MINES
Saint-Étienne

Une école de l'IMT

Projet Conception d'un Système Numérique

2018-2019

Omar KHATIB

Sommaire

Le projet de conception des systèmes numériques (PCSN) de ce semestre , a pour but d'élaborer l'algorithme de chiffrement inverse de l'AES (Advanced Encryption Standard) .L'AES est un algorithme de chiffrement symétrique approuvé pour la première fois en 2000 par la NSA et il est aujourd'hui le plus utilisé pour sa robustesse.

Pour le déchiffrement , l'algorithme AES se compose des 4 transformations *AddRoundKey* ,*InvSubBytes*,*InvShiftRows* et *InvMixColumns* réparties sur 11 rondes. Une première ronde exécute la transformation *AddRoundKey*,puis les 4 transformations sont exécutées 9 fois et finalement une dernière ronde sans la fonction *InvMixColumns*.

Pour déchiffrer , on utilisera initialement une clef secrète ,puis pour les rondes suivantes , des clefs générées par la fonction *Key Expansion* seront utilisées .

L'intérêt d'implémenter l'algorithme inverse de l'AES sur le hardware avec le langage de description VHDL repose sur la rapidité d'exécution des instructions par rapport aux autres langages de haut niveau.

L'algorithme peut-être résumé par le schéma suivant :

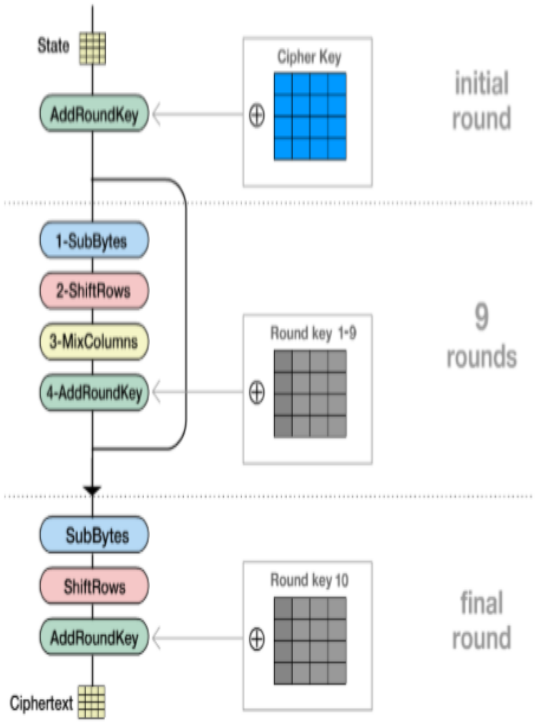


FIGURE 1 – Chiffrement AES 128bits

La simulation du code VHDl a été effectuée sur le logiciel ModelSim.Il faut noter également que dans certains screens de la simulation dans le rapport ,certains signaux disposent du suffixe _is/_os , cela a été corrigé dans le code et ont été remplacés par _s.

I-Description de l'entité INV AES :

L'entité *InvAES* qui représente le « top level » de notre circuit possède comme entrées : un bloc de 128 bits représentant le message à déchiffrer, un bit Start ainsi que le signal d'horloge et un Reset. En sortie, on obtient un bloc de 128 bits qui correspond au message déchiffré .

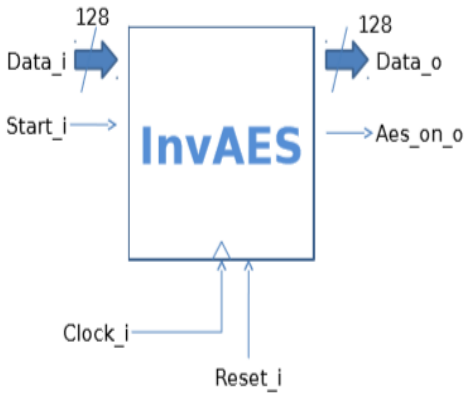


FIGURE 2 – Entrées / Sorties de l'entité InvAES

De façon plus détaillée, l'entité *InvAES* est composée des sous-entités suivantes :

-Une entité *FSM_INVAES* qui permet de donner les états de l'algorithme

-Une entité *Counter* qui permet de compter le numéro de ronde dans l'algorithme et qui le fournit à la machine d'état pour fixer l'état actuel , ainsi qu'à l'entité KeyExpansion_table pour récupérer la clef correspondante.

-Une entité *KeyExpansion_table* déjà fournie , qui permet de donner la clef *Ki* correspondante à la ronde *i.*

-Une entité *RTL_MUX* qui donne en sortie le texte d'entrée initial ou bien le bloc issu de la ronde précédente en fonction de la valeur du bit en entrée

-Une entité *InvAESRound* qui décide d'effectuer les 4 transformations(pendant 9 rondes) ou bien les 3 transformations (pendant la ronde 10) ou encore une seule (la transformation *InvAddRoundKey* durant la toute première ronde) selon les valeurs des entrées enableMixColumns_i et enableRoundcomputing_i ainsi que la valeur du compteur.

On donnera plus de détails sur le fonctionnement de chacune de ces entités plus tard dans le rapport.

Il faut aussi noter que j'ai pas utilisé le registre *RTL_REG* .

I-1 La transformation InvShiftRows:

Cette transformation consiste à décaler les coefficients de chaque ligne de la matrice de départ selon le schéma suivant :

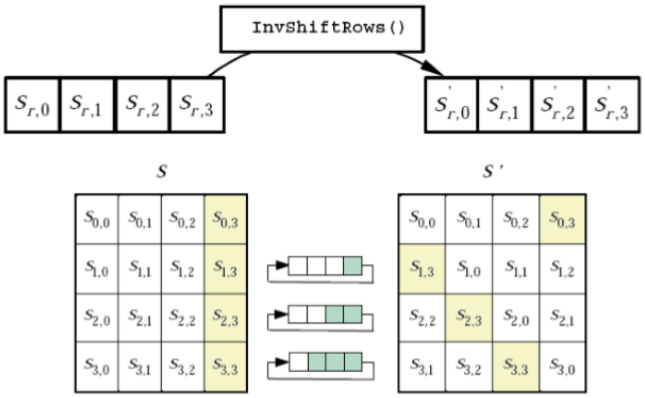


Figure 3 :Principe de la fonction InvShiftRows

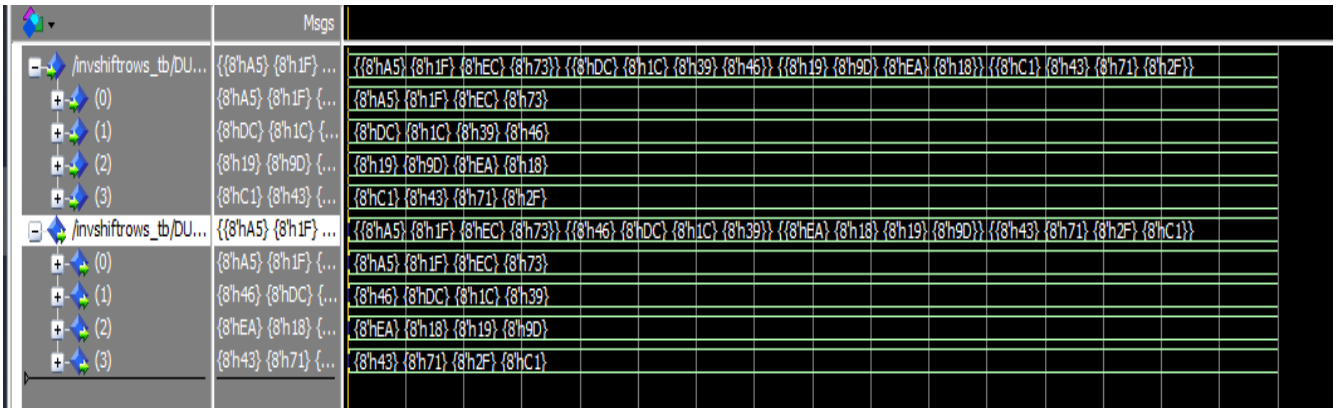


Figure 4 :Simulation de la fonction *InvShiftrows* sur ModelSim

I-2 La transformation InvSubBytes :

Cette transformation non linéaire s'effectue grâce à un tableau appelé S-Box. Chaque octet constituant notre matrice d'état va être remplacé par un autre en fonction de sa valeur

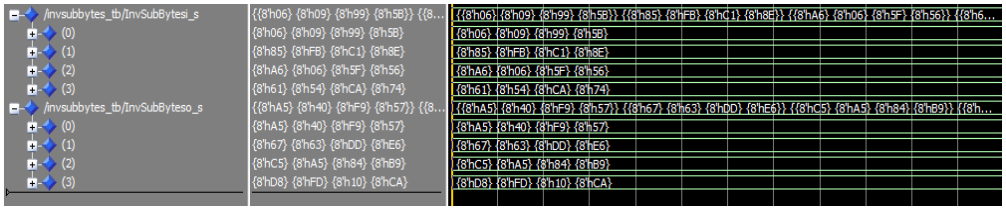


Figure 5 :Simulation de la fonction *InvSubBytes* sur ModelSim

06

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 6 :Illustration du principe de *InvSubBytes*

En guise d'exemple , après avoir subi une transformation InvSubByte , le premier octet 0x06 de notre matrice d'état devient 0xA5.

I-3 La transformation InvAddRoundKey :

Cette fonction consiste à appliquer sur chaque colonne de la matrice état un XOR bit à bit à l'aide de la clef de la ronde courante générée par le Key Expansion.

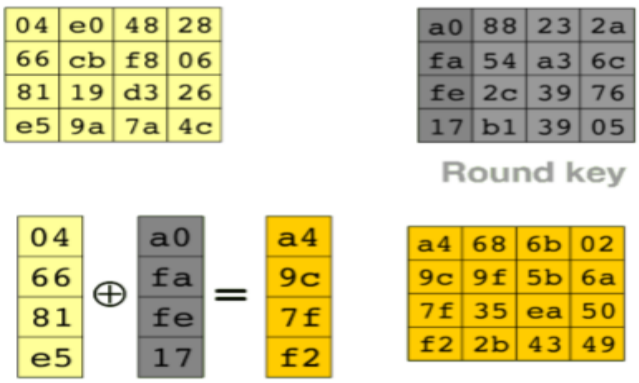


Figure 7 :Principe de la fonction *InvAddRoundKey*

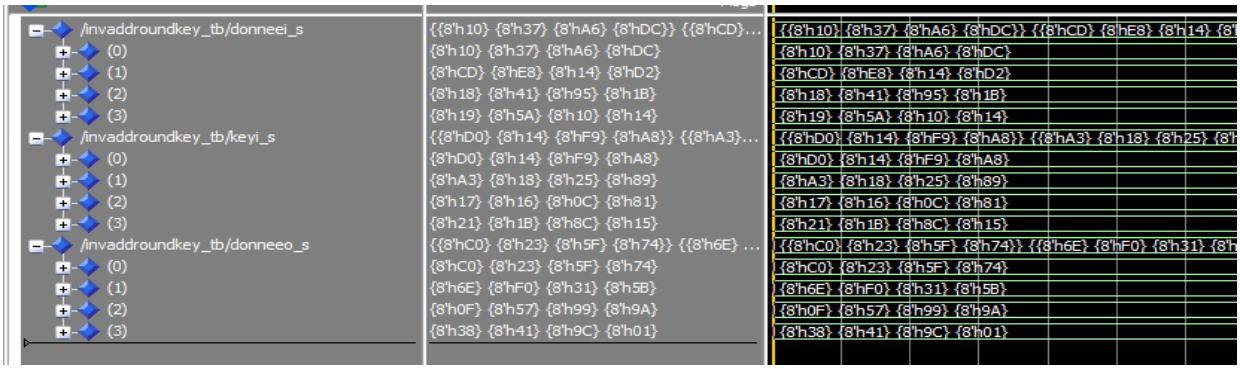


Figure 8 :Simulation de la fonction *InvAddRoundKey* sur ModelSim

I-4 La transformation InvMixColumns :

Cette transformation linéaire consiste à faire subir à notre matrice état un produit matriciel avec une matrice

M=

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

Les colonnes vont être multipliées modulo x^4+1 , et les additions sont effectuées à l'aide d'un XOR.

La difficulté d'implémentation du *InvMixColumns* consiste à effectuer les multiplications par les différents coefficients 0e 0b 0d….. Comme ces coefficients peuvent tous être exprimés comme une somme de 1 , 2,4 et 8 , on a opté pour la solution suivante : il s'agit de stocker dans des signaux intermédiaires le produit de notre colonne d'entrée par 2 , par 4 et par 8 , puis on en déduit les produits par 0e ,0b ,0d et 09. Il faut noter que pour multiplier la colonne par 2 , il suffit d'opérer un décalage à gauche puis effectuer un XOR judicieux.

```
Gen:for i in 0 to 3 generate
        --decalage Ã  gauche
        col2_s(i)(j)<=state_matrix_i(i)(j)(6 downto 0)&"0" xor "000"&state_matrix_i(i)(j)(7)&state_matrix_i(i)(j)(7)&"0"&state_matrix_i

        col4_s(i)(j)<=col2_s(i)(j)(6 downto 0)&"0" xor "000"&col2_s(i)(j)(7)&col2_s(i)(j)(7)&"0"&col2_s(i)(j)(7)&col2_s(i)(j)(7);

        col8_s(i)(j)<=col4_s(i)(j)(6 downto 0)&"0"xor "000"&col4_s(i)(j)(7)&col4_s(i)(j)(7)&"0"&col4_s(i)(j)(7)&col4_s(i)(j)(7);

--le reste des colonnes multipliées seront fonctiondes sommes des colonne2,colonne4,et colonne8
col9_s(i)(j)<=col8_s(i)(j) xor  state_matrix_i(i)(j);
colb_s(i)(j)<=col8_s(i)(j) xor  col2_s(i)(j) xor state_matrix_i(i)(j);
cold_s(i)(j)<=col8_s(i)(j) xor  col4_s(i)(j) xor state_matrix_i(i)(j);
cole_s(i)(j)<=col8_s(i)(j) xor  col4_s(i)(j) xor  col2_s(i)(j);
```

Et pour obtenir le résultat final , il ne reste plus qu'à remplir la colonne résultat avec les bons coefficients, puis de faire un **generate** sur j pour remplir toute la matrice :

```
state_matrix_o(0)(j)<=cole_s(0)(j) xor  cold_s(2)(j) xor colb_s(1)(j) xor  col9_s(3)(j);
state_matrix_o(1)(j)<=cole_s(1)(j) xor  cold_s(3)(j) xor  colb_s(2)(j) xor  col9_s(0)(j);
state_matrix_o(2)(j)<=cole_s(2)(j) xor  cold_s(0)(j) xor  colb_s(3)(j) xor col9_s(1)(j);
state_matrix_o(3)(j)<=cole_s(3)(j) xor cold_s(1)(j) xor colb_s(0)(j)xor  col9_s(2)(j);
```

Voici un exemple de simulation de la transformation *InvMixColumns :*

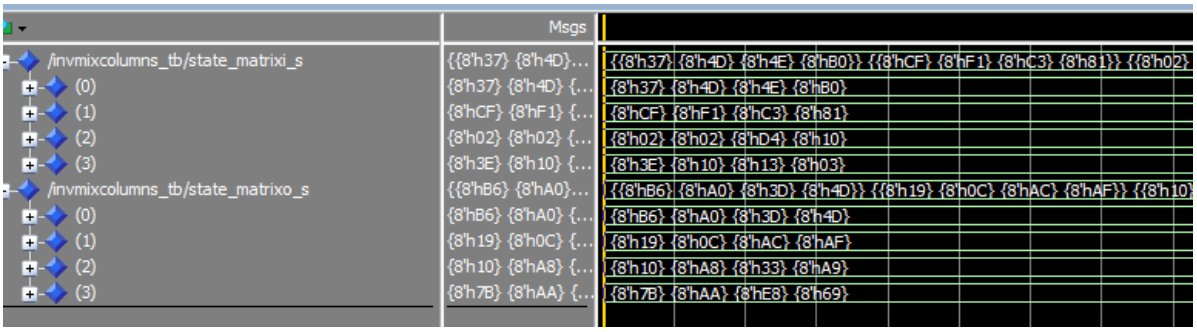


Figure 9 :Simulation de la fonction *InvMixColumns* sur ModelSim

I-5 Le compteur :

L'entité *Counter* sert à fixer la valeur de la ronde actuelle en fonction de l'état actuel décrit par la machine d'état. Le compteur a pour valeur initiale 10 puis se décrémente jusqu'à atteindre 0 pour ensuite être réarmé à 10 de nouveau. Il reçoit en entrée le signal d'horloge , un signal de reset ainsi qu'un bit enable_i qui lui permet de déclencher la décrémentation.

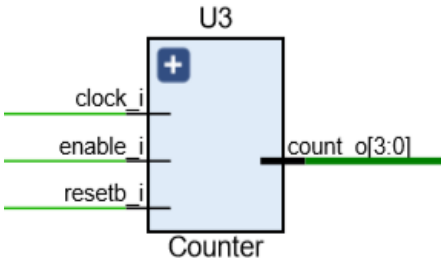


Figure 10 :Représentation de l'entité *Counter*

En sortie on dispose d'une valeur C codée sur 4 bits qui va en entrée de notre machine d'état ainsi que vers celle de l'entité KeyExpansion_table qui permet de fournir la clef correspondante à la ronde C.

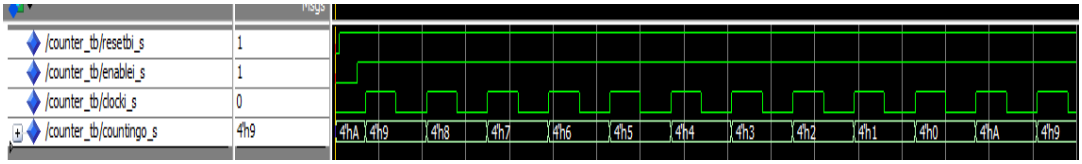


Figure 11 :Simulation du compteur sur ModelSim

I-6 Le multiplexeur :

L'*InvAES* dispose aussi d'un multiplexeur qui permet de choisir l'utilisation du bloc initial , c'est-à-dire le bloc de texte à déchiffrer ou bien le bloc de texte intermédiaire issu d'une des rondes de l'inverse AES (physiquement c'est la sortie de l'entité *InvAESRound).*

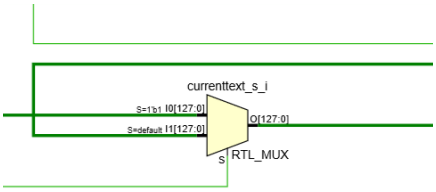


Figure 12 :Représentation fonctionnelle du multiplexeur

On peut remarquer que le multiplexeur possède une entrée, nommée getCipherText_i dans le code, codée sur un bit qui permet de choisir entre les deux textes. Par exemple si getCipherText_i='0' , la sortie prendra la valeur du texte initial, et si getCipherText_i='1' ,elle recevra le bloc de texte intermédiaire.

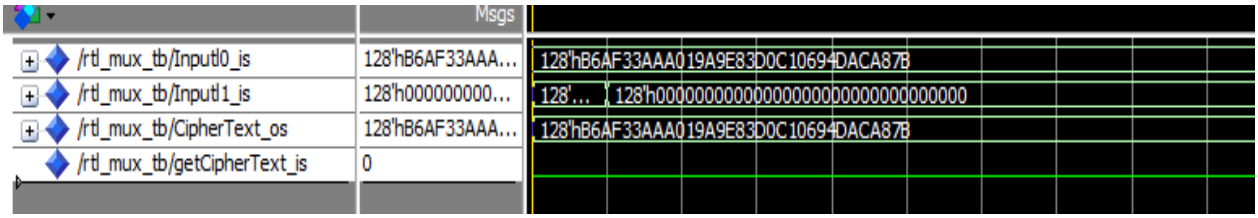


Figure 13 :Simulation du fonctionnement du multiplexeur sur ModelSim

I-7 L'entité *InvAESRound*:

C'est l'entité responsable du déroulement des fonctions selon le numéro du round.On peut résumer son fonctionnement par le diagramme suivant :

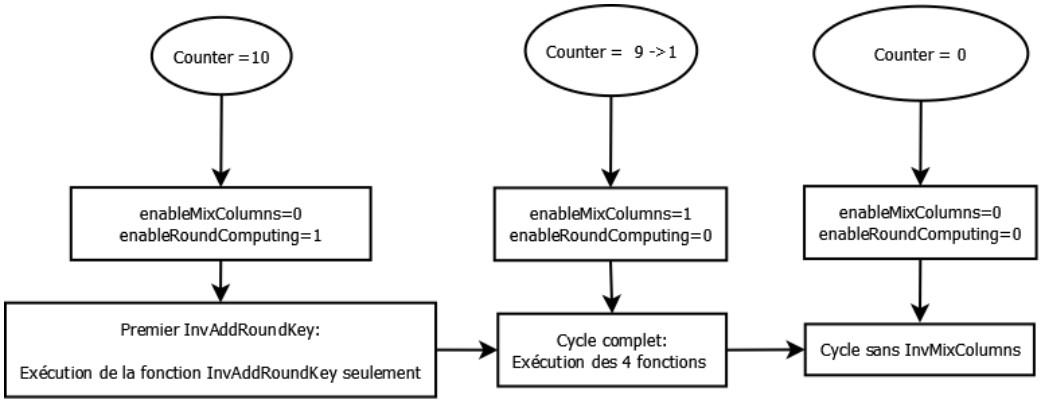


Figure 14 :Principe du fonctionnement de l'entité *InvAesRound*

L'entité prend en entrée le signal d'horloge et de reset , la clef correspondante au numéro de la ronde, le bloc de texte courant ainsi que deux entrées *enableInvMixcolumns*_i et *enableRoundComputing*_i. C'est la FSM(finite state machine) de notre InvAES qui dicte les valeurs de ces deux derniers bits en fonction de l'état présent.

Ainsi l'entité *InvAesRound* « encapsule » les 4 transformations et prend en entrée un bloc de 128 bits puis fournit en sortie un autre bloc de 128 bits qui a subi ces transformations, il faut donc qu'il opère une conversion de l'entrée en une matrice état (type-state) et une autre en sortie , afin de convertir la sortie des transformations en bloc de 128 bits.

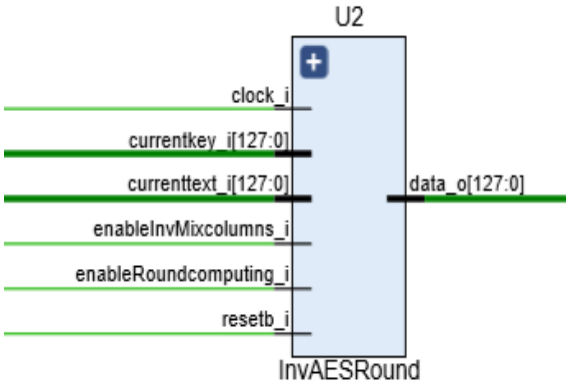


Figure 15 : Représentation fonctionnelle de l'entité *InvAESRound*

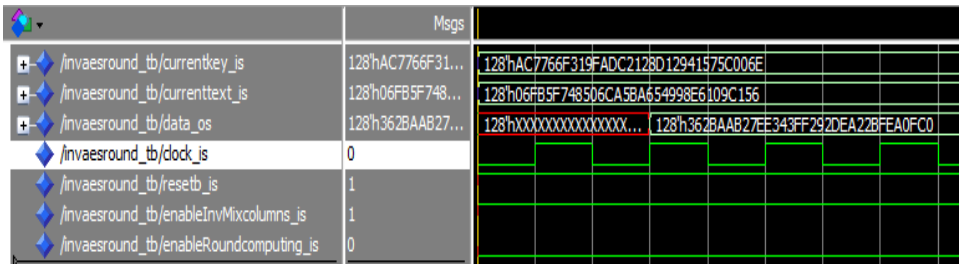


Figure 16 : Simulation de *InvAESRound* pour une round du Cycle Complet(*enableInvMixcolumns*_i=1)

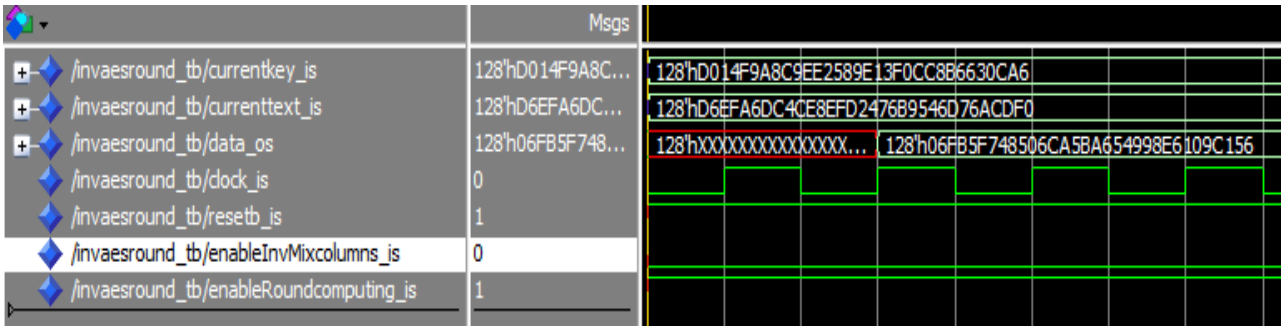


Figure 17 : Simulation de *InvAESRound* pour la ronde 10 (*enableRoundcomputing*_i=1)

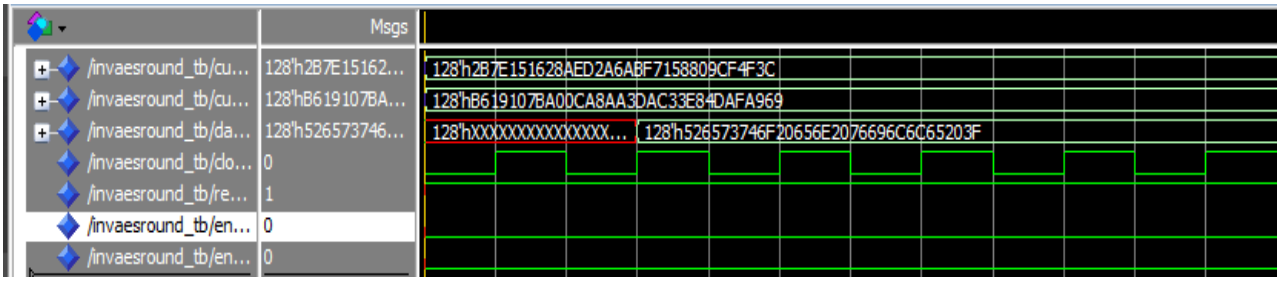
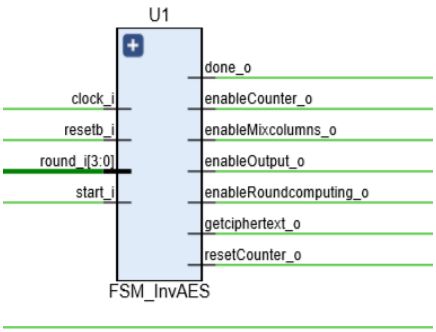


Figure 18 : Simulation de *InvAESRound* pour la ronde 0 (*enableInvMixcolumns*_i=0 et *enableRoundcomputing=1*)

I-8 L'entité InvFSM :



L'entité *InvFSM* a un rôle central dans notre architecture InvAES. C'est elle qui donne les états présents et futurs en fonction de la ronde actuelle. Elle reçoit en entrée le signal d'horloge , le signal de start du reset ainsi que la ronde codée sur 4 bits.

Les différents états utilisés
sont :{*Reset*,*Inacti,fpremierInvRK*,*CycleComplet*,*CycleSansInvMixColumns*,*Fin*} selon le diagramme suivant :

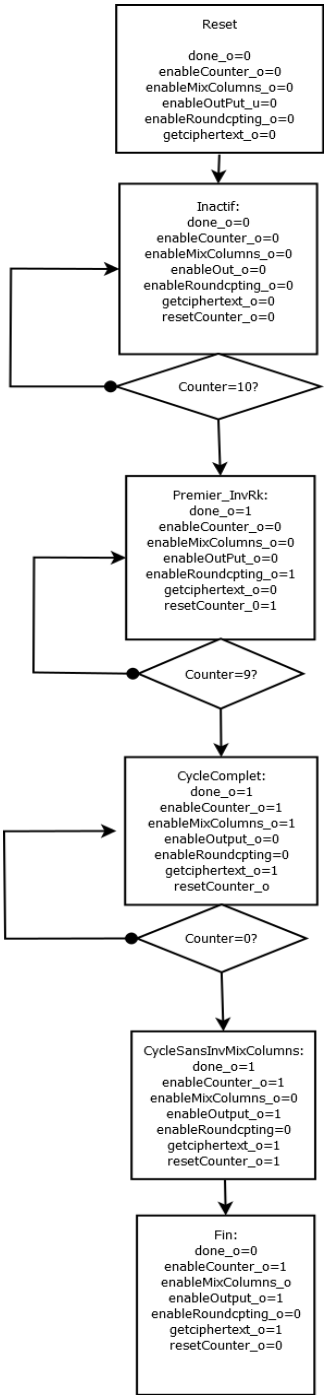


Figure 19 :Diagramme représentant la machine d'états de notre *InvAES*

Quand le *resetCounter_i* est à 0 le compteur est à 10.Dans le cas où *enableCounter_o*=1, le compteur se décrémente , on rappelle que la sortie *enableCounter_o* est reliée à l'entrée du Counter *enable_i.* A la ronde 0, un premier *InvAddRoudKey* est exécuté en mettant la sortie *enableRoundcpting_o* à 1 .Pendant les rondes 1 jusqu'à 9 , la sortie *enableInvMixColumns_o* est mise à 1 et *enableRoundcpting_o* à 0 afin d'exécuter les 4 transformations, puis

finalement , la FSM met le *enableInvMixColumns_o* à 0 et le *enableRoundcpting_o* à 0 pour n'exécuter que les 3 transformations.

L'implémentation de *l'InvFSM* en VHDL repose sur 3 processus :le premier est séquentiel et permet de passer à l'état futur à chaque coup d'horloge , le deuxième est combinatoire et fixe les conditions pour passer d'un état à un autre en fonction de la valeur du compteur , le dernier est combinatoire aussi et son rôle est de déterminer les sorties des états correspondants.

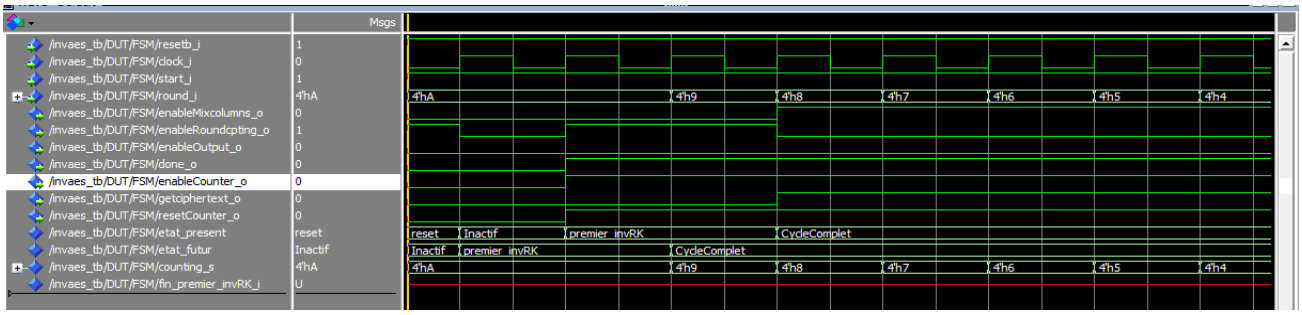


Figure 20 :Simulation de *l'InvFSM* sur ModelSim

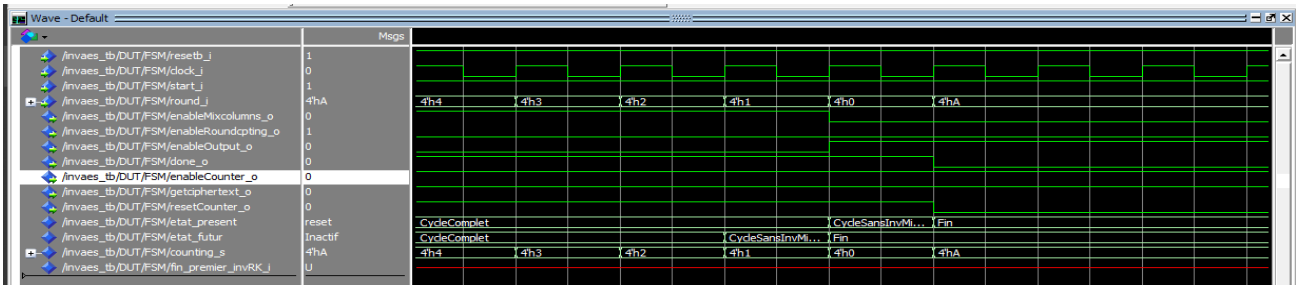
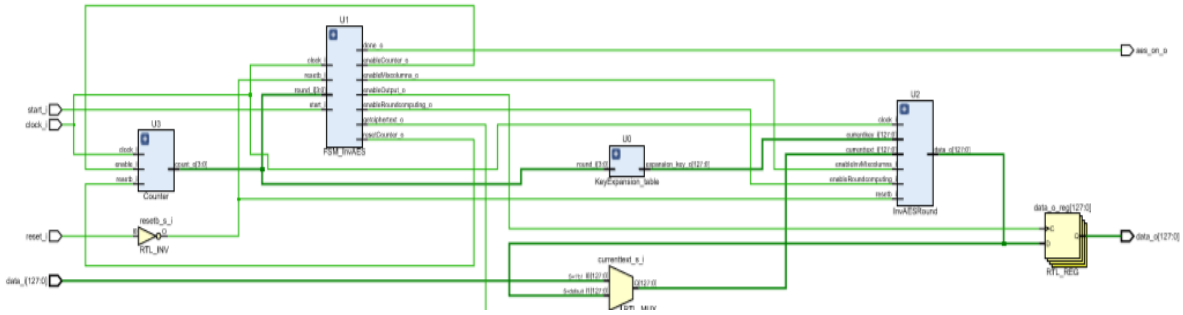


Figure 20 bis :Simulation de *l'InvFSM* sur ModelSim

On peut remarquer que sur la simulation les différents états sont synchronisés avec les valeurs du compteur ,ce dernier lui-même change de valeur sur chaque front montant de l'horloge.

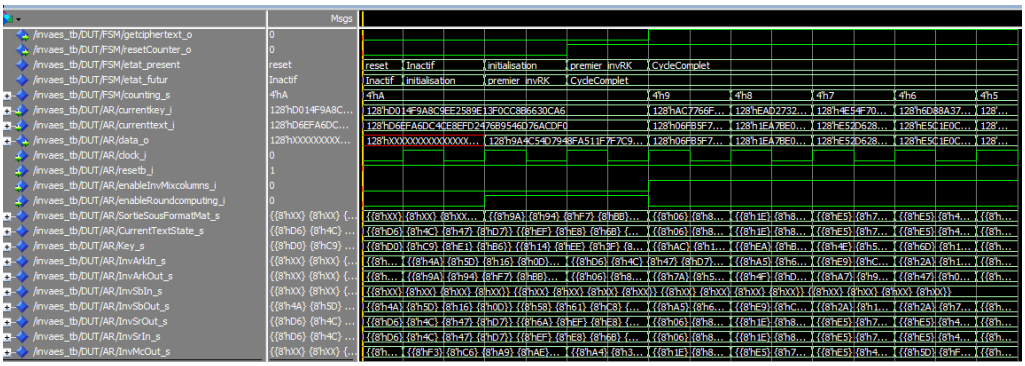
II-Simulation de l'entité InvAES :

Dans le fichier source InvAES.vhd , les différentes instances des sous -entités décrites plus haut sont reliées entre elles , en prenant garde à la dépendance des signaux entre les différentes entités



La simulation nous donne le résultat suivant :

(Il faut noter que dans ma tentative de résoudre un problème lié à la synchronisation des états, j'ai rajouté un état initialisation entre inactif et premierInvRK mais cela n'a pas corrigé le problème)



Pour chaque ronde , *Key_s* correspond à la clef courante,*InvArkIn_s* à l'entrée de l'entité *InvAddRoundKey,InvSrIn_s* celle de *InvShiftRows . InvArkOut_s , InvSrOut_s ,InvSbOut_s, InvMcOut_s* prennent les valeurs respectives des sorties de InvAddRoundKey,InvShiftRows,InvSubBytes et *InvMixColumns*.

On remarque qu'on a pas réservé de signaux pour l'entrée de InvMixColumns ainsi que pour InvSubByte.En effet , comme la transformation *InvAddRoundKey* survient toujours avant la transformation *InvMixColumns(*si *InvMixColumns a lieu) , et InvShiftRows* avant SubBytes , on s'est contenté de diriger le signal *InvShiftRows*_s sur l'entrée de *SubBytes et InvAddRoundKey_s* sur celle de *InvMixColumns.*

Le signal *SortieSousFormatMat_s* comme son nom l'indique prend la valeur de la sortie de chaque Round.

En fonction des valeur de *enableRoundcomputing_i* et *enableInvMixcolumns_i* les valeurs de *InvArkIn_s* ,*InvArkOut_s* et *SortieSousFormatMat_s* vont changer.

En effet si (*enableRoundcomputing_i =1* et *enableInvMixcolumns_i=0) ,*notre currentText_i ne subira qu'un InvAddRoundKey et par conséquent son entrée recevra *currentTextState_s* comme entrée et la *SortieSousFormatMat_s* recevra la valeur de *InvArkOut_s.*

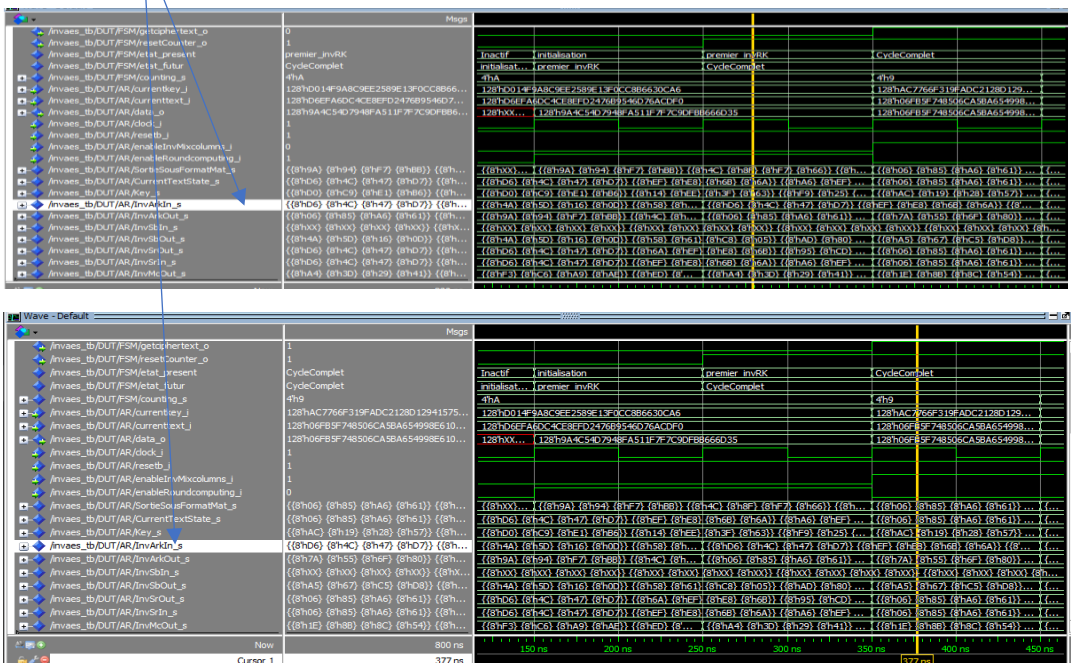
Si (*enableRoundcomputing_i =0* et *enableInvMixcolumns_i=1),* les 4 transformations seront faites et *InvSbOut* _s sera affectée à l'entrée du InvAddRoundKey et la *SortieSousFormatMat_s* recevra la valeur de InvMcOut_s.

Finalement si (*enableRoundcomputing_i =0* et *enableInvMixcolumns_i=1),* on exécutera que les 3 premières transformations et la sortie prendra la valeur de *InvArkOut_s.*

Commentaires :

Quoique toutes les sous-entités qui composent *l'InvAES* donnent le résultat adéquat, la simulation de *l'InvAES* n'aboutit pas au bon résultat. Il me semble que le problème vient du fait qu'à partir de la ronde 9 , l'entrée de *l'InvAddRound* garde la même entrée que lors de la ronde précédente , et ne la change qu'à la ronde après, alors que la clef de la ronde courante change et donc le résultat de l'invAESRound n'est pas le bon.

On peut voir que le résultat de l'entrée de *l'invAddRoundKey* reste inchangé même si la machine change d'état



Le mauvais résultat se prolonge donc tout au long de l'algorithme et aboutit à un mauvais déchiffrement.

Tentative de solution :

J'ai essayé de mettre un autre état à la fin du premier *InvAddRoundKey*(ronde 10) afin de synchroniser les résultats .Cependant , j'obtiens un nouveau décalage au niveau de *l'InvSubByte*.

3-Difficultés rencontrées et conclusion:

Tout au long du développement du projet, j'ai rencontré plusieurs types de difficultés. La première type repose sur le langage en lui-même. En effet, le fait que je sois habitué aux langages séquentiels comme le C par exemple rend la programmation en VHDL laborieuse.

D'une autre part, j'ai rencontré des difficultés dans l'entité *InvMixColumns* surtout quand il s'agissait de multiplier les colonnes par deux.

Finalement , *l'InvAES* m'a posé problème, surtout quand il s'agit de synchroniser les valeurs d'une ronde à l'autre. Malgré mes nombreuses tentatives , j'ai pas réussi à faire fonctionner l'InvAES.

Je pense que ce projet a été instructif sur deux niveaux : d'abord il nous a initié au langage VHDL ainsi que la description matérielle des circuits , puis c'était un exercice concret sur l'utilisation de l'AES.

4-Bibliographie :

-PDF Modélisation VHDL de l'Algorithme de Déchiffrement AES-

-An Introduction to VHDL : Jean-Max Dutertre,Olivier Potin,Jean-Baptiste Rigaud