# Rapport VHDL : Déchiffrement AES

Marion WILMS EI17

7 janvier 2019

## Table des matières

# 1 Algorithme InvAES

## 1.1 Description générale

L'AES (Advenced Encryption Standard) est un algorithme de chiffrement symétrique par bloc. C'est un algorithme de chiffrement à clef secrète et il fait partie des SPN (Substitution and Permutation Network), c'est-à-dire que le texte que l'on souhaite chiffrer va passer dans un réseau de transformations de substitutions ou de permutations. C'est un algorithme robuste face à l'attaque par force brute, c'est-à-dire avec une recherche exhaustive envisageable, contrairement au DES (Data Encryption Standard) devenu obsolète.

L'algorithme de cryptage AES prend en entrée un bloc de données à chiffrer de 128 bits et une clef de chiffrement de 128 bits, 192 bits ou 256 bits. La sortie de l'algorithme est le bloc de données chiffrées de taille 128 bits également.

Les blocs clairs ou chiffrés (plaintext) sont représentés par des matrices 4 x 4 dont les éléments sont les octets du message. Il nous est demandé de développer en VHDL l'algorithme de déchiffrement AES (InvAES) avec une clef de 128 bits.

Le déroulement de InvAES est articulé autour de 4 étapes élémentaires : InvSubBytes, InvShiftRows, InvMixColumn et AddRoundKey suivant l'algorigramme ci-dessous.

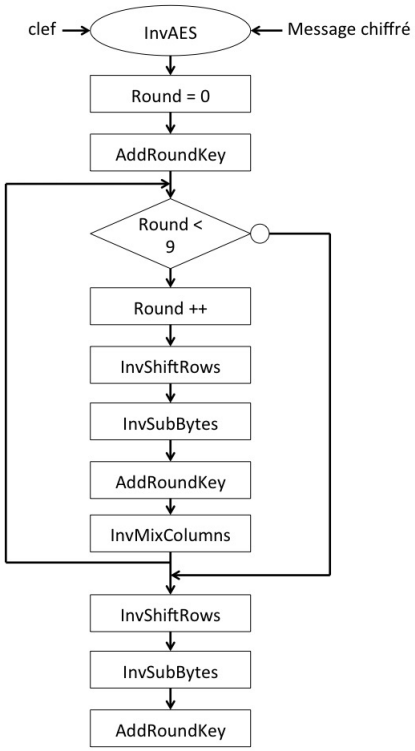


FIGURE 1 – Algorigramme InvAES

## 1.2 Description des transformations

### 1.2.1 Inv SubBytes

Il s'agit de la seule transformation non linéaire de l'algorithme. C'est une transformation appliquée à chaque octet du message chiffré en utilisant l'inverse de la S-Box en susbtituant l'octet {0xij} par l'octet situé à la i$^{ème}$ ligne et à la j$^{ème}$ colonne de l'inverse de la S-Box.

### 1.2.2 AddRoundKey

La fonction AddRoundKey effectue un XOR bit à bit pour chaque octet de la matrice de l'état avec la clef correspondant au Round. Comme il s'agit du déchiffrement AES, les clefs sont prises dans le sens inverse (commençant par la clef 10 pour finir par la clef 0) mais la fonction reste identique à celle du chiffrement AES car l'inverse d'un XOR est un XOR.

### 1.2.3 Inv ShiftRows

La transformation Inv ShiftRows effectue une permutation cyclique des octets des lignes du bloc chiffré. La première ligne reste inchangée, dans la seconde le 4$^{ème}$ octet prend la première place, etc ... comme illustré dans le tableau ci-dessous.

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 4 | 1 | 2 | 3 |
| 3 | 4 | 1 | 2 |
| 2 | 3 | 4 | 1 |

TABLE 1 – Fonctionnement de Inv ShiftRows

### 1.2.4 Inv MixColumns

La fonction InvMixColumn effectue un produit matriciel sur chaque colonne de la matrice d'état. Chaque colonne est multipliée par :

$$M = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix}$$

La multiplication par 2 dans le corps de galois GF($2^8$) correspond à un décalage à gauche et l'addition est un XOR. Pour multiplier par 11 dans le corps de Galois, cela revient à multiplier par 8 + 2 + 1. Le tout doit être calculé modulo b' 100011011.

# 2 Implémentations et Tests des 4 transformations

## 2.1 Notations

Nous devons respecter les contraintes suivantes.
— Les entrées sont suivies du suffixes "i".
— Les sorties sont suivies du suffixes "o".
— Les signaux sont suivies du suffixes "s".
— Les matrices d'état sont représentées comme ci-dessous avec un remplissage par colonne.

| | | | |
|---|---|---|---|
| 1 | 5 | 9 | 13 |
| 2 | 6 | 10 | 14 |
| 3 | 7 | 11 | 15 |
| 4 | 8 | 12 | 16 |

TABLE 2 – Représentation de la matrice d'état

## 2.2 Inv SubBytes

### 2.2.1 Implémentation

L'entité Inv_S_Box prend en entrée un bit8, c'est-à-dire un octet et retourne un bit8.

L'entité Inv_SubBytes prend en entrée un type state, qui est une matrice 4 x 4 de bit8. Pour chaque bit8, on applique la transformation Inv_S_Box.

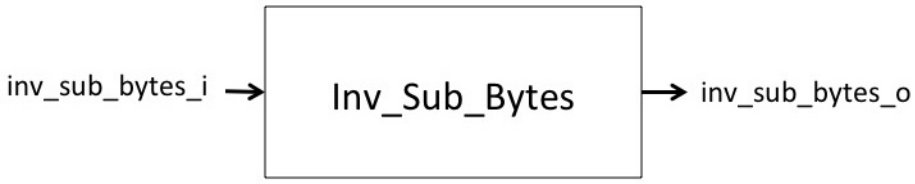


FIGURE 2 – Entité Inv SubBytes

### 2.2.2 Test

Pour tester l'inverse de la SBOX, j'ai incrémenté un compteur bit8 dans un process pour tester les 256 entrées possibles et vérifier si les sorties correspondaient. On peut observer sur la figure 3 une portion du résultat (l'entrée est en décimal et la sortie est en hexadécimal).

Pour tester Inv_SubBytes, j'utilise les résultats de l'algorithme AES donnés dans le PDF de présentation du sujet du projet VHDL pour le texte clair : « Resto en ville ». Je mets en entrée de Inv_SubBytes b6 af 33 aa a0 19 a9 e8 3d 0c 10 69 4d ac a8 7b (sortie de SubBytes du Round 1) et on vérifie qu'on a en sortie 79 1b 66 62 47 8e b7 c8 8b 81 7c e4 65

aa 6f 03 (sortie de AddRoundKey du Round 0) en respectant la forme des matrices d'états (remplissage en colonne). On peut voir le resultat sur la figure 4.

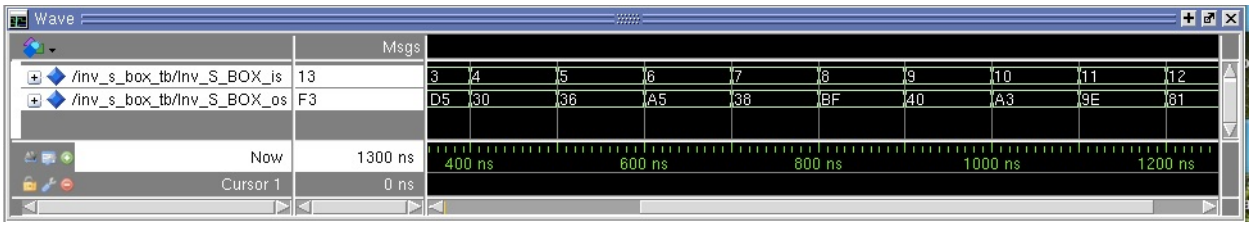


FIGURE 3 – Résultat de Inv SBOX

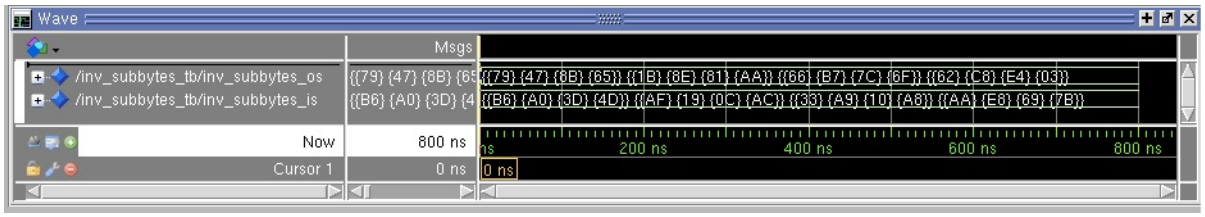


FIGURE 4 – Résultat de Inv SubBytes

## 2.3 AddRoundKey

### 2.3.1 Implémentation

L'entité Add_Round_Key prend deux type_state en entrée : la clef de 128 bits et le message chiffré, et retourne un type_state.

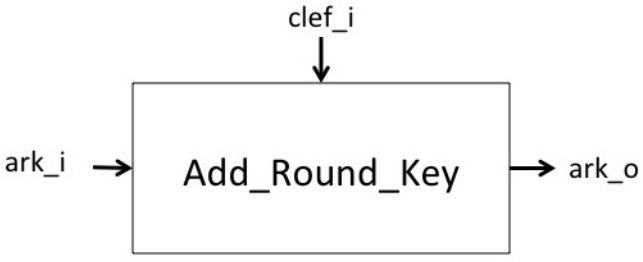


FIGURE 5 – Entité AddRoundKey

### 2.3.2 Test

Pour tester Add_Round_Key, j'utilise les mêmes résultats que pour le test de Inv_SubBytes. Je mets en entrée de Add_Round_Key 79 1b 66 62 47 8e b7 c8 8b 81 7c e4 65 aa 6f 03

(sortie de AddRoundKey du Round 0) et on vérifie qu'on a en sortie 52 65 73 74 6f 20 65 6e 20 76 69 6c 6c 65 20 3f (le Plaintext) en respectant la forme des matrices d'états (remplissage en colonne). On peut voir le resultat sur la figure ci-dessous.

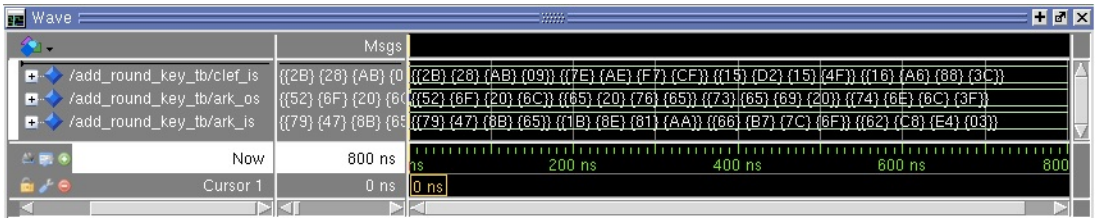


FIGURE 6 – Résultat de AddRoundKey

## 2.4 Inv ShiftRows

### 2.4.1 Implémentation

L'entité Inv_Shift_Rows prend un type_state en entrée : le message chiffré, et retourne un type_state.

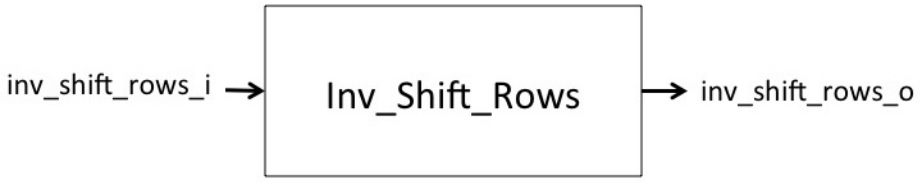


FIGURE 7 – Entité Inv ShiftRows

### 2.4.2 Test

Pour tester Inv_Shift_Rows, je mets en entrée b6 19 10 7b a0 0c a8 aa 3d ac 33 e8 4d af a9 69 (sortie de ShiftRows du Round 1) et on vérifie qu'on a en sortie b6 af 33 aa a0 19 a9 e8 3d 0c 10 69 4d ac a8 7b (sortie de SubBytes du round 1). On peut voir le resultat sur la figure ci-dessous.

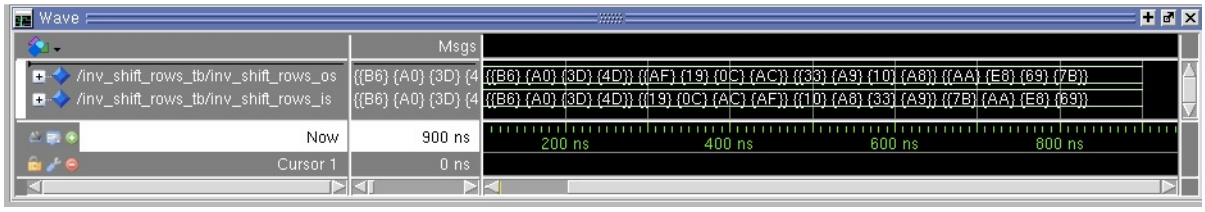


FIGURE 8 – Résultat de Inv ShiftRows

## 2.5 Inv MixColumns

### 2.5.1 Implémentation

Cette transformation est la plus complexe à implémenter, c'est pour cela que j'ai d'abord commencé par implémenter la transformation Matrix_Multiplier prenant en entrée une colonne (column_state) et en retournant la colonne (column_state) multipliée par la matrice M.

Pour effectuer le modulo pour rester dans le corps de Galois, il faut faire un xor entre le bit8 et b'100011011 si le bit de poids fort du bit8 est 1 et ne rien faire si celui-ci est nul. Pour cela, si on appelle $b_f$ le bit de poids fort du bit8, il suffit de faire un xor avec $000b_fb_f0b_fb_f$.

L'entité Inv_Mix_Columns prend en entrée un type_state et retourne un type_state en utilisant 4 composants Matrix_Multiplier pour chaque colonne et non pas un seul composant pour ne pas utiliser de machine d'état pour cette transformation. Dans un premier temps, comme un type_state est un tableau de ligne, il faut transformer le tableau de lignes en 4 colonnes, puis appliquer la transformation Matrix_Multiplier à chaque colonne et retransformer les colonnes en tableau de lignes.

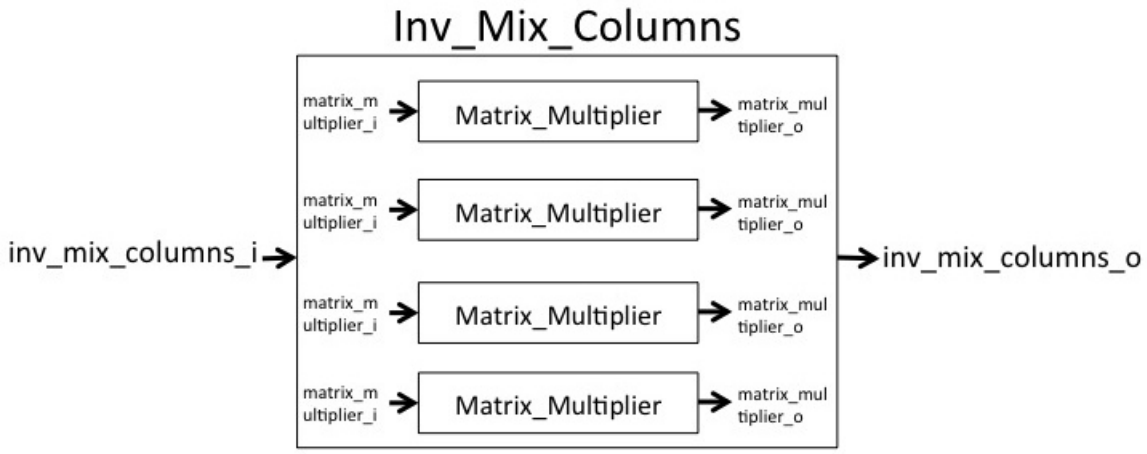


FIGURE 9 – Entité Inv MixColumns

### 2.5.2 Test

Pour tester Matrix_Multiplier, je mets en entrée la première colonne du round 1 de MixColumns (37 cf 02 3e) et je vérifie qu'on a en sortie la première colonne du round 1 de ShiftRow (b6 19 10 7b). On peut voir le resultat sur la figure 10. Pour tester Inv_Mix_Columns, je mets en entrée la sortie du round 1 de MixColumns (37 cf 02 3e 4d f1 02 10 4e c3 d4 13 b0 81 10 03) et je vérifie qu'on a en sortie la sortie du round 1 de ShiftRow (b6 19 10 7b a0 0c a8 aa 3d ac 33 e8 4d af a9 69). On peut voir le resultat sur la figure 11.

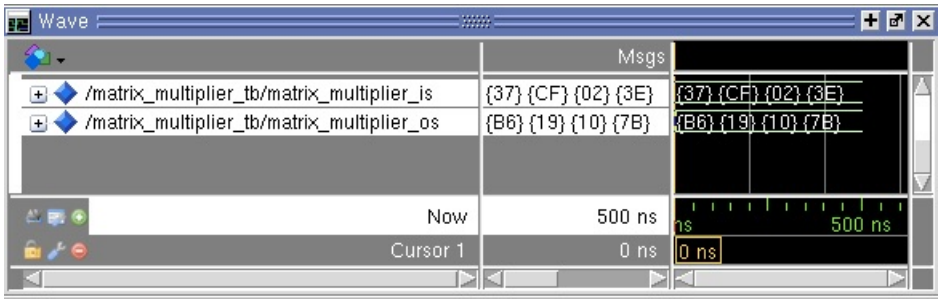


FIGURE 10 – Résultat de Matrix Multiplier

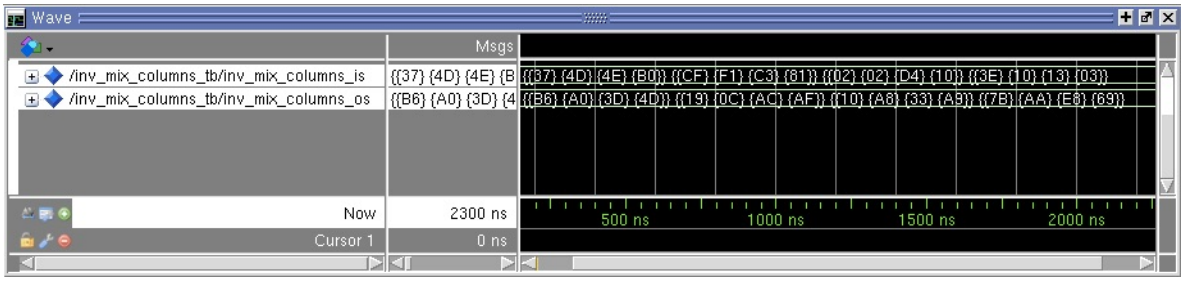


FIGURE 11 – Résultat de Inv MixColumns

# 3 Implémentation générale

## 3.1 Machine d'état

Nous utilisons pour cet algorithme une machine de Moore. La sortie ne dépend que de l'état présent et ainsi le temps de réaction dépend de l'horloge. Il s'agit d'une machine d'état synchrone, l'état change sur un front d'horloge. L'état future est calculé à partir des entrées et de l'état présent.

Ainsi la machine d'état prend en entrée :
— clock_i (l'horloge)
— resetb_i (un reset à l'état bas)
— round_i un bit4 (numero de round)

— et start_i (si début de déchiffrement).

La machine d'état retourne :

— done_o (phase de déchiffrement),
— enableCounter_o (autoriser le changement de round),
— enableMixColumns_o (autoriser MixColumns),
— enableOutput_o (autoriser l'envoie du texte déchiffré),
— enableRoundComputing_o (autoriser InvSubBytes et InvShiftRows),
— getciphertext_o (autoriser l'envoie du text à déchiffrer)
— et resetCounter_o (remettre à 10 le décompteur).

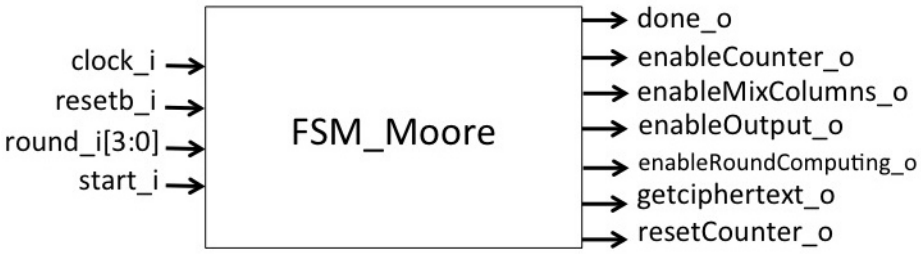


FIGURE 12 – Entité Machine de Moore

On a le diagramme d'état ci-dessous :

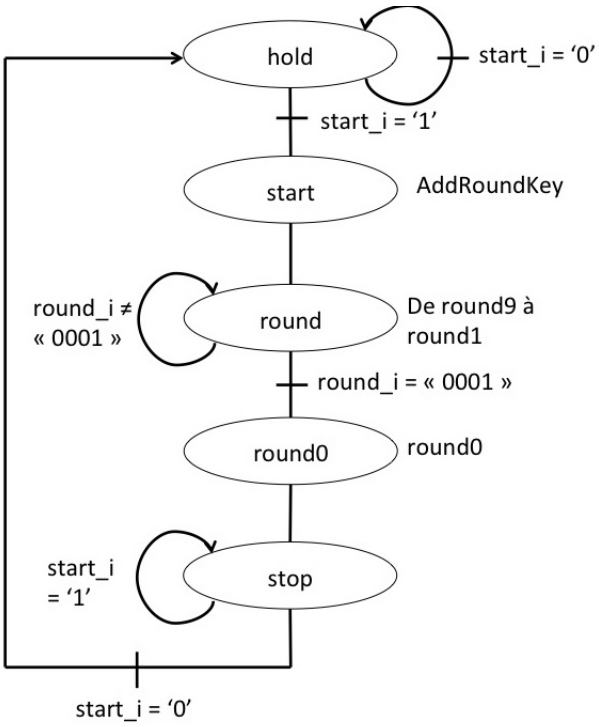


FIGURE 13 – Diagramme d'état

Et on obtient sur modelsim le resultat suivant :

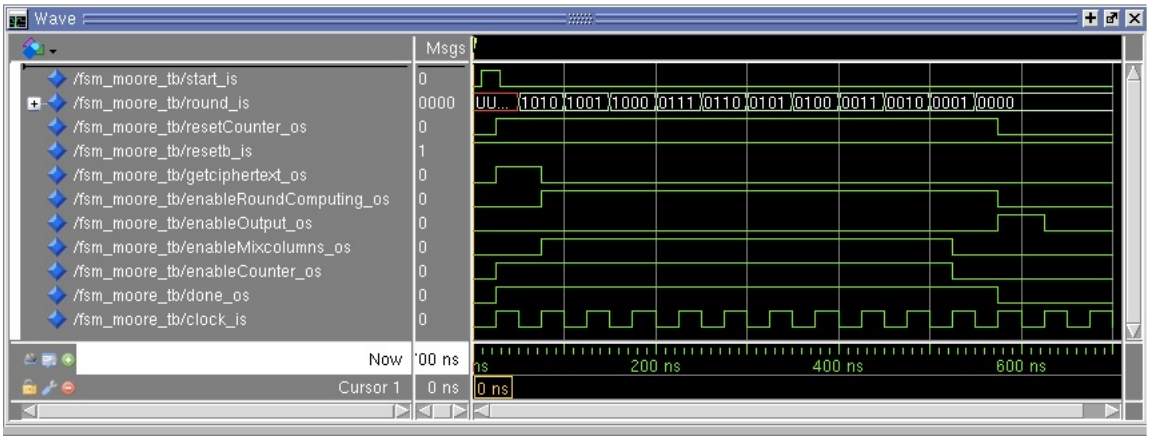


FIGURE 14 – Test de la machine d'état

## 3.2 Décompteur

Le décompteur permet de générer les round de manière synchrone lorsque "enable_i" vaut 1. La sortie counter_o est un bit4.

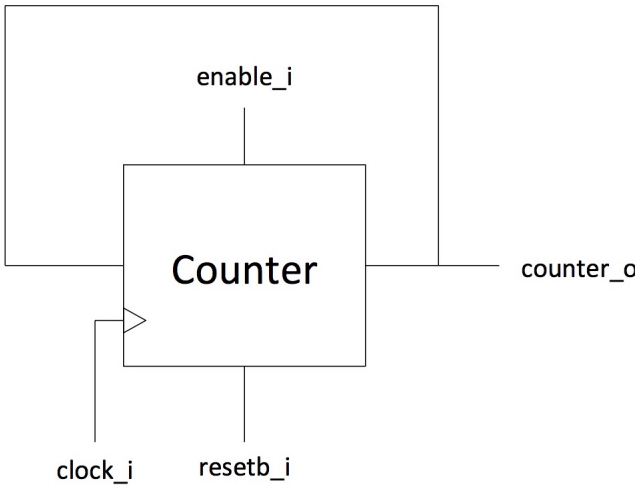


FIGURE 15 – Décompteur

On obtient le résultat suivant après simulation sur modelsim :

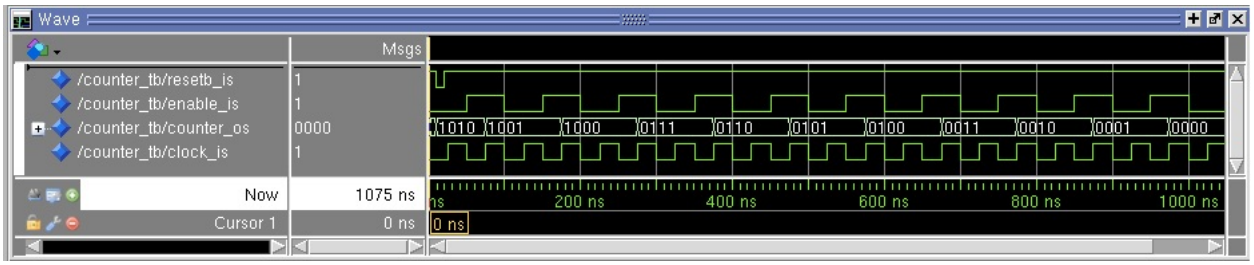


FIGURE 16 – Test du décompteur

## 3.3 Registre

Le registre permet si "enable_i" vaut 1 d'envoyer la sortie de déchiffrement sur un front d'horloge comme dans la figure 17 ci-dessous.

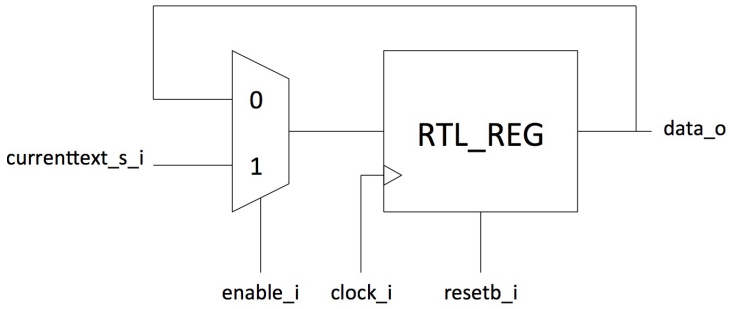


FIGURE 17 – Registre

On obtient comme résultat de simulation la figure 18, le message déchiffré est envoyé en sortie après le enable.

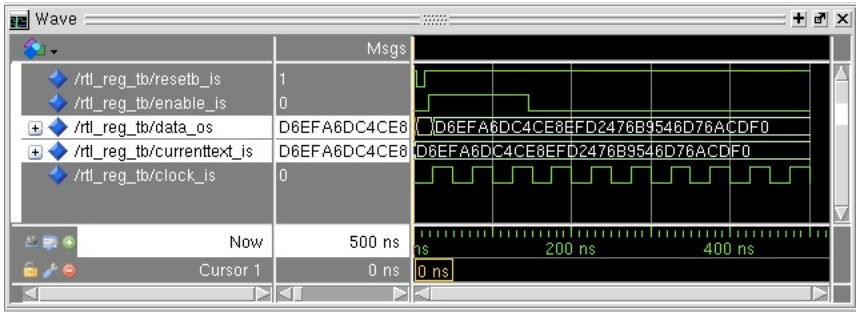


FIGURE 18 – Test du registre

## 3.4 InvAESRound

InvAESRound prend en entrée le texte du round en bit128, la clef du round en bit128, la clock, un reset, enableRoundComputing et enableMixColumns. J'utilise deux multiplexeur pour rediriger les signaux internes comme sur la figure 19 pour pouvoir effectuer uniquement AddRoundKey au round 10, et ne pas effectuer MixColumns au round 0 grâce aux "enable" de sortie de la machine d'état.

Les 4 transformations à l'interieur de InvAESRound prennent en entrée des type_state et retournent des type_state. Il faut donc convertir grâce à un for generate avant l'entrée de InvShiftRows le texte courant en type_state et la sortie avant le registre en bit128.

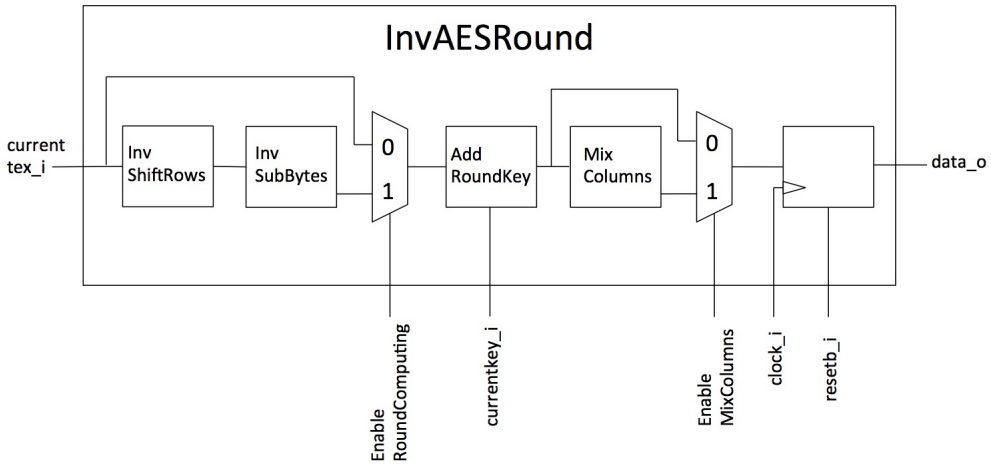


FIGURE 19 – InvAESRound

On obtient le resultat suivant après simulation, en entrant la clef du round 9 ac 77 66 f3 19 fa dc 21 28 d1 29 41 57 5c 00 6e et en mettant en entrée 06 fb 5f 74 85 06 ca 5b a6 54 99 8e 61 09 c1 56 (la sortie du round 10 de add round key lors du déchiffrement). Comme il s'agit du round 9, on autorise les 4 transformations.

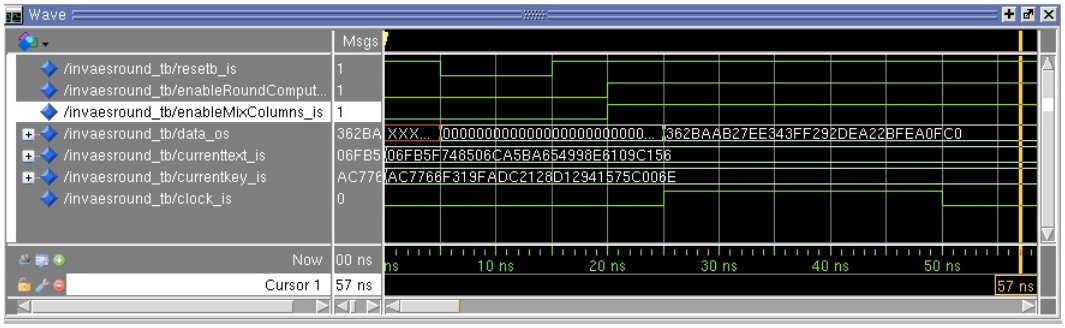


FIGURE 20 – Test de InvAESRound

## 3.5 InvAES

Pour implémenter l'architecture globale de InvAES, j'ai reproduit le schéma suivant, avec en entrée :

— start_i : déclanchant le début de déchiffrement,
— clock_i : l'horloge,
— data_i : le texte à déchiffrer,
— reset_i : permettant une remise à 0

et en sortie :

— aes_on_o : indiquant quand l'entité est en phase de déchiffrement,
— data_o : le texte déchiffré.

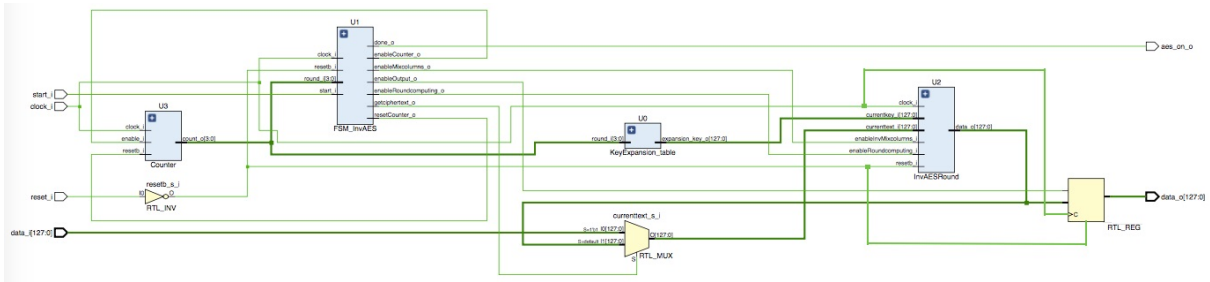


FIGURE 21 – Structure de InvAES

Les tests des entités précédentes ont été fait à partir du message chiffré "Resto en ville ?" que Bob envoie à Alice et on peut déchiffrer la réponse d'Alice "Sushi ou crepe" grâce à InvAES comme sur le résultat de simulation ci-dessous.

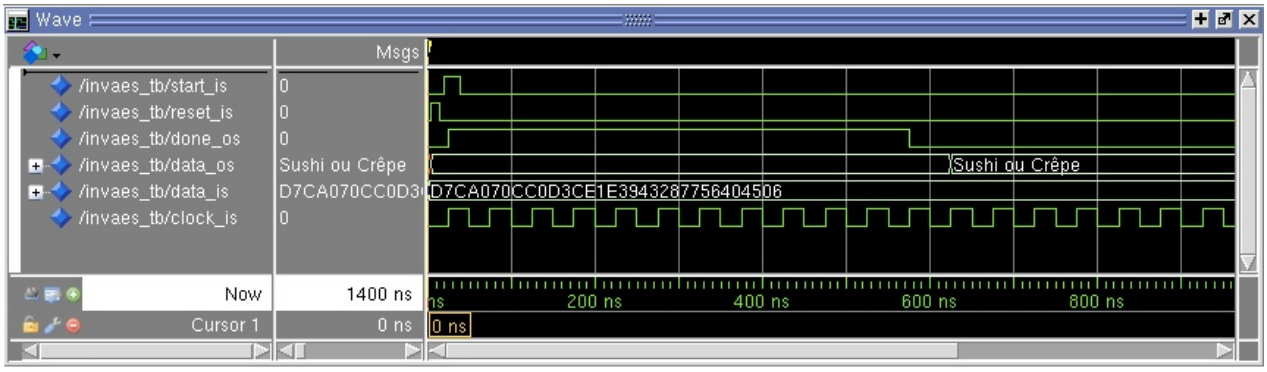


FIGURE 22 – Test de InvAES

# 4 Conclusion

Ainsi, mon implémentation de l'algorithme de déchiffrement AES fonctionne. J'ai modifié le registre RTL_REG par rapport au schéma donné dans l'énoncé en rajoutant le signal

d'horloge et l'entrée enableOutput. Lors de l'implémentation générale, des légères erreurs dans les différents blocs non visibles lors des tests des blocs seuls ont été un point bloquant. Pour résoudre ce problème, j'ai visualisé sur modelsim les signaux des différents DUT, notamment les "enable" et le numéro de round. J'ai rencontré ainsi deux problèmes :

— les resets à l'état haut et pas à l'état bas

— et un oubli de signal dans la liste de sensibilité.

De plus, j'ai rencontré des difficultés lors de l'implémentation de MixColumns pour la compréhension de la fonction premièrement et comment retranscrire cette fonction dans un second temps.