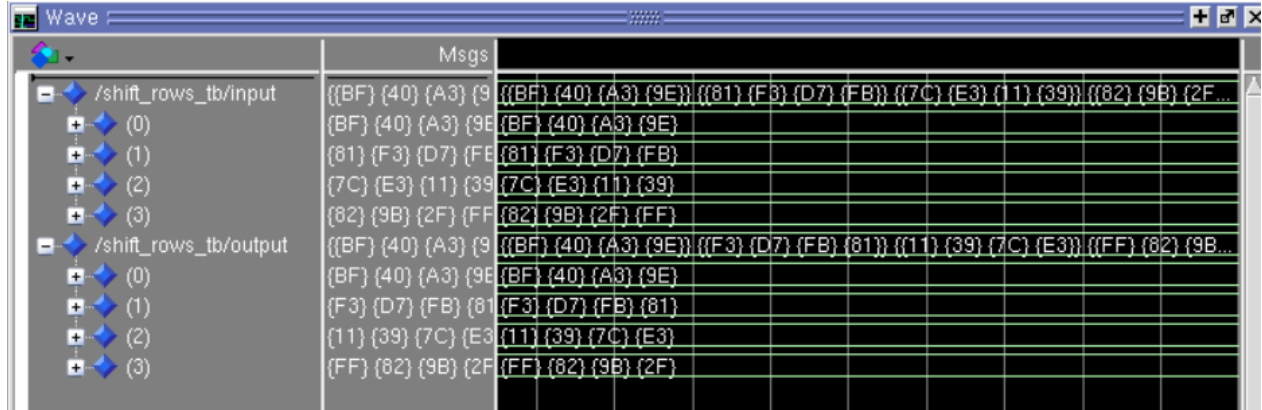


Figure 6: Simulation de ShiftRows

### 3. MixColumns:

#### a. MultiplyMatrix

J'ai implémenté le composant MultiplyMatrix pour faciliter l'implémentation de MixColumns, ce composant applique la transformation linéaire décrit ci-dessous sur une seule colonne.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

$$\begin{aligned}
 s'_{0,c} &= (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \\
 s'_{1,c} &= s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c} \\
 s'_{2,c} &= s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c}) \\
 s'_{3,c} &= (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c})
 \end{aligned}$$

Figure 7: Transformation linéaire sur une colonne



Voila la description de ce fonctionnement en code VHDL :

```
x : for i in 0 to 3 generate
    col1_s(i) <= (data_i(i)(6 downto 0) & '0') xor "00011011" when data_i(i)(7) = '1'
    else data_i(i)(6 downto 0) & '0';
end generate;

y: for i in 0 to 3 generate
    col2_s(i) <= col1_s(i) xor data_i(i);
end generate;

data_o(0) <= col1_s(0) xor col2_s(1) xor data_i(2) xor data_i(3);
data_o(1) <= data_i(0) xor col1_s(1) xor col2_s(2) xor data_i(3);
data_o(2) <= data_i(0) xor data_i(1) xor col1_s(2) xor col2_s(3);
data_o(3) <= col2_s(0) xor data_i(1) xor data_i(2) xor col1_s(3);
```

Figure 8 : Architecture de MultiplyMatrix

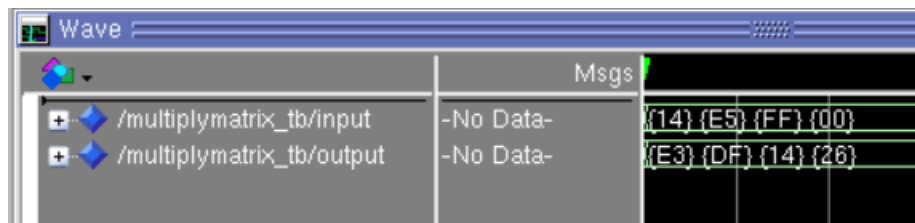
On met dans le signal col1\_s le résultat de la multiplication de chaque élément de la colonne d'entrée par (02), ce qui revient à faire l'opération suivante :

- Si le 7<sup>ème</sup> bit de poids fort est '1' => faire un shift vers la gauche d'un seul bit, puis appliquer le XOR entre le résultat du shift et la valeur « 00011011 ».
- Sinon on fait seulement l'opération shift vers la gauche d'un seul bit.

Pour le signal col2\_s, on y met le résultat de la multiplication de chaque élément de la colonne d'entrée par (03), ce qui revient à faire le XOR entre le contenu de col1\_s et la colonne d'entrée.

Enfin, on affecte à chaque élément de data\_o sa valeur en suivant les calculs expliqués dans la figure 7.

La simulation de MultiplyMatrix donne les résultats suivants :



Signal	Value
/multiplymatrix_tb/input	-No Data-
/multiplymatrix_tb/output	-No Data-

Msgs
{14} {E5} {FF} {00}
{E3} {DF} {14} {26}

Figure 9 : Simulation de MultiplyMatrix

## b. Mixcolumns

En utilisant 4 instances du composant MatrixMultiply ,on peut calculer le produit de chaque colonne de la matrice d'entrée et la matrice fixe pour obtenir le résultat de la transformation linéaire.

Voilà les résultats de la simulation

Signal	Cycle	Value
/muxcolumns_tb/input	(0)	{AF} {16} {CE} {E6} {91} {62} {44} {01} {06} {D3} {20} {BC} {E0} {81} {FC}
	(1)	{AF} {16} {CE} {BC} {E6} {91} {62} {44} {01} {06} {D3} {20} {BC} {E0} {81} {FC}
	(2)	{E6} {91} {62} {44} {01} {06} {D3} {20} {BC} {E0} {81} {FC} {AF} {16} {CE} {BC}
	(3)	{D5} {AB} {B1} {AE} {A0} {29} {43} {21} {AE} {8E} {D5} {FA} {2F} {6D} {D9} {51}
/muxcolumns_tb/output	(0)	{A0} {29} {43} {21} {AE} {8E} {D5} {FA} {2F} {6D} {D9} {51} {BC} {E0} {81} {FC}
	(1)	{AE} {8E} {D5} {FA} {2F} {6D} {D9} {51} {BC} {E0} {81} {FC} {A0} {29} {43} {21}
	(2)	{2F} {6D} {D9} {51} {BC} {E0} {81} {FC} {A0} {29} {43} {21} {AE} {8E} {D5} {FA}
	(3)	{BC} {E0} {81} {FC} {A0} {29} {43} {21} {AE} {8E} {D5} {FA} {2F} {6D} {D9} {51}

Figure 10: Simulation de MixColumns

#### 4. AddRoundKey

Ce composant applique le XOR bit à bit entre les octets de la matrice 4x4 et les octets de la clef de ronde (Ronde 0 à 10).

Voilà les résultats de simulation du composant :

Signal	Cycle	Value
/addroundkey_tb/input	(0)	{BF} {40} {A3} {9E} {81} {F3} {D7} {FB} {7C} {E3} {11} {39} {82} {9B} {2F} {FF}
	(1)	{BF} {40} {A3} {9E} {81} {F3} {D7} {FB} {7C} {E3} {11} {39} {82} {9B} {2F} {FF}
	(2)	{81} {F3} {D7} {FB} {7C} {E3} {11} {39} {82} {9B} {2F} {FF} {BF} {40} {A3} {9E}
	(3)	{7C} {E3} {11} {39} {82} {9B} {2F} {FF} {BF} {40} {A3} {9E} {81} {F3} {D7} {FB}
/addroundkey_tb/key	(0)	{FF} {20} {A3} {9E} {71} {F3} {D7} {FB} {7A} {E1} {11} {39} {82} {9B} {2F} {FF}
	(1)	{FF} {20} {A3} {9E} {71} {F3} {D7} {FB} {7A} {E1} {11} {39} {82} {9B} {2F} {FF}
	(2)	{71} {F3} {D7} {FB} {7A} {E1} {11} {39} {82} {9B} {2F} {FF} {FF} {20} {A3} {9E}
	(3)	{7A} {E1} {11} {39} {82} {9B} {2F} {FF} {FF} {20} {A3} {9E} {71} {F3} {D7} {FB}
/addroundkey_tb/output	(0)	{40} {60} {00} {00} {F0} {00} {00} {00} {06} {02} {00} {00} {00} {00} {00} {00}
	(1)	{40} {60} {00} {00} {F0} {00} {00} {00} {06} {02} {00} {00} {00} {00} {00} {00}
	(2)	{F0} {00} {00} {00} {06} {02} {00} {00} {00} {00} {00} {00} {00} {00} {00} {00}
	(3)	{06} {02} {00} {00} {00} {00} {00} {00} {00} {00} {00} {00} {00} {00} {00} {00}

Figure 11: Simulation de AddRoundKey

## 5. KeyExpansion :

Pour la ronde 0, la clef utilisée est la clef de chiffrement alors que les clefs des rondes suivantes (de 1 à 10) sont issues d'un processus de diversification des clefs qui s'appelle 'Key Expansion'.

Ce processus fait appel à une fonction de rotation appliquée sur une colonne (RotWord), il utilise également la table de substitution (SubBytes) et la fonction booléenne XOR. L'algorithme qui représente ce processus est le suivant :

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
    word temp

    i = 0

    while (i < Nk)
        w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
        i = i+1
    end while

    i = Nk

    while (i < Nb * (Nr+1))
        temp = w[i-1]
        if (i mod Nk = 0)
            temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
        else if (Nk > 6 and i mod Nk = 4)
            temp = SubWord(temp)
        end if
        w[i] = w[i-Nk] xor temp
        i = i + 1
    end while
end
```

Dans notre cas, au lieu d'implémenter un composant qui va générer les clés pour chaque Ronde, nous allons utiliser une table fixe qui contient les différentes clés que l'AES va utiliser dans chaque Ronde.

## 6. La machine d'état (FSM\_AES):

Pour passer d'une ronde à une autre, on a opté à l'implémentation d'une machine d'état de type Moore

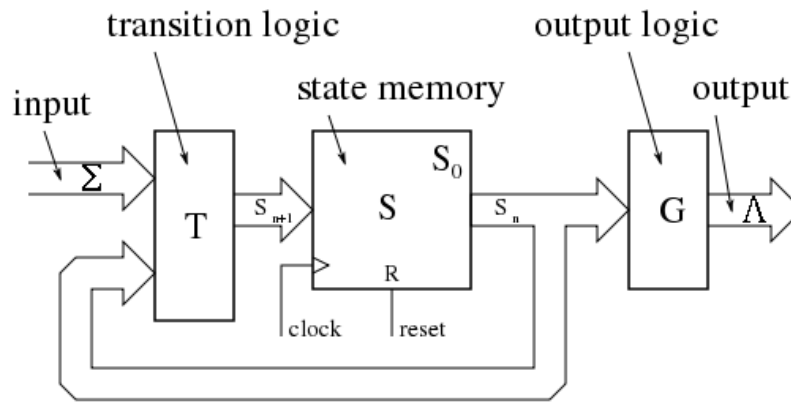


Figure 12: la machine d'état de Moore

La FSM que nous avons implémenter est la suivante :

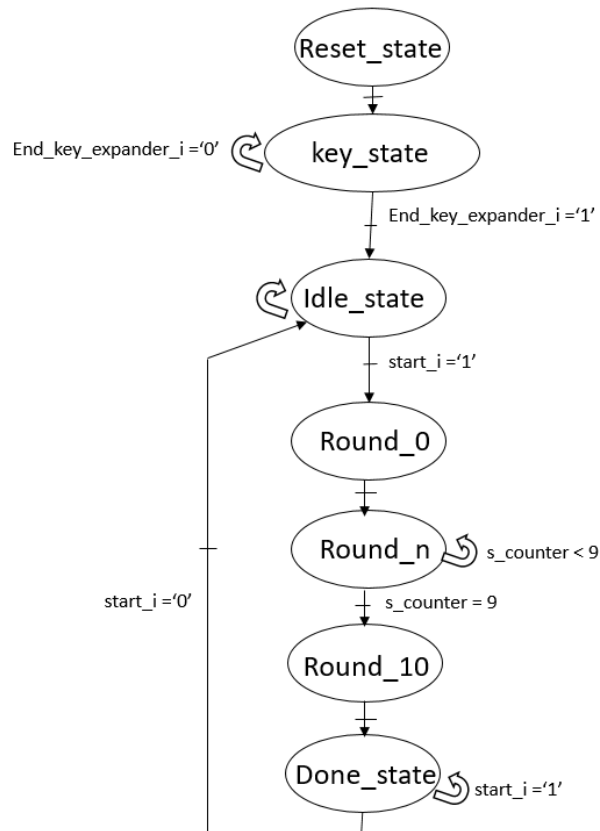


Figure 13: la machine d'état implémentée

## Explication des états :

Reset\_state : c'est l'état d'initialisation.

Key\_state : c'est l'état dans laquelle le composant key\_expansion\_I\_O\_table donne la bonne clé pour l'état présente.

Idle\_state : on attend que start\_i passe à '1' pour commencer les calculs des rondes.

Round\_0 : la ronde initiale dans laquelle on fait seulement le AddRoundKey.

Round\_n : c'est la ronde intermédiaire, on distingue entre les 9 rondes intermédiaires en utilisant un signal interne comme compteur (s\_counter).

On utilise dans ces rondes les quatre composants pour faire les calculs.

Round\_10 : dans la ronde finale, tous les composants sont utilisés sauf MixColumns.

Done\_state : dans l'état de fin, on met enableOutput\_o à '1' pour avoir le résultat final à la sortie data\_o.

La simulation de la FSM est la suivante :

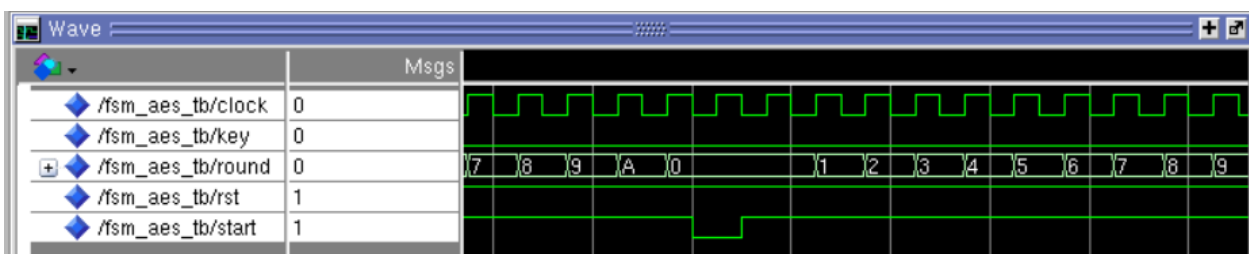


Figure 14: Simulation de la FSM

## 7. AESRound :

Ce circuit assemble tous les composants déjà créés pour faire le calcul d'une seule ronde. Voila le schéma de son implémentation :

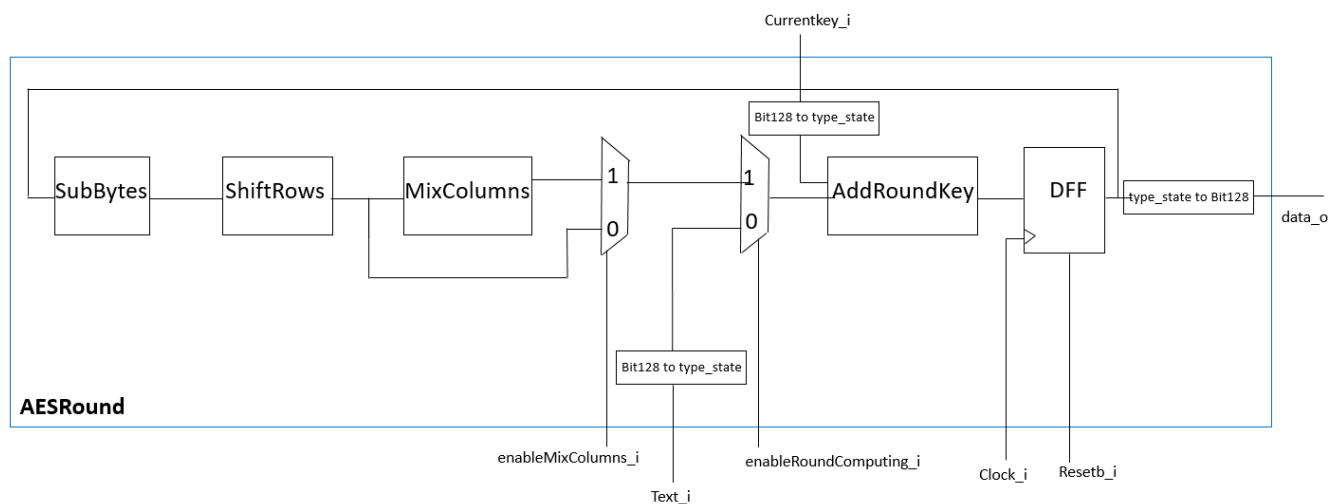


Figure 15: Schéma de AESRound

On a implémenté dans le code VHDL trois conversions de bit128 vers type\_state et également l'inverse.

La simulation de AESRound a donné les résultats suivants :

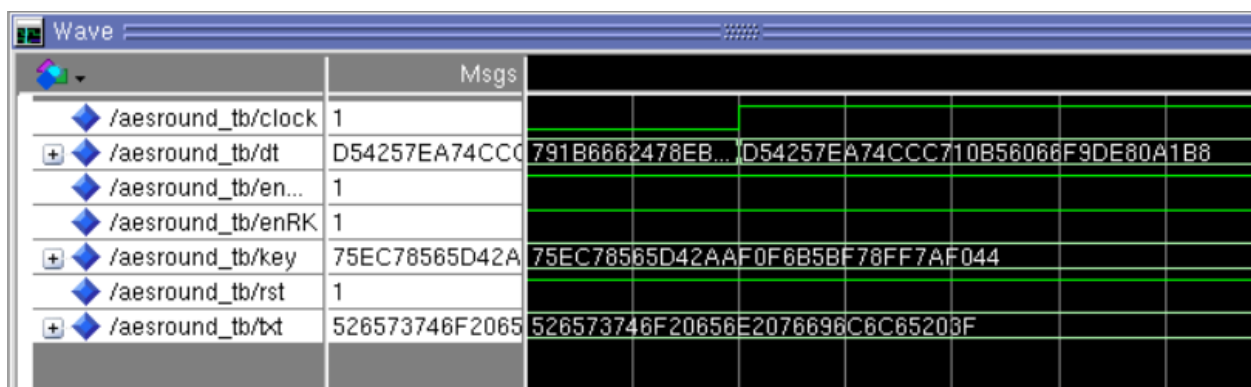


Figure 16: Simulation du passage de la Ronde\_0 à la Ronde\_1

## 8. AES Complet :

Pour obtenir l'implémentation complète de l'algorithme, on connecte KeyExpansion\_I\_O avec FSM\_AES et AESRound de la manière suivante :

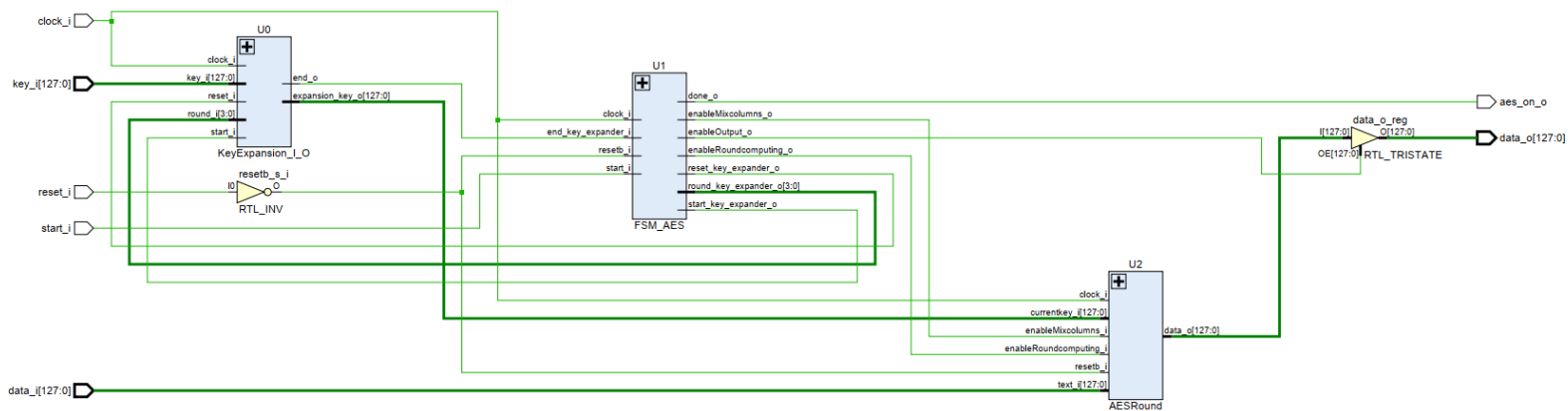


Figure 17: Schéma de AES complet

La simulation du AES complet nous a donné le texte crypté suivant :

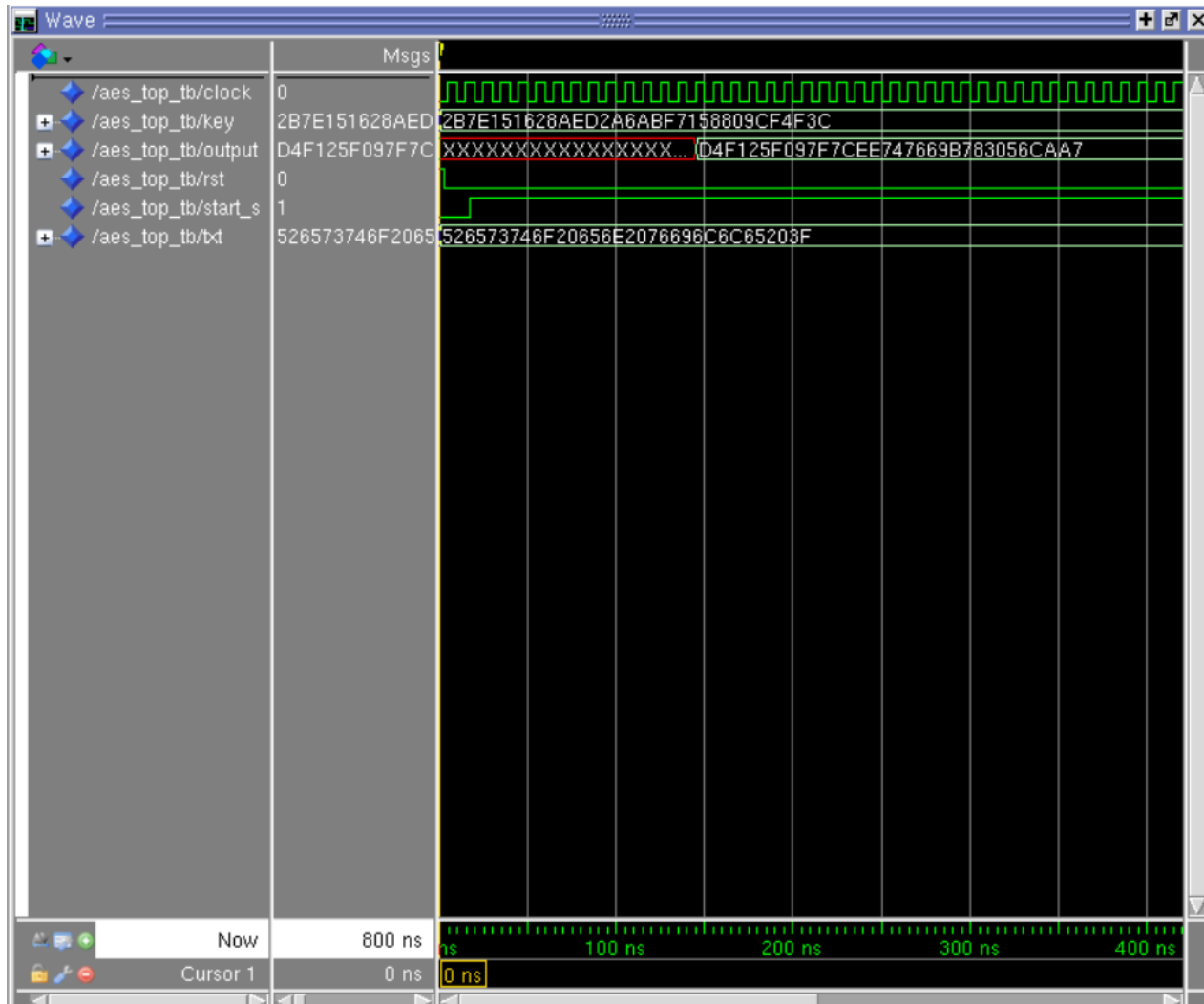


Figure 18: Simulation du AES complet



### III. Conclusion :

J'ai beaucoup apprécié ce projet car il m'a permis de renforcer mes capacités en VHDL qui est l'une des compétences indispensables pour un Microélectronicien .

Par ailleurs, l'implémentation matérielle d'un algorithme cryptographique comme AES qui est largement utilisé actuellement dans les appareils, m'a permis d'approfondir mes connaissances en Sécurité matérielle.

J'espère que j'aurai encore l'opportunité de faire d'autres projets dans le domaine de la sécurité matérielle et surtout la « Security by Design » au sein de l'école ou du labo SAS.

## IV. Références :

- [1] J. Boyar and R. Peralta, A Small Depth-16 Circuit for the AES S-Box, pp. 287–298. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012.
- [2] W. Stallings, Cryptography and Network Security : Principles and Practice. Upper Saddle River, NJ, USA : Prentice Hall Press, 5th ed., 2010.
- [3] NIST, “Fips-197, announcing the advanced encryption standard (aes),”
- [4] [https://fr.wikipedia.org/wiki/Advanced\\_Encryption\\_Standard](https://fr.wikipedia.org/wiki/Advanced_Encryption_Standard)