Rapport projet VHDL
Algorithme de déchiffrement AES 128bits

Loïc DELEFORTERIE

05/01/2019

Table des matières

Chapitre 1

Introduction

1.1 Présentation de l'AES

1.1.1 L'histoire de l'AES

L'AES (Advanced Encryption Standard) est un algorithme de chiffrement symétrique. Il a été conçu par deux belges : Joan DAEMEN et Vincent RIJMEN qui lui ont donné le nom de Rijndael. Cet algorithme de chiffrement a remporté l'appel à candidatures international lancé par le NIST (National Institute of Standards and Technology) en 1997.
Suite à ce prix, il devient l'AES et devient le nouveau standard en remplaçant le DES (Data Encryption Standard) datant des années 1970.
L'AES est adopté par le NIST en 2001.

1.1.2 Algorithme de chiffrement

Au niveau de l'algorithme de chiffrement, l'AES prend en entrée des blocs de 128 bits. Il existe plusieurs tailles de clé : 128, 192, 256 bits. La taille de la clé influence de nombre de tours que va effectuer l'algorithme afin de chiffrer un message. En effet pour une clé de 128 bits, l'agorithme va effectuer 10 tours, pour une clé de 192 bits il effectuera 12 tours et pour une clé de 256 bits 14 tours.
Lorsque un mot de 128 bits est donné en entrée de l'AES, l'algorithme va stocker ce mot dans une matrice carrée de taille 4, chaque case comporte donc un octet (8 bits).
Cette matrice va ensuite subir des transformations et ressortir de l'AES sous la forme d'un mot de 128 bits chiffré.
Afin d'expliquer le principe de l'algorithme, nous allons partir du principe que la taille de la clé

est de 128 bits se qui implique que l'algorithme effectuera 10 tours.
Dans l'algorithme il existe 4 transformations élèmentaires : AddRoundKey, SubBytes, ShiftRows et MixColumns.
Lors du premier tours, l'algorithme effectue seulement la transformation AddRoundKey. Du tour 2 au tour 9, l'algorithme effectue toutes les transformations. Lors du dernier tour, l'algorithme effectue seulement les transformations AddRoundKey, SubBytes et ShiftRows.
A la fin de chaque tours la matrice transformée est réinjectée dans le tour suivant.

1.1.3 Algorithme de déchiffrement

Au niveau de l'algorithme de déchiffrement, les fonctionnements de la taille des clés et le nombre de tours sont identiques au chiffrement.
Le fonctionnement de l'agorithme est similaire, il y a seulement les transformations élèmentaires qui changent. Les transformations sont : InvAddRoundKey, InvSubBytes, InvShiftRows et InvMixColumns. Les transformations de l'algorithme de déchiffrement sont les inverses des transformations de l'algorithme de chiffrement.
L'algorithme de déchiffrement fonctionne à l'identique de celui de chiffrement. Les transformations du chiffrement sont simplement remplacées par leurs inverses.
Voici le pseudo-code de l'algorithme de déchiffrement de l'AES :

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   for round = Nr-1 step -1 downto 1
      InvShiftRows(state)
      InvSubBytes(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      InvMixColumns(state)
   end for

   InvShiftRows(state)
   InvSubBytes(state)
   AddRoundKey(state, w[0, Nb-1])

   out = state
end
```

FIGURE 1.1 – Pseudo-code de l'algorithme de déchiffrage de l'AES

1.2 Présentation du projet VHDL 2018

L'objectif du projet est d'implémenter l'lagorithme de déchiffrage de l'AES avec une clé de 128 bits.
Chaque transformation aura donc son composant ainsi qu'un testbench afin de vérifier son fonctionnement.
Dans un premier temps, nous avons 5 séances acompagnés de professeur, l'objectif est de réaliser un composant par séance.
Après la dernière séance, nous avons 1 mois pour finaliser le projet ainsi que rédiger le rapport.

Chapitre 2

Implémentation du déchiffrage de l'AES

2.1 La transformation ExpansionKey

2.1.1 Présentation

Cette transformation prend en entrée le tour de l'algorithme et ressort une expansion de clé de 128 bits. L'expansion de clé est issue de la clé principale. Il existe une expansion de clé pour chaque tours (10 tours) plus la clé principale.

2.1.2 Implémenation

Cette transformation était fournie par les professeurs afin d'implémenter le déchiffrage de l'AES. Il ne reste qu'à tester si le composant fonctionne corectement. Voici le schéma à implémenter :

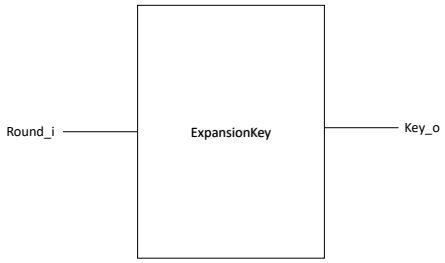

FIGURE 2.1 – Schéma du composant ExpansionKey

2.1.3 Résultats

Au niveau des résultats on a bien la bonne clé en fonction du nombre de tour.

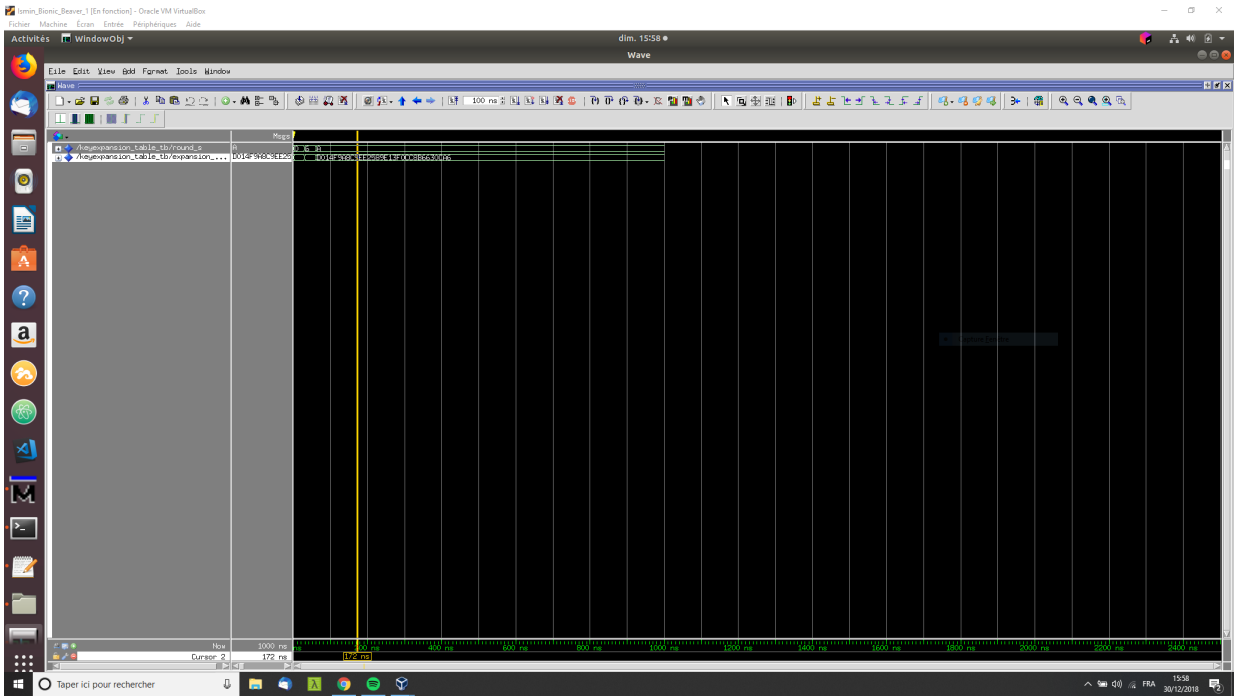

FIGURE 2.2 – Testbench de KeyExpansion

2.2 La transformation InvAddRoundKey

2.2.1 Présentation

Cette transformation permet d'ajouter la clé récupérée via le composant ExpansionKey au mot qui est en cours de traitement. Il suffit de faire un XOR entre le mot de 128 bits et la clé de 128 bits. Il y a donc deux entrées : le mot en cours de traitement et la clé et une seule sortie qui est la somme du mot et de la clé.

2.2.2 Implémenation

Ce composant est simple à implémenter étant donné que sa fonction est de réaliser simplement un xor entre deux mots de 128 bits.
Voici le schéma du composant :

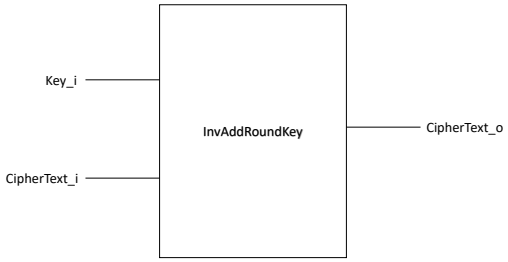

FIGURE 2.3 – Schéma du composant InvAddRoundKey

2.2.3 Résultats

Lors du test de cette transformation, nous avons bien en résultat le XOR des deux mots de 128 bits.

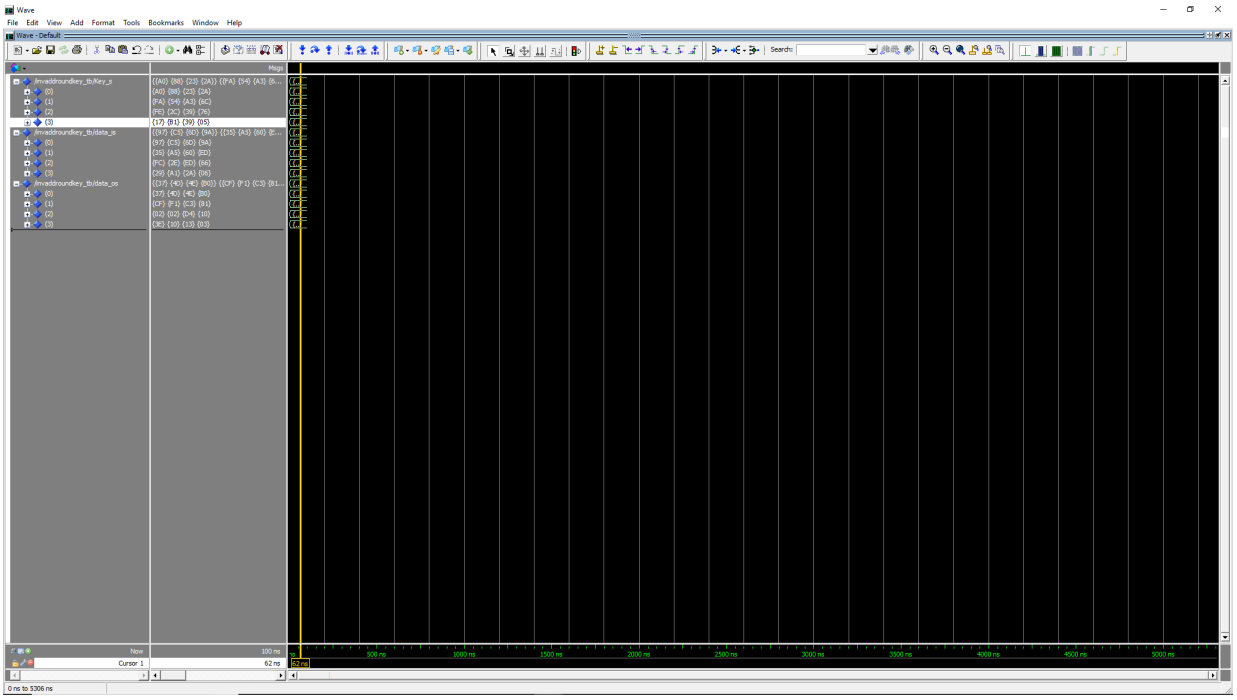

FIGURE 2.4 – Testbench de InvAddRoundKey

2.3 La transformation InvShiftRows

2.3.1 Présentation

La transformation InvShiftRows consiste à décaler certains élèments de la matrice. Étant donné que lors du chiffrage les décalages se font vers la gauche, les décalages pour le déchiffrage se feront vers la droite.
La première ligne reste inchangée étant donné que son décalage est de 0. Chaque élément de la ligne 1 est décalé de 1 vers la droite. Pour la ligne 2 et 3, le décalage est repectivement de 2 et 3 vers la droite.

2.3.2 Implémenation

Pour implémenter cette transformation, j'ai simplement réatribué toutes les valeurs de la matrice sauf pour la première ligne. Une amélioration aurait été de créer un modulo afin qu'une simple boucle effectue toutes les décalages. Voici le schéma à implémenter :

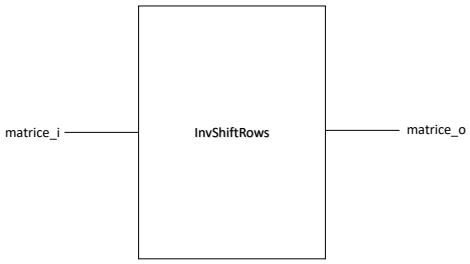

FIGURE 2.5 – Schéma du composant InvShiftRows

2.3.3 Résultats

Au niveau des résultats, on obtient bien le fonctionnement voulu. Voici la simulation :

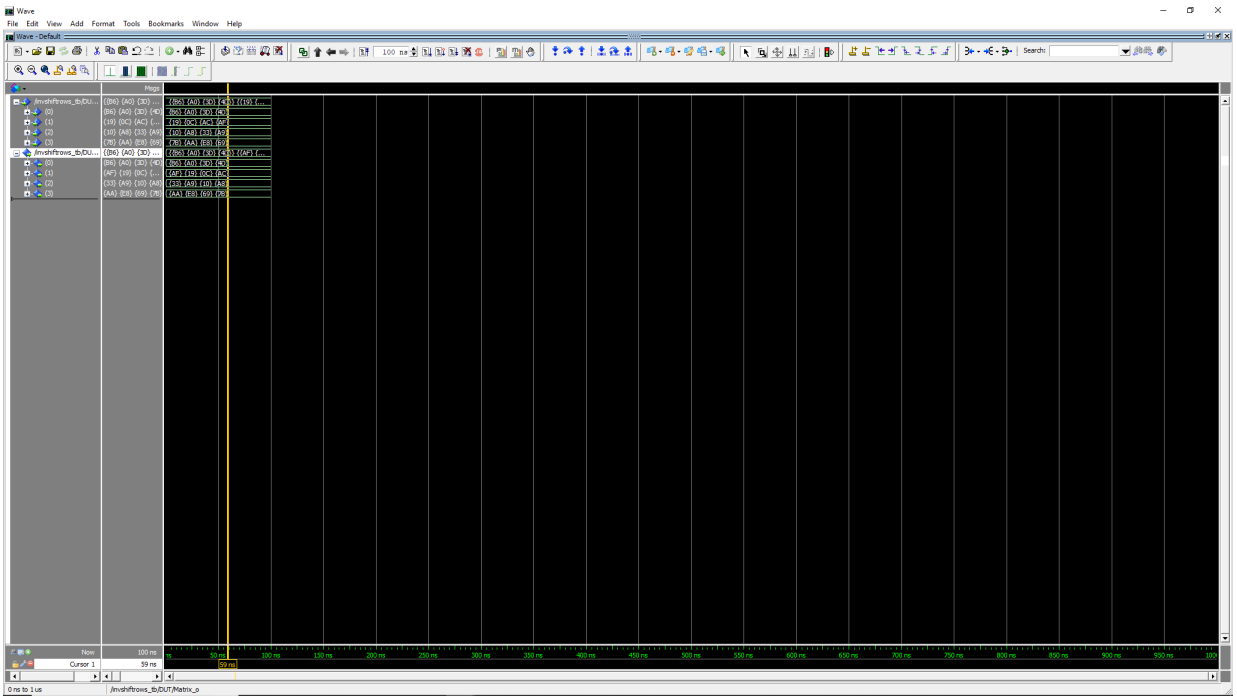

FIGURE 2.6 – Testbench de InvShiftRows

2.4 La transformation InvSubBytes

2.4.1 Présentation

Cette transformation non linéaire consiste à prendre chaque octet, le passer dans une SBOX afin d'obtenir un nouvel octet. Lorsqu'un octet rentre dans la SBOX, l'octet va servir de coordonnée dans un tableau nommé SBOX : les 4 bits de poids forts correspondent à la ligne et les 4 bits de poids faible à la colonne.
La SBOX est donc un tableau de 16 par 16 avec des octets dans chaque case. Elle est générée au préalable et simplement rentrée dans l'algorithme.
Voici un exemple de SBOX qui sera utilisé dans le projet :

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
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 2.7 – SBOX utilisée dans le projet

2.4.2 Implémenation

Pour implémenter cette transformation, j'ai réalisé deux composants : InvSBOX et InvSubBytes. Le composant InvSBOX prend en entrée un octet et en sortie un octet. C'est la que la valeur de la SBOX va être récupéré.
Le composant InvSubBytes prend en entrée un type_state et ressort un type_state. Dans ce composant il y a deux boucles qui vont premettre de parcourir la totatilité de la matrice (colonne et ligne). Pour chaque élèment, le composant va faire appel à la fonction SBOX afin d'effectuer la transformation.
Voici les schéma des composants à implémenter :

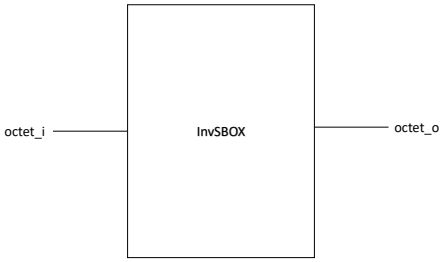

FIGURE 2.8 – Schéma du composant InvSBOX

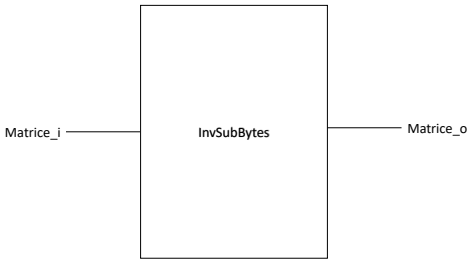

FIGURE 2.9 – Schéma du composant InvSubBytes

2.4.3 Résultats

Afin de verifier les résultats de la transformation, j'ai effectué 2 tests : un pour le composant InvSBOX et un autre pour InvSubBytes. Les deux composants fonctionnent parfaitement.

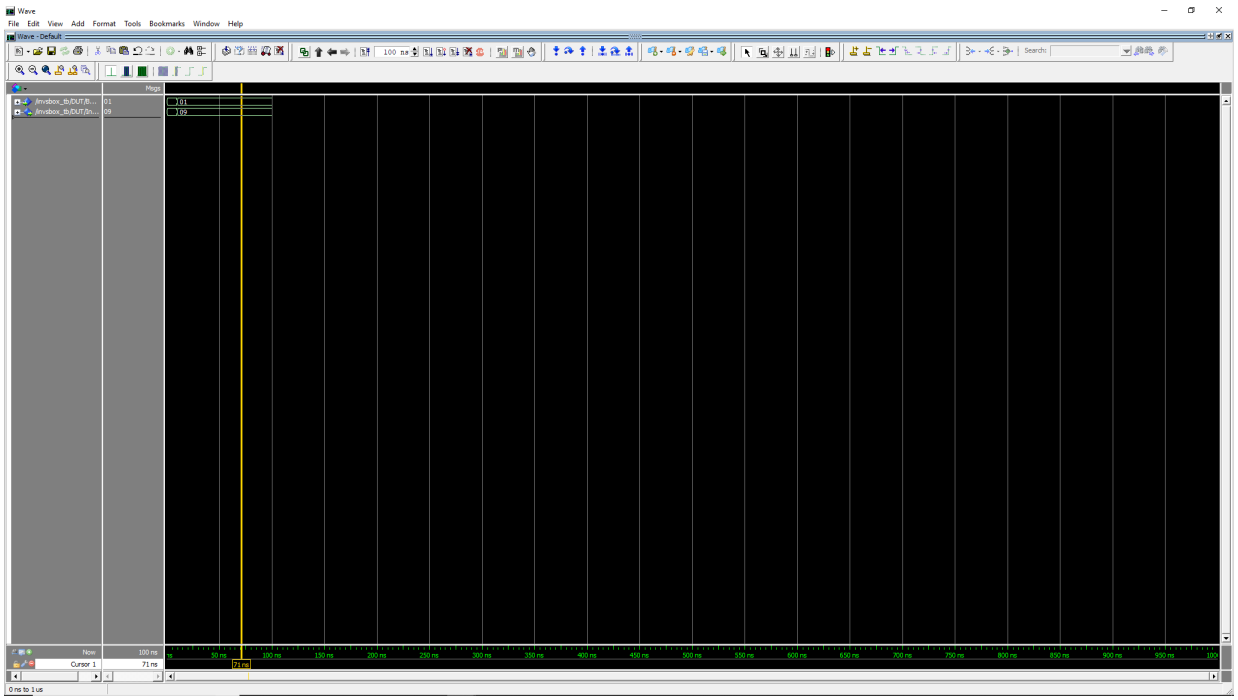

FIGURE 2.10 – Testbech de la SBOX

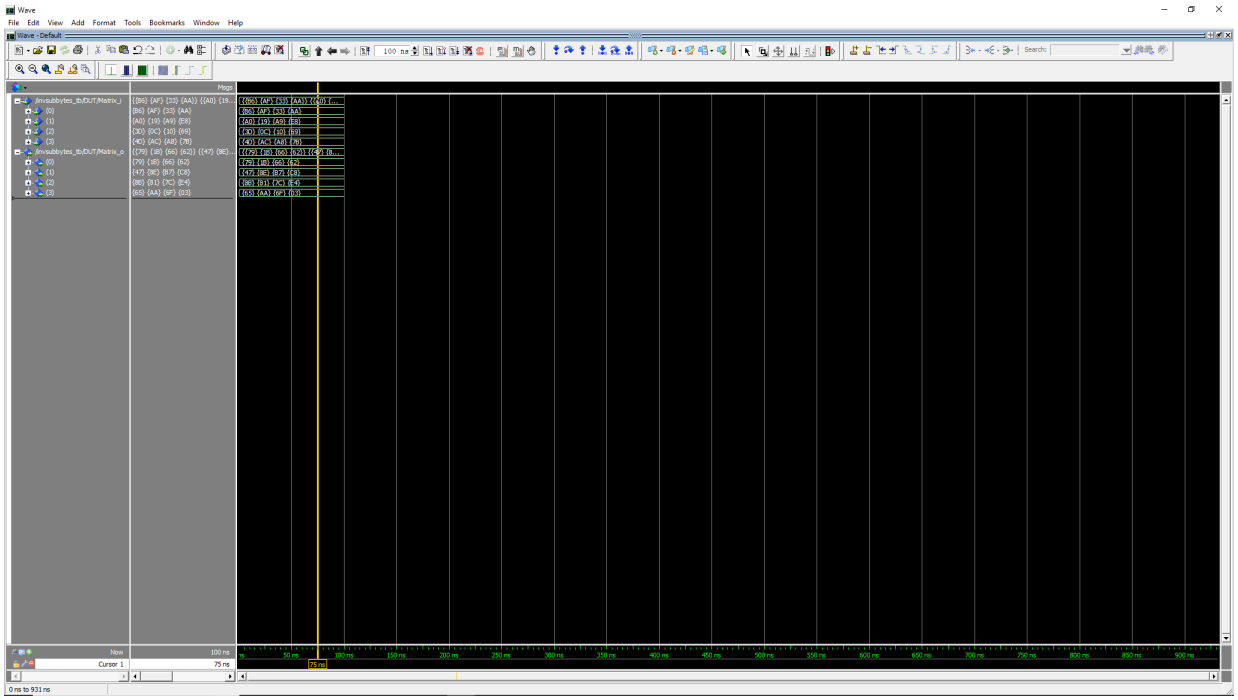

FIGURE 2.11 – Testbech de InvSubBytes

2.5 La transformation InvMixColumns

2.5.1 Présentation

Cette transformation applique à toutes les colonnes de la matrice un produit matriciel. Il multiplie une colonne par une matrice carrée prédéfinie de taille 4.
Voici la transformation ainsi que la matrice utilisée :

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

$$\begin{aligned} s'_{0,c} &= (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c}) \\ s'_{1,c} &= (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c}) \\ s'_{2,c} &= (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c}) \\ s'_{3,c} &= (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c}) \end{aligned}$$

FIGURE 2.12 – Produit matriciel de la fonction InvMixColumns

2.5.2 Implémenation

Afin de réaliser cette transformation, j'ai encore divisé cette transformation en deux composants : InvMixColumn et InvMixColumns.
Le composant InvMixColumn s'ocupe de réaliser le produit matriciel entre une colonne et la matrice présentée ci-dessus. Sa sortie est le résultat du produit matriciel.
Le composant InvMixColumns prend en entrée un type_state et ressort un type_state avec la transformation effectuée. Ce composant fait appel à InvMixColumn pour chaque colonne. Un problème rencontrée est le fait que un type_state est un tableau de ligne, il a donc fallu enregistrer la matrice dans 4 colonnes afin de les rentrer et après il était nécessaire de recréer un type_state de sortie avec les nouvelles colonnes.
Voici les schémas des composants à implémenter :

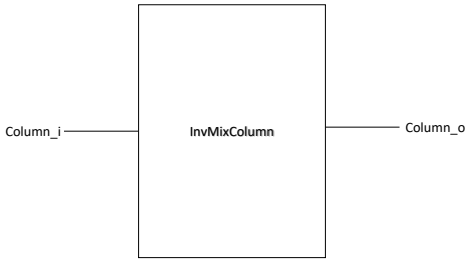

FIGURE 2.13 – Schéma du composant InvMixColumn

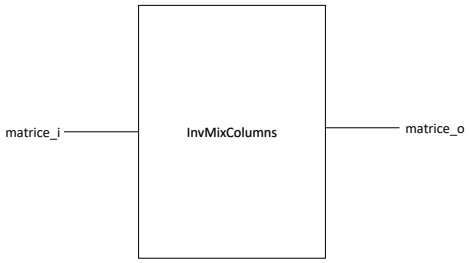

FIGURE 2.14 – Schéma du composant InvMixColumns

2.5.3 Résultats

Voici les résultats obtenus pour les deux composants :

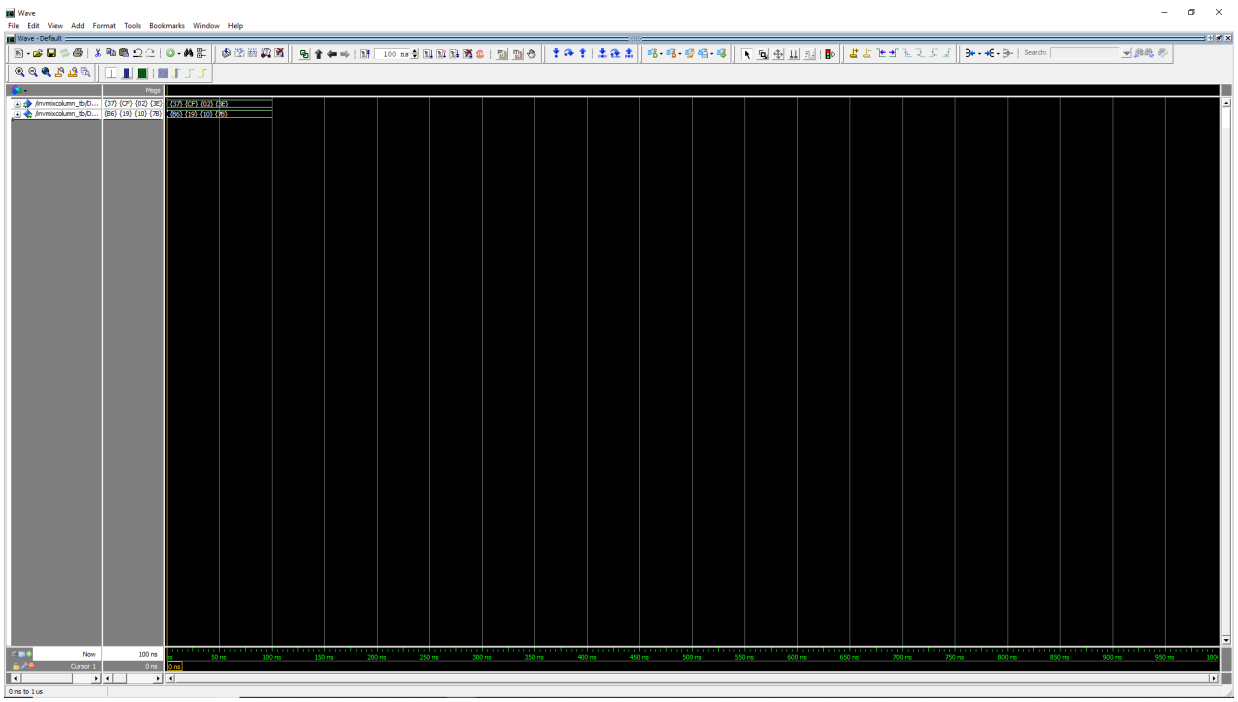

FIGURE 2.15 – Testbech de InvMixColumn

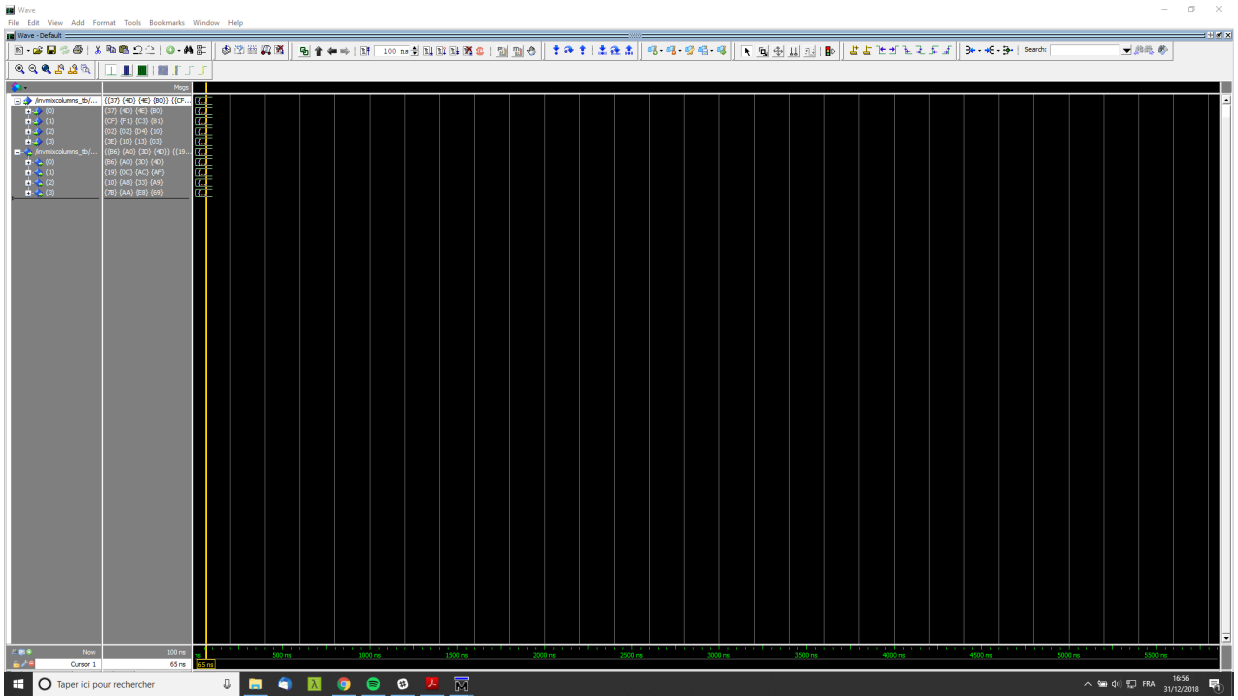

FIGURE 2.16 – Testbech de InvMixColumns

2.6 Le composant Counter

2.6.1 Présentation

Ce composant sert à implémenter l'aspect de tour dans l'algorithme. Son seul rôle est de gérer le numéro de tour. Lorsque que le composant détecte un front d'horloge montant, il doit décrémenter une valeur qui est stockée en lui et l'envoyer en sortie.

2.6.2 Implémenation

Au niveau des entrées, nous avons une pour l'horloge, une qui autorise le composant à fonctionner puis deux entrées qui servent à réinitialiser le compteur. Le composant a une seule sortie qui est codée sur 4 bits.
Au niveau du choix du fonctionnement, il y avait deux choix : un décompteur ou un compteur. Étant donné le fonctionnement du composant ExpansionKey, il est plus intéressant d'implémenter un décompteur.
Voici le schéma du composant à implémenter :

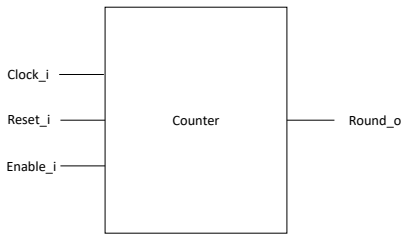

FIGURE 2.17 – Schéma du composant Counter

2.6.3 Résultats

Le test du composant counter consiste à voir si ça sortie se décrémente à chaque coup d'horloge. Voici le résultat, le fonctionnement est correct :

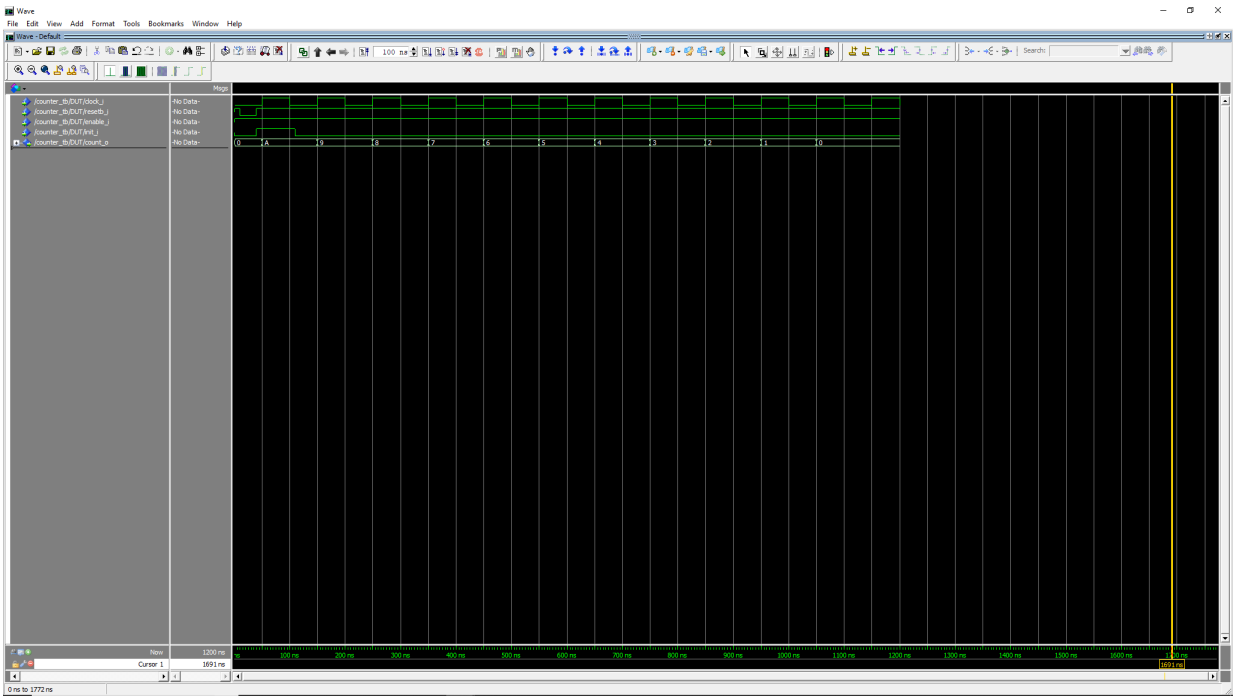

FIGURE 2.18 – Testbech du compteur

2.7 Le composant Registre

2.7.1 Présentation

Le registre sert à autoriser la sortie du composant global. Étant donné qu'il y a 11 tours, il faut authoriser la sortie lorsque le traitement est fini dans sa totatilé.

2.7.2 Implémenation

Le composant a deux entrées : un mot de 128 bits et un bit enable qui sert d'intérupteur. l'unique sortie est un mot de 128 bits.
L'autorisation de sortie se fait lorsque le bit enable est à l'état et que l'horloge est sur un front montant.
Voici le schéma du composant à implémenter :

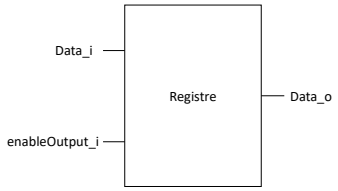

FIGURE 2.19 – Schéma du composant Registre

2.7.3 Résultats

Voici la simulation du registre, on voit qu'il répond aux attentes :

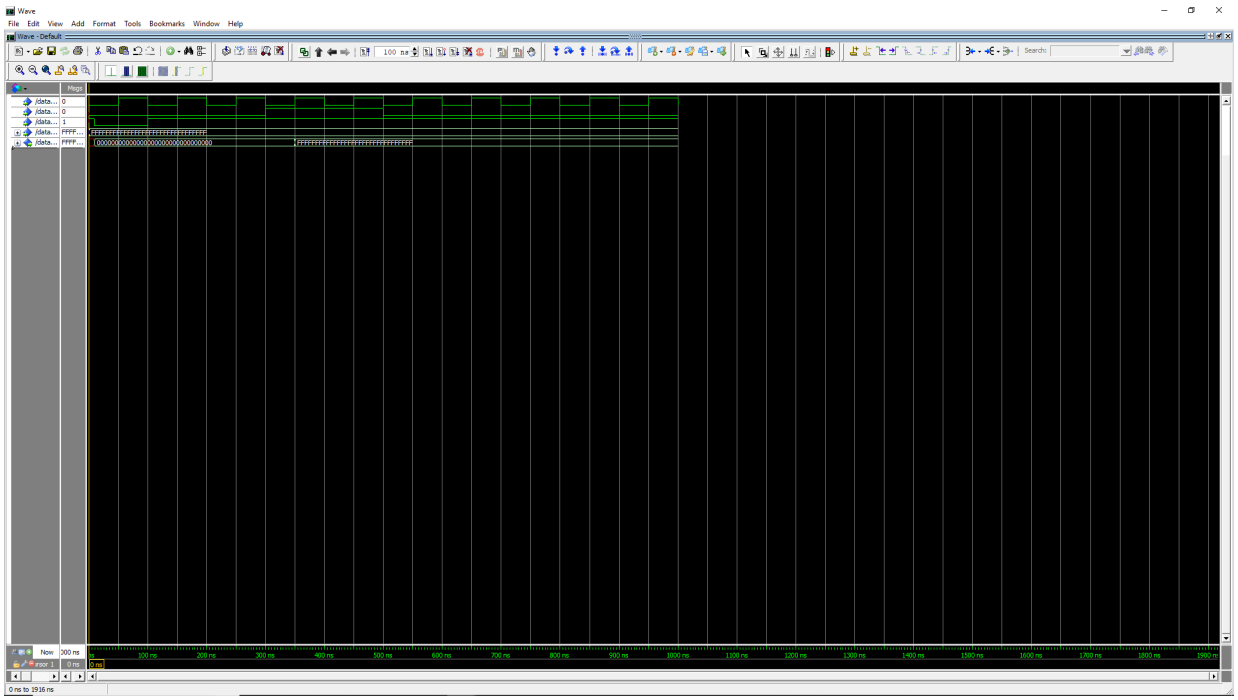

FIGURE 2.20 – Testbech du registre

2.8 Le composant Multiplexeur

2.8.1 Présentation

Le multiplexeur sert à choisir quel mot de 128 bits va être en entrée du composant s'occupant des transformations (InvRoundAES). Il y a deux choix possibles : soit on utilise le texte sortant de InVRoundAES, soit on utilise un nouveau mot.

2.8.2 Implémenation

Le multiplexeur a deux entrées de mots de 128 bits ainsi qu'une entrée dite de commande qui sert à selectionner le mot qui va être en sortie du multiplexeur. Étant donné qu'il n'y a que deux mots, un seul bit est nécessaire afin de faire le choix entre les deux entrées.
Voici le schéma du composant à implémenter :

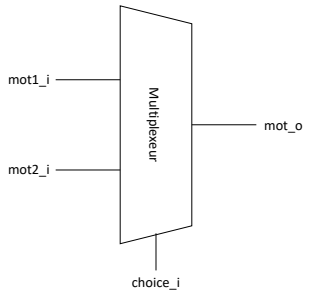

FIGURE 2.21 – Schéma du composant Multiplexeur

2.9 Le composant FSM (machine d'états)

2.9.1 Présentation

Le composant clé du système est la FSM (Finite-State Mahine). C'est elle qui va en fonction des entrées donner des ordres aux autres composants.
Il existe deux grands types de machine d'états : machine de Moore et machine de Mealy. Pour notre machine d'état nous implémenterons une machine de Moore car les sorties dépendent uniquement de l'état dans lequel est la FSM.

2.9.2 Implémenation

Le composant a 4 entrées : une pour la réinitialisation, une pour l'horloge, une pour le start et une pour le tour de l'algorithme. Au niveau des sorties, nous en avons 7 : une pour indiquer que l'algorithme est fini, une pour autoriser le composant Counter à décompter, une pour autoriser le MixColumns, une pour autoriser la sortie final, une pour autoriser le InvShiftRows et le InvSubBytes, une pour autoriser un nouveau texte dans l'algorithme et une sortie pour réinitialiser le composant Counter.
Au niveau de l'implémentation nous avons besoin de 3 process : un pour gérer l'état présent, un pour gérer l'état futur et un pour gérer les sorties en fonction de l'état présent.
Un dernier état a été rajouté suite au test de InvAES. En effet sans cet état, j'avais plusieurs sorties différents à la suite dont la bonne.
Voici le graphe d'état qui va nous servir à implémenter la FSM :

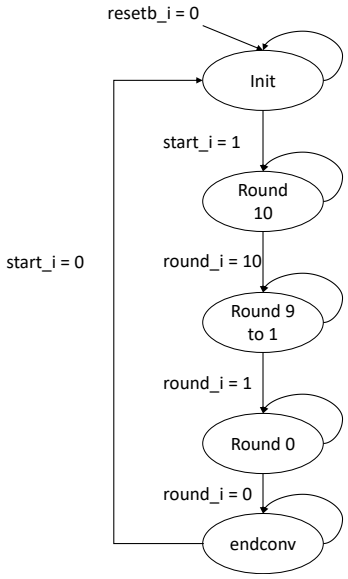

FIGURE 2.22 – Graphe d'état de la machine d'état

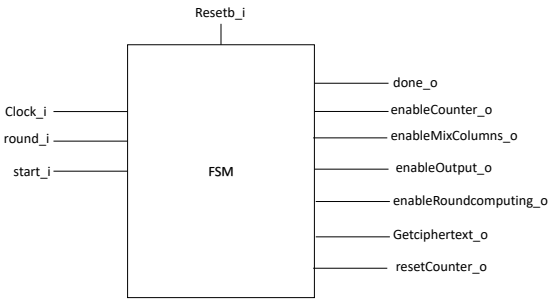

FIGURE 2.23 – Schéma du composant FSM

2.9.3 Résultats

Afin de simuler la machine d'état, j'ai simulé la décrémentation de la valeur de round_i. On obtient bien les résultats voulus avec les bonnes sorties en fonction de l'état présent.

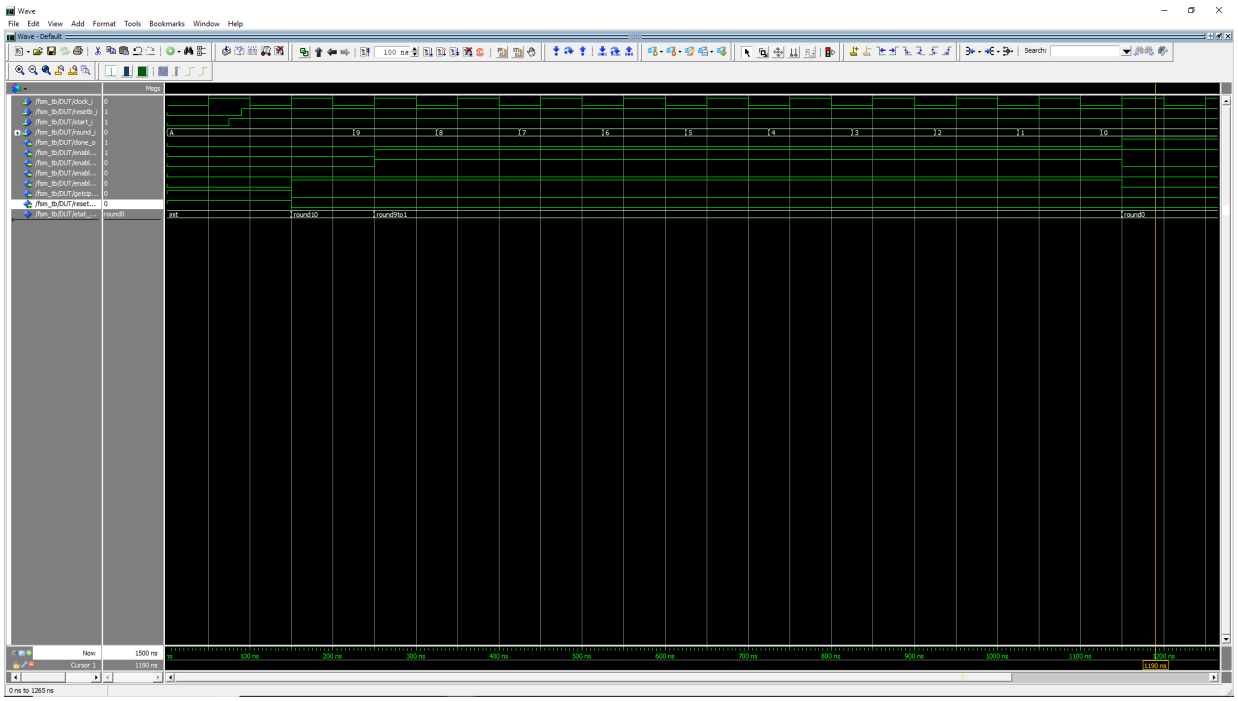

FIGURE 2.24 – Testbech de la FSM

2.10 Le composant InvRoundAES

2.10.1 Présentation

Ce composant regroupe tous les composants effecutant une transformation. Étant donné que un mot de 128bits peut avoir 3 transformations différentes : celle du tour 10, celle du tour 1 à 9 et celle du tour 0. Dans ce composant nous allons seulement relier les transformations.

2.10.2 Implémenation

Au niveau de l'implémentation, il faut créer des signaux afin de connecter tous les composants : l'objectif étant le liée la sortie d'un composant à l'entrée d'un autre composant.
Il faut aussi prendre en compte que le composant accepte en entrée et en sortie des bit128 il faut donc effectuer une conversion de bit128 vers type_state afin de traiter le mot.
Voici le schéma à implémenter :

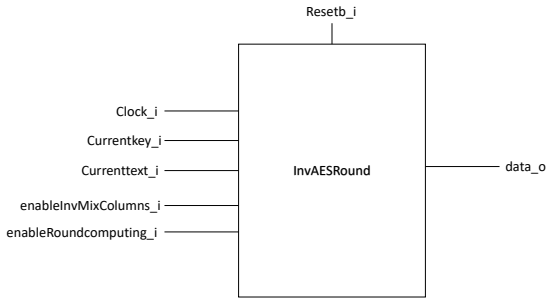

FIGURE 2.25 – Schéma du composant InvRoundAES

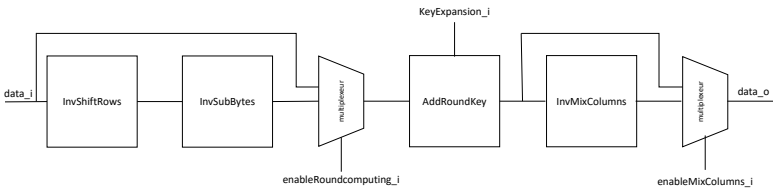

FIGURE 2.26 – Schéma intérieur du composant InvRoundAES

2.10.3 Résultats

Pour tester ce composant, j'ai simulé chaque type de fonctionnement possible : tous les fonctionnements sont opérationnel.
Voici un exemple pour le tour 8 :

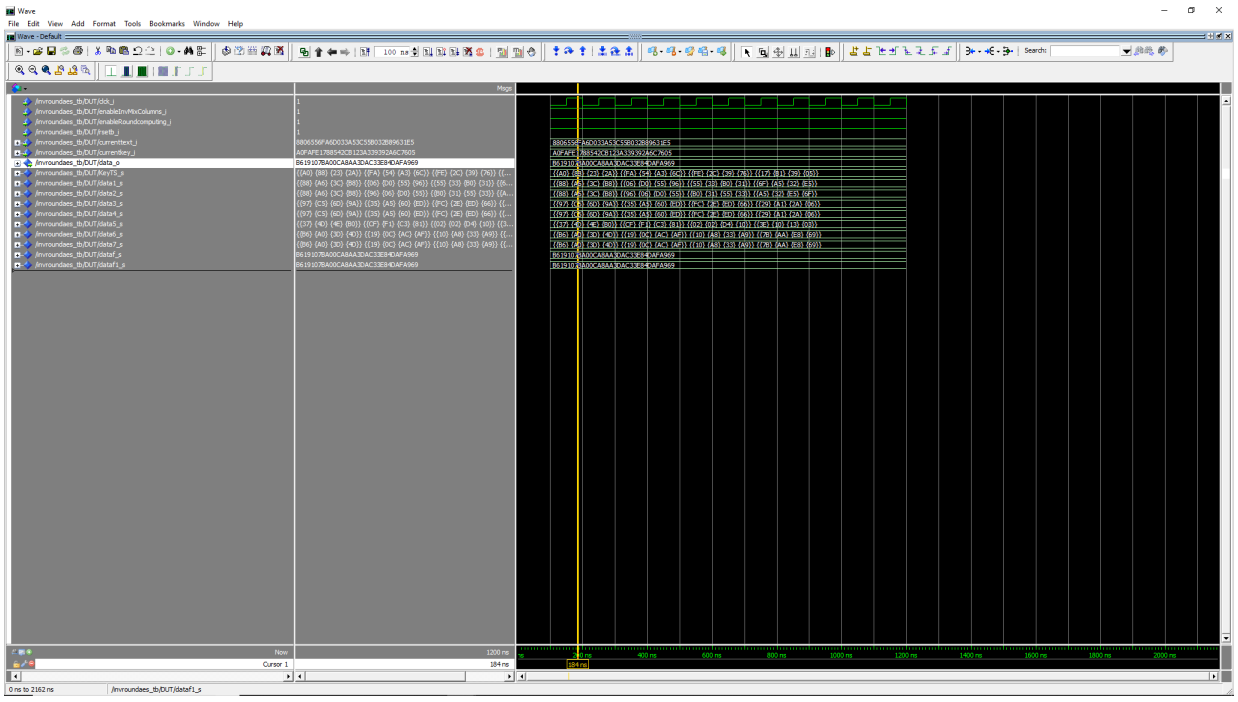

FIGURE 2.27 – Testbech de InvRoundAES

2.11 Le composant InvAES

2.11.1 Présentation

Comme le InvRoundAES, le InvAES est un composant qui regroupe plusieurs composants. InvAES est le composant final, c'est lui qui regroupe tout l'algorithme de déchiffrage de l'AES. Dans ce composant nous rentrons un message chiffré et à ça sortie nous récuperons le message original. Voici le schéma du composant InvAES fourni par les professeurs :

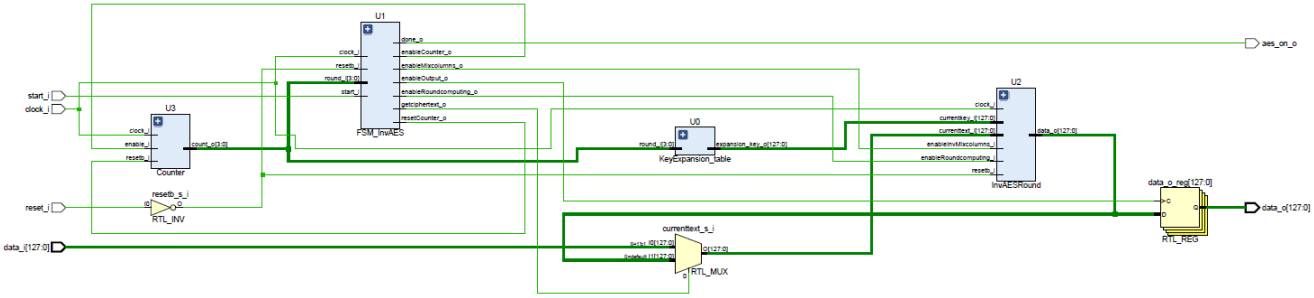

FIGURE 2.28 – Schéma interne du composant InvAES

2.11.2 Implémenation

Au niveau des entrées, on retrouve le mot à déchiffrer, l'horloge, un reset et un start. Au niveau des sorties nous avons le signal "done" qui indique que le déchiffrement est terminé (à la place de aes_on_o) puis la sortie principale qui est le mot de 128bits déchiffré.
Au niveau du code il a été nécessaire de créer comme dans InvRoundAES, des singaux intermédiaire entre les composants afin de les relier. Voici le schéma à implémenter :

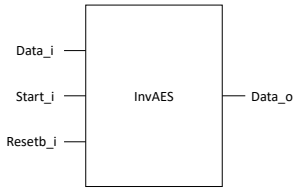

FIGURE 2.29 – Schéma du composant InvAES

2.11.3 Résultats

Dans ce test, il suffit simplement d'entrée le mot chiffré et de voir si la sortie correspond au mot non chiffré. Afin d'être plus représentatif, j'ai converti la sortie en ASCII pour voir le message : on a bien "Resto en ville ?". Le composant InvAES est opérationnel.
Voici le résultat obtenu :

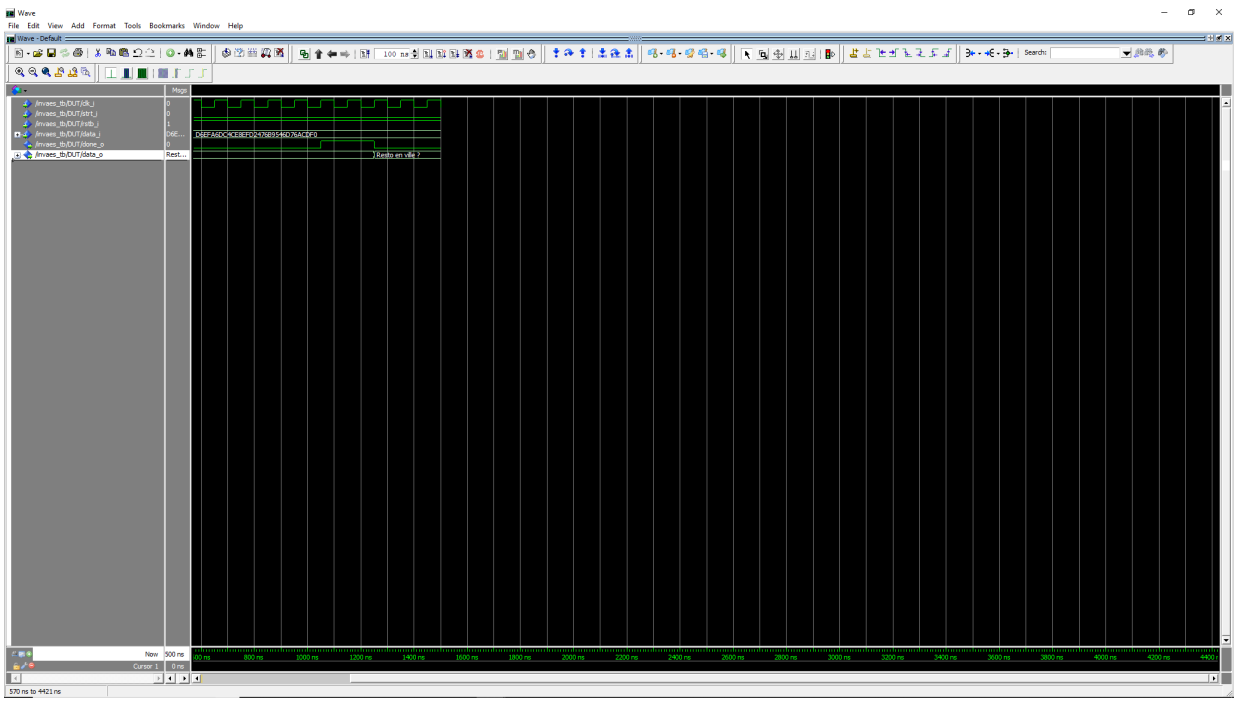

FIGURE 2.30 – Testbech de InvAES

Chapitre 3

Conclusion

Pour conclure ce projet m'aura permis de me familiariser avec le langage VHDL.
Au niveau du déroulement du projet, j'ai du dans un premier temps, comprendre les différentes transformations ainsi que trouver la meilleur implémentation possible.
Au niveau de la difficulté, les différents composants principaux n'étaient pas dur à implémenter cependant lors de l'implementation de InvRoundAES et InvAES, le fonctionnement de certains composants se sont révélés incompables avec d'autres composants. Par exemple, j'ai du ré-implémenter le composant Counter afin d'obtenir le résultat correct.
J'ai aimé ce projet car il m'a permis de maîtriser les bases du VHDL ainsi que d'implémenter un algorithme utile. De plus le fait que le résultat soit correct m'apporte de la satisfaction.