Conception de Système Numérique - Déchiffrement AES

Auteur:

DIOP Mohamed

Responsables:

REYMOND Guillaume, ZGHEIB Anthony

11 Février 2022

Table des matières

1 Introduction **4**

2 Organisation générale du projet **4**
2.1 Vue générale du dossier AES 4
2.2 Compilation du projet 5

3 Implémentation du InvShiftRows **5**
3.1 Principe 5
3.2 Réalisation du composant 5
3.3 Test Bench 5

4 Implémentation du InvSubBytes **6**
4.1 Principe 6
4.2 Réalisation du composant 6
4.3 Test Bench 7

5 Implémentation du AddRoundKey **7**
5.1 Principe 7
5.2 Réalisation du composant 8
5.3 Test Bench 8

6 Implémentation du InvMixColumns **9**
6.1 Principe 9
6.2 Réalisation du composant 9
6.3 Test Bench 9

7 Réalisation de l'InvAESRound **10**
7.1 Réalisation du composant 10
7.2 Test Bench 11

8 Réalisation de la machine d'états de Moore **13**
8.1 Principe 13
8.2 Réalisation 14

9 Réalisation de l'InvAES **15**
9.1 Réalisation du composant 15
9.1.1 Intégration du Counter 15
9.1.2 Intégration du KeyExpansion_table 15
9.1.3 Registre de sortie 15
9.1.4 Multiplexeur 15

9.2 Suite de l'entité InvAES 16
9.3 Test Bench 16

10 Conclusion **17**

1 Introduction

L'objectif de ce projet est de mettre en oeuvre les méthodes d'implémentation de composants avec le langage VHDL, en particulier pour la réalisation d'une puce de déchiffrement AES 128 bits. Ce projet se décompose principalement en 3 étapes : réalisation des composants de la puce, réalisation de l'entité gloable de déchiffrement AES (Top Level) et enfin, il s'agit de faire une réalisation de "testbench" pour ces deux premiers éléments.

2 Organisation générale du projet

2.1 Vue générale du dossier AES

Le dossier AES comporte un fichier script de compilation appelé *compile.sh*. Il permet de compiler l'intégralité du projet AES (fichiers du test-bench, fichiers sources, librairies, et packages).

Concernant l'organisation générale du dossier *AES*, il subdivise en deux dossiers : **LIB et SRC** comme attendu dans les exigences du cahier des charges. LIB contient les librairies du projet, et SRC contient les fichiers sources. Ensuite SRC est subdivisé en trois dossiers : BENCH pour les fichiers de test-bench, PACKAGE pour le fichier définissant les bit8 etc.. Et enfin RTL contenant les fichiers sources, permettant d'instancier les différents composants du projet.

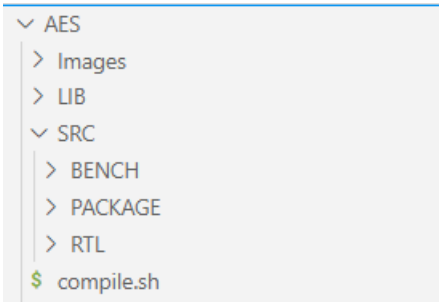

Figure 1 – Vue globale du dossier AES

2.2 Compilation du projet

Pour compiler le projet, , il suffit de lancer la commande *bash compile.sh* en se plaçant dans AES.

3 Implémentation du InvShiftRows

3.1 Principe

— **Objectif du composant :** La fonction *InvShiftRows* effectue une permutation cyclique (i.e. rotation) des octets des lignes de la matrice d'entrée. Le décalage des octets correspond à l'indice de la ligne considérée ($0 <= r < 4$). Ainsi, la ligne 0 reste identique, la deuxième voit ses coefficients décalés à droite d'un indice, et c'est le même principe pour les deux dernières lignes, avec respectivement un décalage de 2 et 3 indices à droite.

3.2 Réalisation du composant

Pour réaliser ce bloc, il n'est pas nécessaire d'implémenter une logique particulière, il suffit juste de manipuler les indices de la matrice d'entrée avec celle de la sortie.

3.3 Test Bench

Pour vérifier la cohérence du résultat, il suffit d'utiliser les résultats attendus dans le document relatant les résultats attendus après exécution des différents bloc (voir Fig. ci-dessous) (NB : cette *Round* servira de référence pour les tests dans la suite).

Par exemple après le Round 10, la Round 9 débute avec l'InvShiftRows qui prend en entrée la sortie du AddRoundKey (Round 10) : *0x6b 14 25 4f 88 2d 06 f6 a6 87 77 8d db 19 a2 63* et la sortie du InvShiftRows attendue est : *0x6b 19 77 f6 88 14 a2 8d a6 2d 25 63 db 87 06 4f*

Pour tester cela, il faut implémenter un test-bench, en instanciant et réutilisant le composant InvShiftRows à l'aide d'un fichier de configuration.

Pour voir le code complet de ce test-bench et sa configuration, voir le fichier correspondant en annexe dans le dossier du projet. Il est important de noter que ce code est la stratégie générale adoptée pour tout les test-bench, les variations sont seulement au niveau des composants utilisés et des entrées/sorties utilisés mais le principe reste le même. Il est jugé pertinent de ne pas réafficher les codes des test-bench pour les différents blocs suivant jusqu'à l'InvAES, par souci de non redondance.

Pour simuler ce test-bench, nous utilisons le logiciel *ModelSim*, ainsi on obtient bien le résultat attendu ci-dessous :

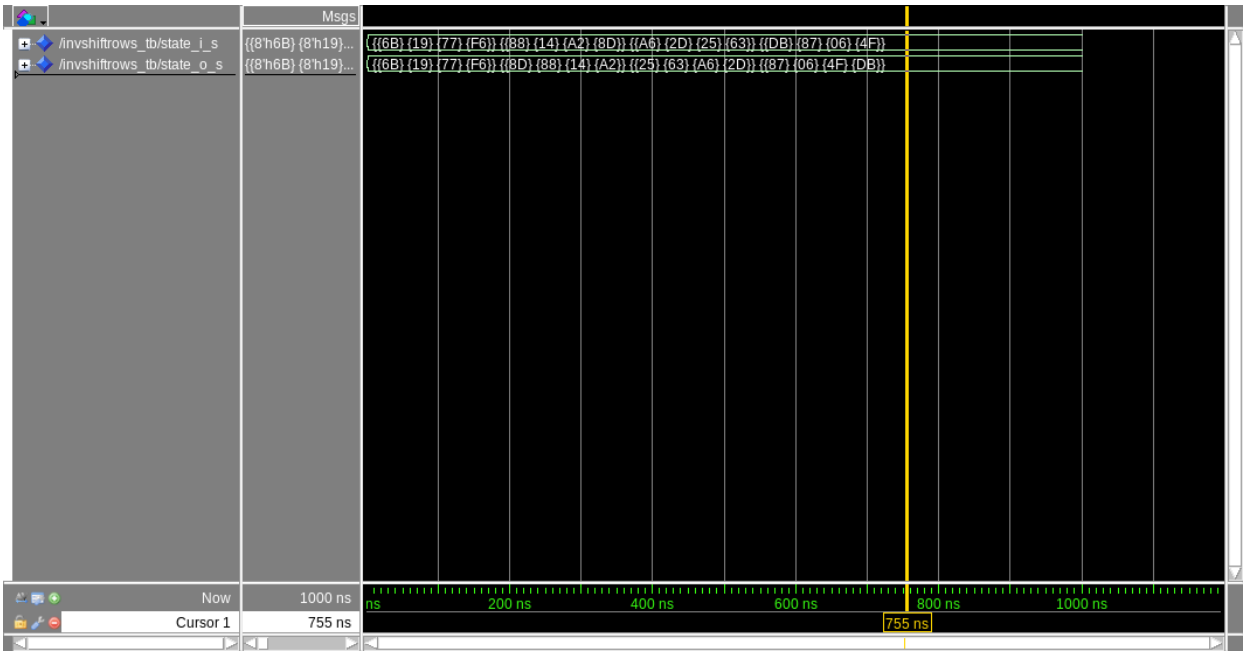

FIGURE 2 – Résultat de la simulation du InvShiftRows

On récupère donc bien le résultat attendu dans *state_o_s* : *0x6b 19 77 f6 88 14 a2 8d a6 2d 25 63 db 87 06 4f.*

4 Implémentation du InvSubBytes

4.1 Principe

— **Objectif de la méthode :** La fonction InvSubBytes est une transformation non-linéaire, qui vient faire correspondre à chaque octet (xy) de la matrice d'entrée, un autre octet dans une matrice de correspondance appelée *S-box*, à l'aide des coordonnées (x,y) de l'octet d'entrée.

4.2 Réalisation du composant

Pour réaliser le composant, il n'y pas de logique particulière à implémenter, il s'agit premièrement de rentrer directement chaque correspondance octet par octet de *0x00* à *0xFF* dans un fichier *SubBytes*. .

Ensuite, il faut implémenter une entité *InvSubBytes*, qui à partir d'une matrice d'entrée, va venir réaliser la correspondance grâce à l'instance précédente *Sbox*, on y instancie donc le composant précédent à l'intérieur. Et par conséquent, on rédige aussi un fichier de configuration.

On remarque qu'il faut effectuer deux *generate* pour parcourir chaque élément de la matrice d'entrée (voir code correspondant).

4.3 Test Bench

De même qu'avec le composant *InvShiftRows*, on simule avec *ModelSim*, et l'entrée : *0x 6b 19 77 f6 88 14 a2 8d a6 2d 25 63 db 87 06 4f* qui est la sortie du InvShiftRows (Round 9) :

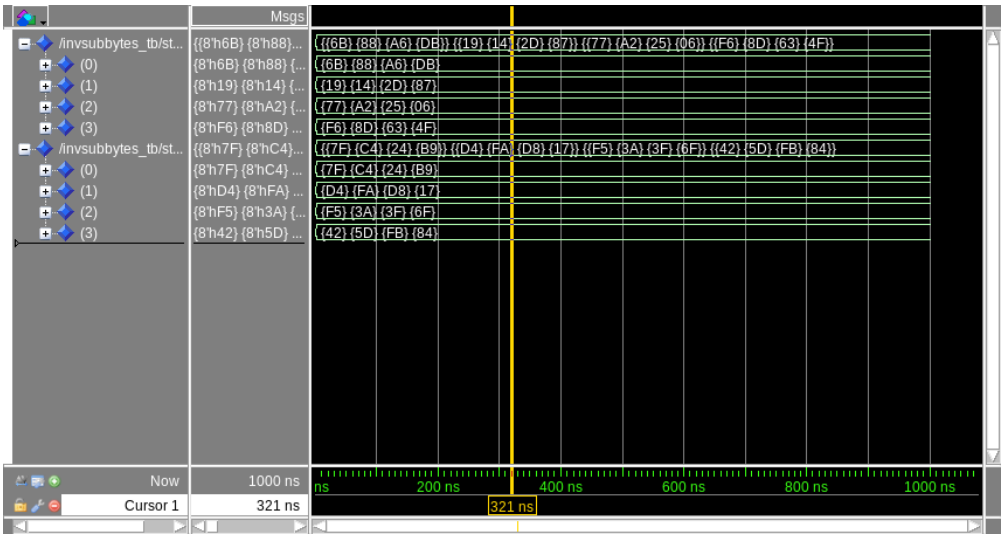

FIGURE 3 – Résultat de simulation du InvSubBytes

On remarque que l'on obtient : *0x 7f d4 f5 42 c4 fa 3a 5d 24 d8 3f fb b9 17 6f 84*, ce qui est bien le résultat attendu (signal du bas).

5 Implémentation du AddRoundKey

5.1 Principe

— **Objectif de la méthode :** La fonction AddRoundKey réalise une addition entre la matrice d'entrée et la clé de ronde actuelle fournie par la table *KeyExpansion_table* fournie par défaut dans les codes sources du projet.

Cette addition est effectué au sens ou-exclusif, on utilise donc le XOR pour réaliser cette opération.

5.2 Réalisation du composant

Pour réaliser le bloc *AddRoundKey*, il n'y pas de logique particulièrement complexe, il suffit de récupérer chaque élément de la matrice d'entrée, et effectuer un XOR avec l'élément de même indice (i,j) de la matrice d'*AddRoundKey*.

5.3 Test Bench

On simule encore avec *ModelSim*, et l'entrée : *0x8c 11 35 44 06 ad 44 88 de ca ec 83 aa 03 43 06* qui est le cipher text d'entrée (Round 10) et on utilise la clé de la ronde 10 : *0x e7 05 10 0b 8e 80 42 7e 78 4d 9b 0e 71 1a e1 65*,tout en implémentant un fichier test-bench (voir fichier joint en annexe, dossier BENCH) reprenant le même principe que InvShiftRows :

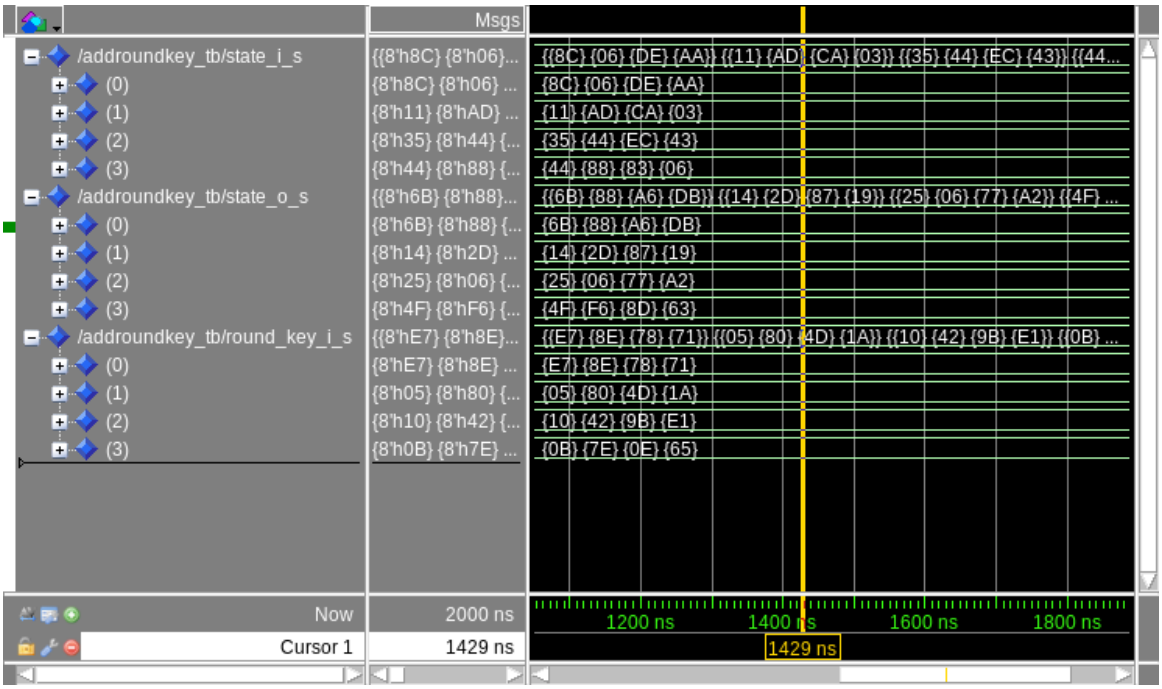

FIGURE 4 – Résultat de la simulation du AddRoundKey

On remarque que l'on obtient : *6b 14 25 4f 88 2d 06 f6 a6 87 77 8d db 19 a2 63*, ce qui est bien le résultat attendu à la fin de la ronde 10 dans *state_o_s*.

6 Implémentation du InvMixColumns

6.1 Principe

— **Objectif de la méthode :** La fonction InvMixColumns permet une transformation linéaire colonne par colonne de la matrice d'entrée. Pour cela on vient multiplier chaque coefficient colonne de la matrice d'entrée, qui sont traités comme des polynômes par une matrice donnée :

On fait donc attention à réaliser les opérations d'additions avec un XOR. Ensuite, prenons le cas du premier coefficient : pour multipler S0 avec 0x0E = 8 + 4 + 2, on va calculer le produit de S0 avec 2, qui est un décalage d'un bit vers la gauche si le bit fort de S0 est nul. Si il est égal à 1, on effectue un XOR avec le polynome *x1B*.On ajoute un 0 au poids faible.
On remarque que 4 = 2x2 et 8= 4x2. On peut donc réutiliser le résultat, appelé *data2*, précédent pour calculer la multiplication par 4 en utilisant le même principe que le calcul de *data2* mais avec *data2* à la place de S0.
Et de même pour la multiplication par 8 avec le résultat de la multiplication par 4. Ainsi on peut appeler le résultat de l'addition de ces trois résultats pour former la multiplication de S0 par 0x0E, tout simplement *dataE*.

En notant que : 0xb = 8 + 2 + 1, 0xD = 8 + 4 +1 et 0x9 = 8 + 1, on utilise le protocole précédent, et on peut calculer le premier coefficient de la première colonne S0' à l'aide de XOR pour l'addition. En itérant sur les 4 coefficients, on détermine alors la colonne de sortie :

6.2 Réalisation du composant

La logique exposée ci-dessus, permet l'implémentation d'un premier fichier permettant le calcul de la colonne de sortie en fonction d'une colonne d'entrée (**InvMixColumnsX**).

Le composant primaire du *InvMixColumns* est donc instancié. Désormais, il suffit d'effectuer une itération sur chaque colonne de la matrice d'entrée pour réaliser complètement le composant *InvMixColumns*, et d'utiliser un fichier de configuration pour utiliser le composant *InvMixColumnsX* :

6.3 Test Bench

De même qu'avec le composant *InvShiftRows*, tout en implémentant un fichier test-bench (voir fichier joint en annexe, dossier BENCH), on simule avec *ModelSim*, et l'entrée : *0x 74 6c e0 09 ad 7f 68 28 d2 15 e6 8b b0 40 15 ef* qui est la sortie du AddRoundKey (Round 9) :

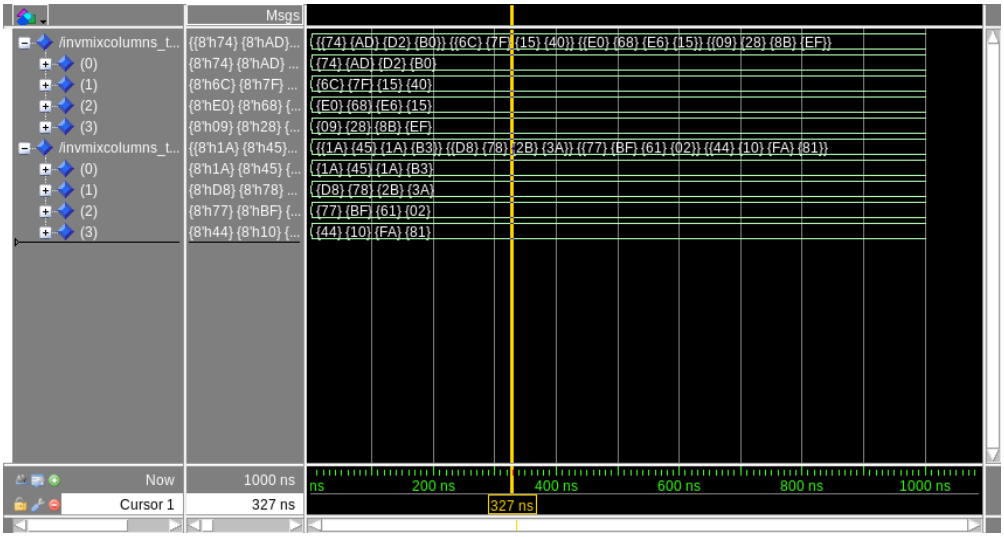

FIGURE 5 – Résultat de simulation du InvMixColumns

On remarque que l'on obtient : *0x 1a d8 77 44 45 78 bf 10 1a 2b 61 fa b3 3a 02 81*, ce qui est bien le résultat attendu (signal du bas).

7 Réalisation de l'InvAESRound

— **Objectif de la méthode :** La fonction InvAES permet de réaliser les différentes *round* du déchiffrement. La première round (Round 10) constitue une exécution du bloc AddRoundKey seul, ensuite il y a les round intérmédiaires (9 à 1) avec l'exécution des quatre blocs, successivement : InvShiftRows, InvSubBytes, AddRoundKey et InvMixColumns. Et finalement le round 0, qui est un round intermédiaire sans InvMixColumns.

7.1 Réalisation du composant

Pour réaliser ce bloc, il faut considérer qu'en entrée il y ait deux paramètres *enableRound_i* pour activer le fait de faire la round intermédiaire ou seulement l'AddRoundKey, mais aussi un paramètre *enableInvMixColumns_i* pour activer ou non le composant InvMixColumns dans la ronde.

Dans le code cela se traduit par une sélection de l'entrée du AddRoundKey et/ou une sélection de la sortie de pour l'InvAESRound :

enableRound_i	enableInvMixColumns_i	Entrée AddRoundKey	Sortie InvAES
0	0	Entrée InvAES	Sortie AddRoundKey
1	1	Sortie InvSubBytes	Sortie InvMixColumns
1	0	Sortie InvSubBytes	Sortie AddRoundKey

TABLE 1 – Gestion de l'entrée/sortie du InvAESRound

7.2 Test Bench

Pour faire le test bench, un fichier de configuration est évidemment rédigé pour les composants, et 3 tests se rapportant au trois cas du tableau précédent sont effectués :

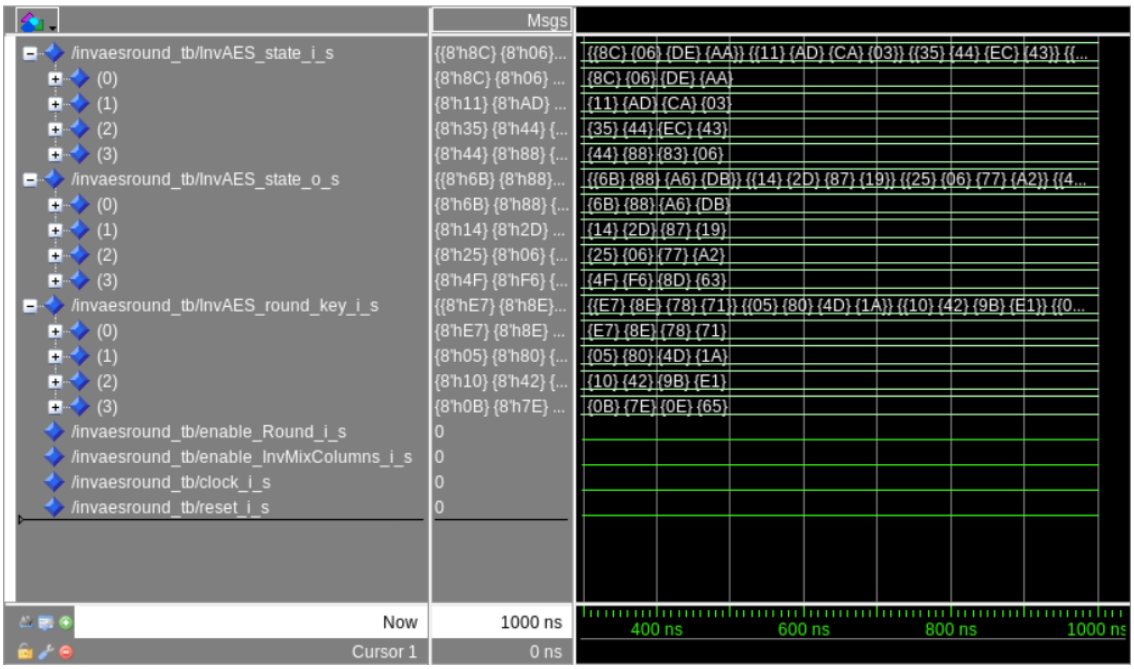

FIGURE 6 – Simulation Round 10 - AddRoundKey seul

On obtient bien le résultat attendu après le round 10 comme vu lors de l'implémentation du bloc AddRoundKey : *0x 6b 14 25 4f 88 2d 06 f6 a6 87 77 8d db 19 a2 63.*

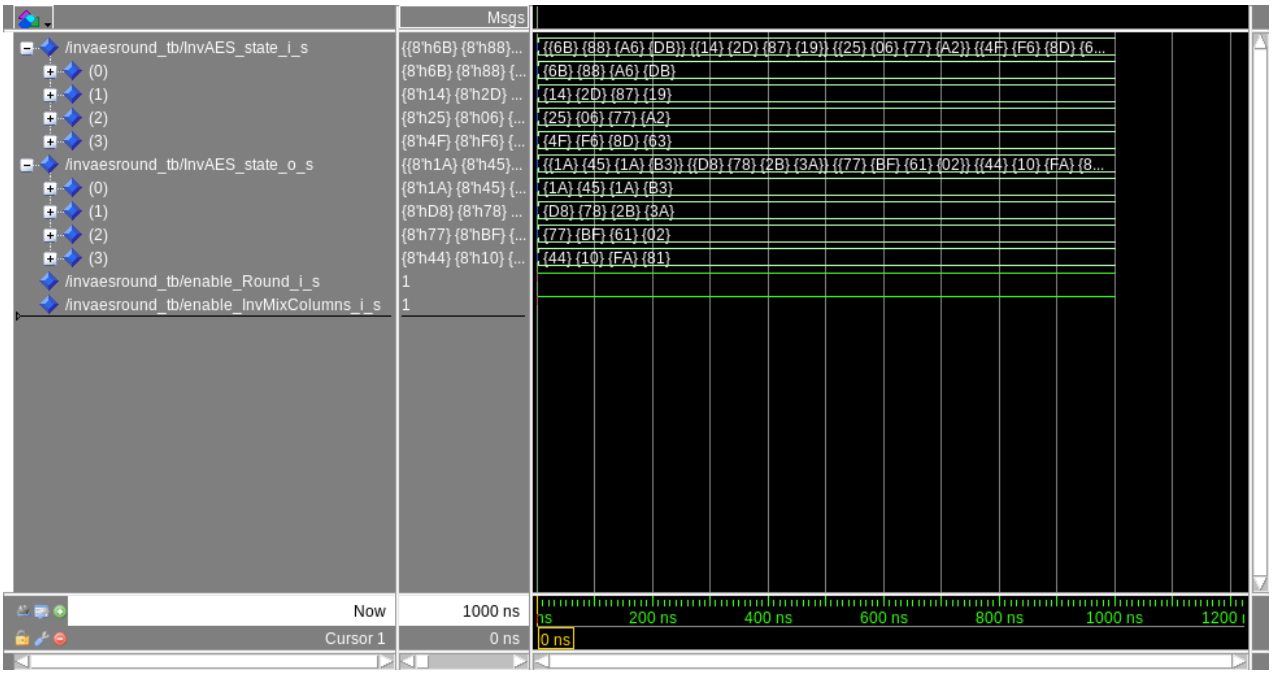

FIGURE 7 – Simulation Round 9 - Round intermédiare

On obtient bien *0x1a d8 77 44 45 78 bf 10 1a 2b 61 fa b3 3a 02 81*, qui est le résultat de l'InvMixColumns du round 9, ce qui est le résultat attendu.

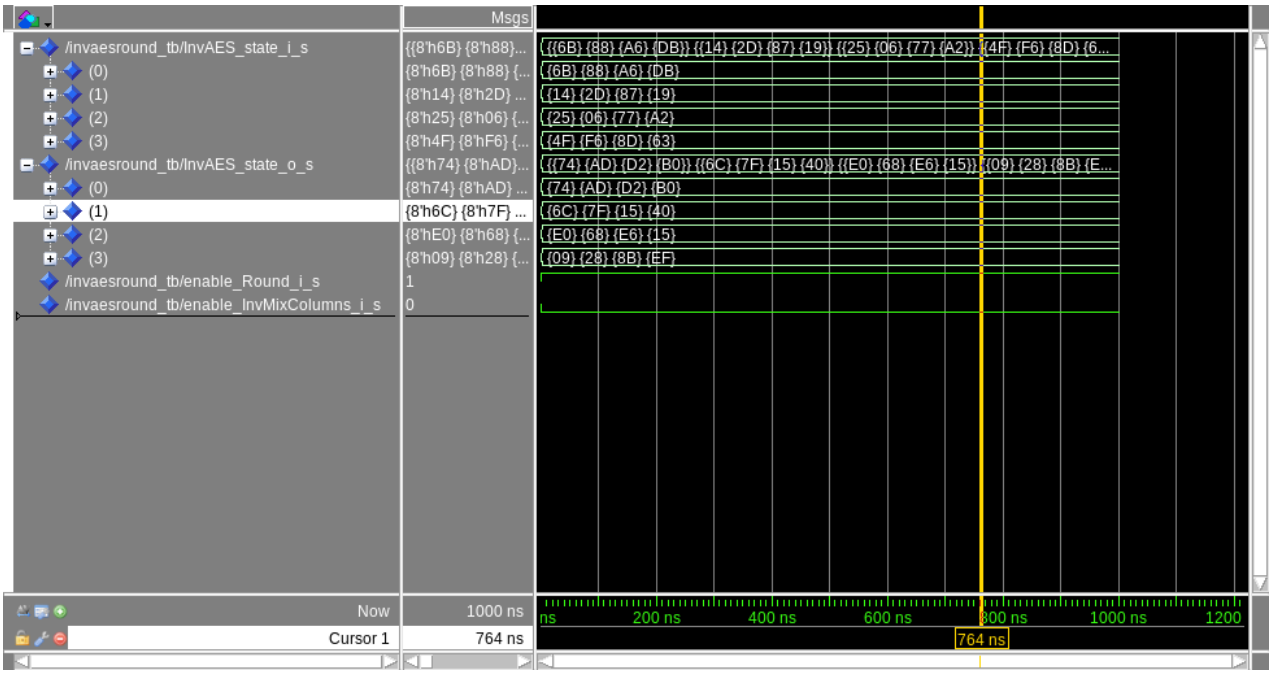

FIGURE 8 – Simulation Round 9 - Sans InvMixColumns

On obtient bien *0x 74 6c e0 09 ad 7f 68 28 d2 15 e6 8b b0 40 15 ef*, qui est le résultat de l'AddRoundKey du round 9, ce qui est le résultat attendu.

8 Réalisation de la machine d'états de Moore

8.1 Principe

La machine d'état finis gère l'exécution de l'algorithme InvAES, avec le lancement du déchiffrement lorsque start_i = 1, le basculement du signal aes_done_o à 1 lors de la fin du déchiffrement, ou sa réinitilisation à 0, la sélection de la clé de ronde en activant le composant Counter qui gère le compteur, Elle pilote aussi les blocs InvAESRound avec le choix du cipher text en entrée ou sa sortie et la KeyExpansion_table pour la clé de ronde de l'AddRoundKey.
Dans une machine de Moore, les sorties ne sont fonctions que de l'état présent. Dans une machine de Mealy, les sorties sont fonctions de l'état présent ainsi que des entrées. On va donc utiliser **une machine de Moore** car c'est l'état présent qui détermine les sorties (Round 10 / Round 9 à 1 / IDLE / Round 0).

Pour choisir l'état futur, on utilise principalement, la valeur du *start_i* pour être en IDLE, puis le compteur de round, permet de basculer du Round 10 à

(Round 9 à 1 - Même état), puis au Round 0.

On fixe ensuite les sorties de la FSM pour piloter le reste l'InvAES en fonction de la cohérence, des attendus.

8.2 Réalisation

Pour réaliser la FSM, on utilise le code présent dans le cours, qui constitue une bonne base adaptable pour la gestion des états, et on change d'état avec un front montant de l'horloge.

On vérifie ensuite la cohérence de notre FSM avec une horloge dans le test-bench, un *start*, et un compteur artificiel de ronde (voir le code source en annexe), pour s'assurer de la transition des états en fonction du compteur par exemple :

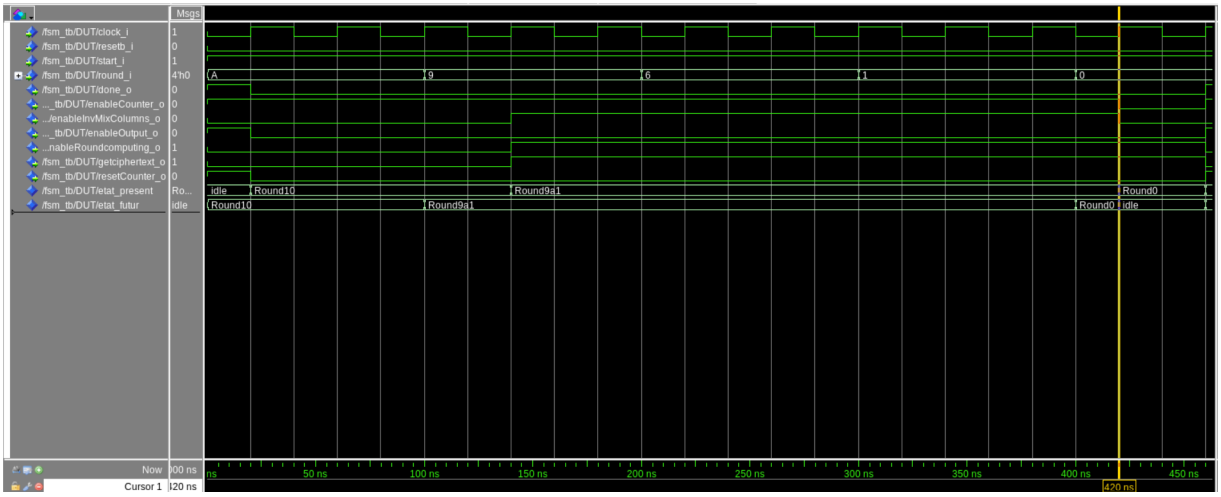

FIGURE 9 – Simulation de la FSM

On constate qu'au début, le *start* = 1 fait passer l'état présent à *IDLE* comme attendu, ensuite, cela active le compteur qui passe à 10d = 0xA, l'état futur passe donc à *Round10* comme attendu. Au front montant suivant on a une actualisation de l'état présent qui passe au Round 10 et effectue un paramétrage des sorties comme (*enableRoundcomputing_o,enableInvMixColumns*) = (0 ;0) comme attendu pour le Round 10 (car AddRoundKey seul), de plus on sélectionne bien le cipher text d'entrée de l'AES avec *getciphertext_o* = 1.

Enfin on entre dans les rounds intermédiaires lorsque le compteur est à 9 jusqu'à 1, et on passe au Round 0 à la fin avec le compteur à 0. On constate qu'il y a un délai entre le changement de valeur du compteur et l'actualisation de l'état présent qui correspond logiquement à un cycle d'horloge, car l'actualisation de l'état présent

est effectué avec un front montant. C'est un aspect qui me pose problème pour la réalisation de l'InvAES global.
En effet, si le compteur de round n'est pas synchronisé avec l'état présent, alors la clé de ronde en particulier n'est pas la bonne a utilisé lorsque InvAES Round est en fonctionnement. Ce qui nuise la cohérence de mon déchiffrement. C'est le **problème principal qui bloque la finalisation du projet**.

9 Réalisation de l'InvAES

L'architecture globale de l'InvAES est décrite dans le sujet.

9.1 Réalisation du composant

9.1.1 Intégration du Counter

Il a été nécessaire de modifier le Counter pour que le reset = 1 puisse mettre le compteur à 10, et que les coups d'horloge décrémente le compteur : voir code VHDL du compteur.

Enfin, il suffit d'instancier le composant de l'InvAES en intégrant les autres composants à l'intérieur et les signaux reliant ces composants.

9.1.2 Intégration du KeyExpansion_table

L'intégration de la KeyExpansion_table n'est pas complexe, il faut juste considérer que sa sortie est de type bit128, il faut donc convertir cela en *type_state* pour l'InvAESRound dans l'InvAES.

9.1.3 Registre de sortie

Pour la sortie *data_o*, lorsque *enableOutput* vaut 1, on prend la sortie de l'InvAESRound, que l'on traduit de *type_state* à bit128 :

Cela joue le rôle de registre de sortie.

9.1.4 Multiplexeur

Le multiplexeur permet la séléction de la source d'entrée du InvAESRound en fonction de l'état du round, est effectuée par une ligne de code qui attribue l'entrée de l'InvAESRound de façon conditionnelle.

9.2 Suite de l'entité InvAES

Pour le reste du code VHDL de l'*InvAES*, voir le code source en annexe, qui intégre tous les composants précédents (InvShiftRows, AddRoundKey, InvSubBytes,...) et relie l'ensemble des entrées/sorties selon la fig. 27.

9.3 Test Bench

On réalise ici un test bench, en configurant une clock, un start, un reset, et un fichier de configuration :

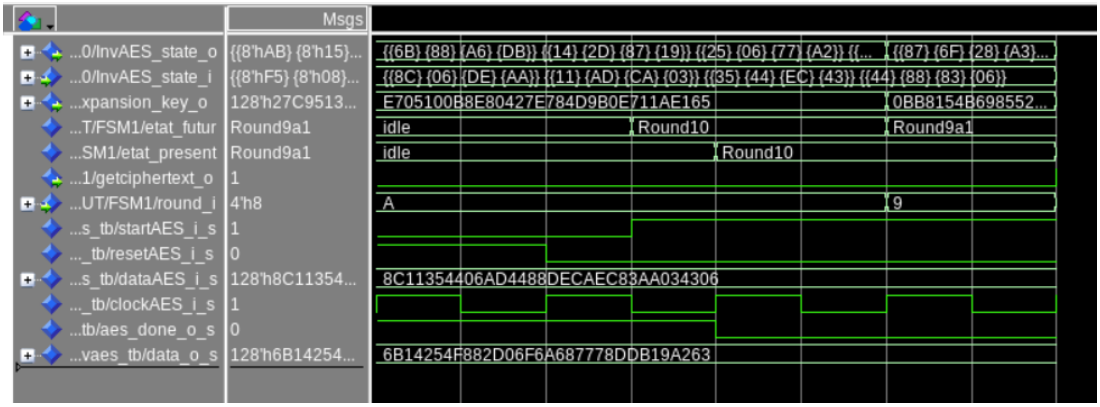

FIGURE 10 – Sortie de l'InvAES

On remarque qu'il y a un problème car la sortie n'est pas celle attendue malgré, les bons paramètres d'entrées. Le soupçon se porte sur le problème exposé dans la partie précédente. En effet.

On voit donc que la clé de ronde de la ronde 9 est utilisé pour la ronde 10, et ainsi de suite pour les autres rondes. C'est le délai entre l'actualisation des états présents qui crée ce problème. On pourra résoudre cela par l'ajout d'un délai artificiel, mais c'est impropre car il ne s'adapte pas à la fréquence de l'horloge et ralentirait trop l'InvAES, voir le désynchroniserait.
Une autre idée était de faire une actualisation des états présents sur chaque front montant et **descendant** mais il y aurait toujours un délai, mais plus petit (divisé par 2).

10 Conclusion

Finalement, tout les blocs élémentaires ont été crées et fonctionnement bien de façon cohérente et individuelle après vérification dans les test-bench. Le problème est peut être au niveau de la synchronisation de l'*etat_present* avec le compteur de ronde.

Après résolution de ce problème, il est très probable que l'entité InvAES global fonctionne de façon correcte.