Rapport de conception de la modélisation VHDL de l'algorithme de chiffrement AES-128

Julien MASTRANGELO

15 janvier 2021

Superviseur pédagogique : Olivier Potin

Table des matières

1 Introduction

On souhaite implémenter l'algorithme de chiffrement symétrique par blocs AES-128 (Advanced Encryption Standard), un standard de chiffrement actuellement considéré comme sûr pour protéger des documents secrets [1] en VHDL, un langage de description matériel.

Ce rapport débute par une vision générale de l'algotrithme AES-128, puis nous nous attarderons sur ses différentes composantes et leur conception : l'AESRound, le KeyExpansion ; puis nous les rassemblerons afin de manière cohérente afin de former l'AES.

Pour réaliser ce projet, on s'appuie sur le logiciel de simulation SystemVision, sur le sujet, la bibliothèque VHDL CryptPack.vhd et une partie du code source fourni par notre enseignant Oliver Potin [2] ainsi que sur le support de ce dernier.

2 Le chiffrement AES

La version d'AES que nous implémentons est un algorithme prenant en entrée un message clair à chiffrer de 128 bits ainsi qu'une clef secrète de 128 bits et renvoyant un message chiffré de 128 bits, qu'il sera possible de déchiffrer uniquement avec la clef secrète. L'algorithme se décompose en 11 cycles de 3 types différents durant lesquelles s'exécutent diverses opérations sur le message en cours de chiffrement et la clef secrète (cf. figure 1) :

— Cycle 0 : Application de la fonction addRoundKey à la clef secrète.
— Cycle 1 à 9 : Application des fonctions subBytes, shiftRows, mixColumns, addRoundKey.
— Cycle 10 : Application des fonctions subBytes, shiftRows, addRoundKey.

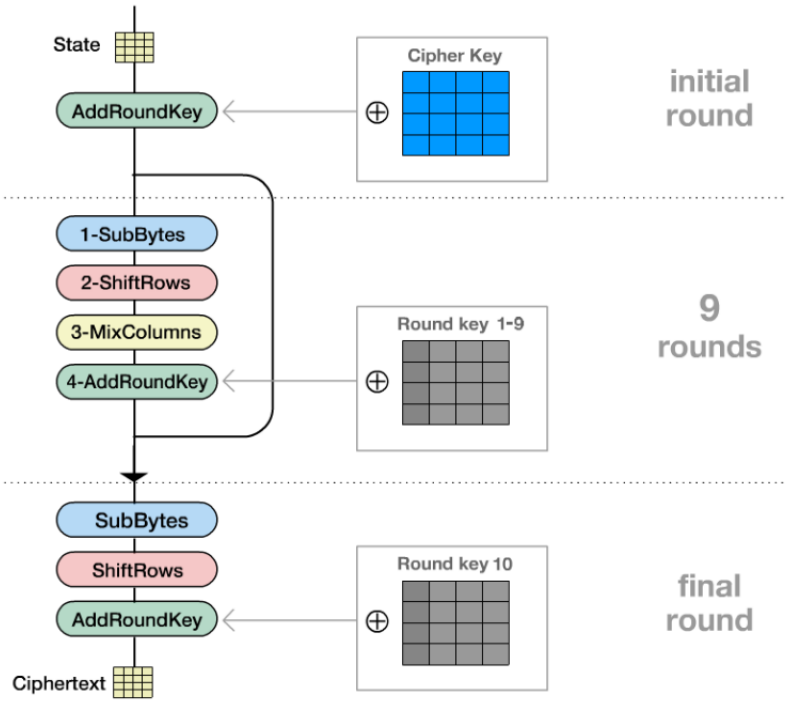

FIGURE 1 – Synoptique du chiffrement AES-128 bits

C'est l'entité AESRound qui coordonne les entités qui exécutent les fonctions composant les cycles.

L'entité KeyExpansion_I_O est chargée de générer une clef unique pour chaque ronde à partir de la clef secrète d'entrée.

C'est l'entité AES qui coordonne éléments entre eux.

3 AESRound

Le rôle d'AESRound est d'effectuer les cycles de l'algorithme (les "rounds" sur la figure 1).

L'entité prend donc en entrée les 128 bits de l'état actuel du message et les 128 bits de la clef du cycle. Les signaux d'entrées logiques enableMixcolumns_i et enableRoundcomputing_i permettent de sélectionner lequel des 3 types de cycles possible va être effectué. L'entrée clock_i est une horloge permettant le changement synchrone du signal de sortie data_o. L'entrée resetb_i est un signal d'initialisation, actif à l'état haut.

La sortie data_o est le résultat de l'AESRound : l'état du message après avoir effectué le cycle.

```
 9  entity AESRound is
10      port(
11          text_i          : in  bit128;
12          currentkey_i        : in  bit128;
13          data_o          : out bit128;
14          clock_i     : in  std_logic;
15          resetb_i        : in  std_logic;
16          enableMixcolumns_i  : in  std_logic;
17          enableRoundcomputing_i : in  std_logic);
18  end entity AESRound;
```

FIGURE 2 – Entrée/sorties de la fonction AESRound

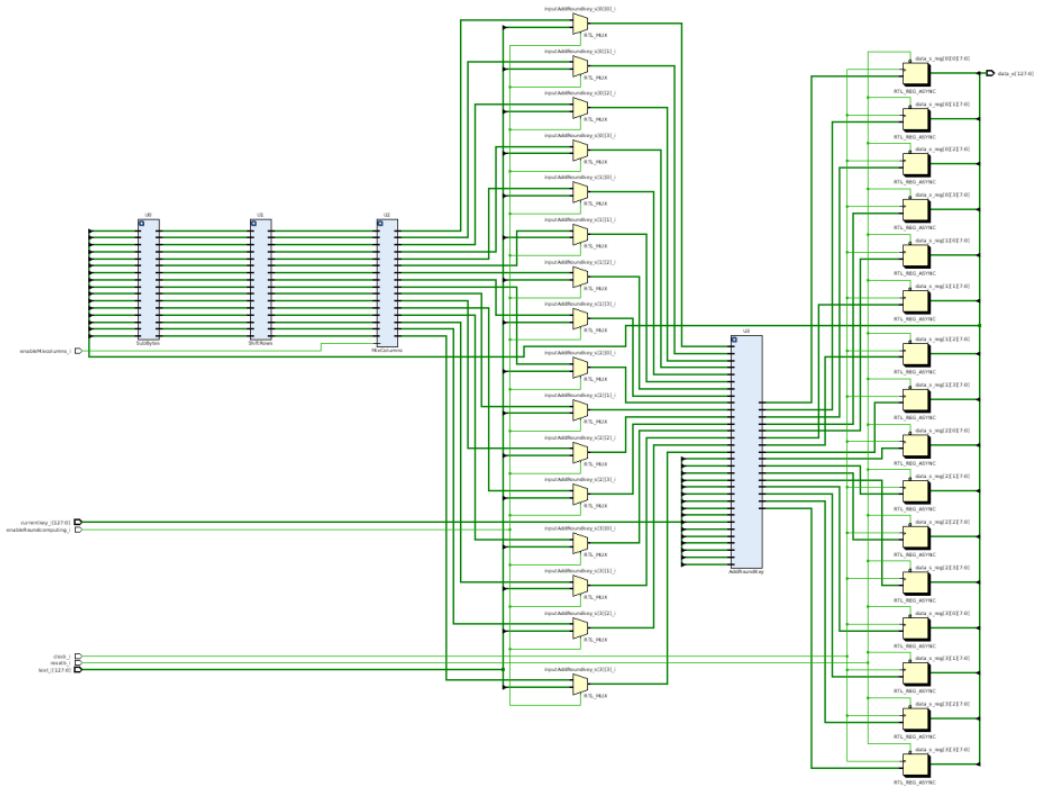

FIGURE 3 – Synoptique de la fonction AESRound

3.1 subBytes et Sbox

L'entité subBytes prend 128 bits (16 octets) en entrée sous la forme d'un type_state : une matrice de 4x4 octets ; et substitue chaque octet selon une tableau de substitution appelé SBox (cf. figure 4).

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

FIGURE 4 – Table de substitution SBox

3.1.1 Implémentation

La SBox a une modélisation dataflow. Elle est implémentée sous la forme d'une tableau, ce qui rend la compréhension du code VHDL claire. Par exemple, l'octet $0x53$ sera substitué par l'octet situé à la $0x5^{ème}$ ligne, $0x3^{ème}$ colonne de la SBox : ici $0xed$.

3.1.2 Simulation de subBytes

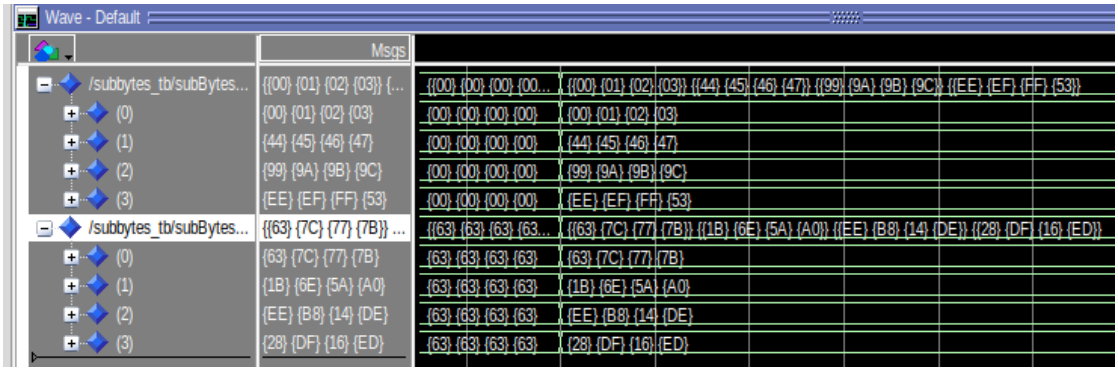

FIGURE 5 – Simulation de subBytes

L'entité subBytes a bien le comportement désiré.

3.2 ShiftRows

L'entité shiftRows prend en entrée une matrice de 4x4 octets et effectue une permutation cyclique comme explicité sur la figure 6.

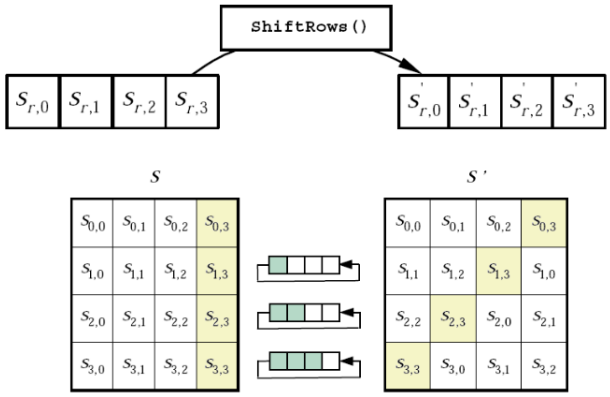

FIGURE 6 – Permutations effectuées par shiftRows

3.2.1 Implémentation

L'implémentation VHDL se fait très facilement à l'aide de 2 boucles *for* imbriquées pour permuter un par un les 16 octets de données (description comportementale).

3.2.2 Simulation

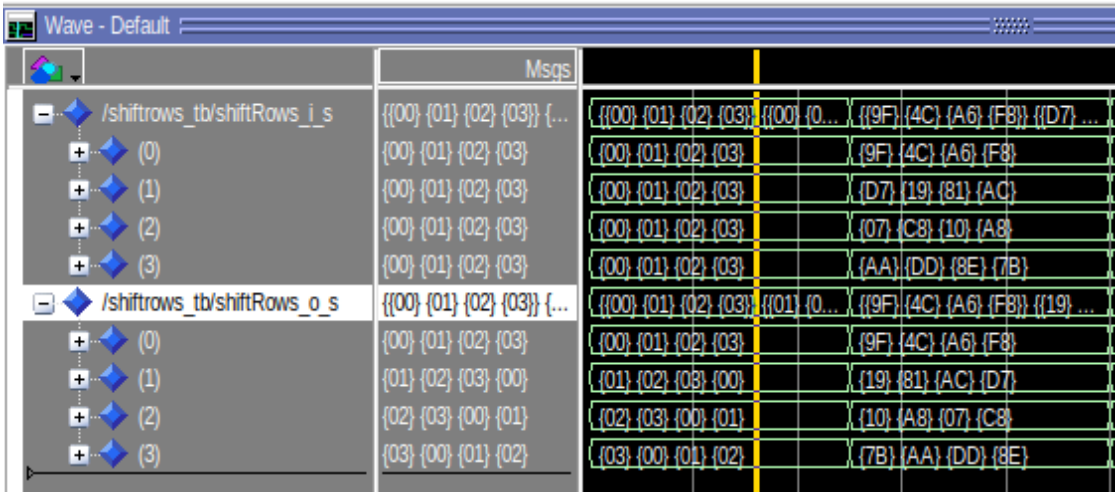

FIGURE 7 – Simulation de subBytes

L'entité shiftRows a bien le comportement désiré.

3.3 mixColumns

Le mixColumns est une entité qui prend en entrée une matrice 4x4 octets et qui effectue un produit matriciel entre chaque colonne de la matrice d'entrée et une matrice constante composé de 1, 2 et 3 (cf. figure 8). Dans l'espace dans lequel on travail, $GF(2^8)$, l'addition est un XOR ; les multiplications par 2 et 3 s'effectue comme décrit ci-dessous.

Multiplication par 2 dans $GF(2^8)$

- La valeur de l'octet est décalée à gauche : $b'11010100 << 1 = b'10101000$.
- Si le bit de poids fort vaut $'1'$, on effectue un XOR ($\oplus$) avec $b'00011011$: $b'10101000 \oplus b'00011011 = 10110011$

Multiplication par 3 dans $GF(2^8)$
On remarque que $3 = b'11 = 10 \oplus 01$.
D'où $03 \cdot S_{x,c} = (02 \cdot S_{x,c}) \oplus S_{x,c}$

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

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

FIGURE 8 – Produit matriciel appliqué à chaque colonne de la matrice d'entrée du mixColumns

3.3.1 Implémentation

Pour implémenter le mixColumns, il a été choisi d'utiliser une entité intermédiaire appelée mixMul prenant en entrée une colonne d'octets afin de réaliser le produit matriciel entre cette colonne et la matrice constante décrite ci-dessus (description comportementale). L'entité mixColumns est donc simplement composée de 4 mixMul (description structurelle).

3.3.2 Simulation

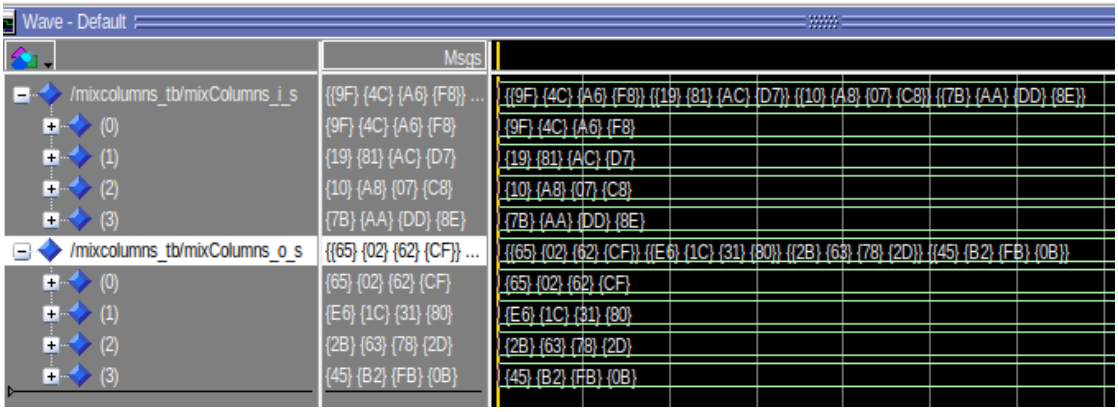

FIGURE 9 – Simulation de mixColumns

L'entité mixColumns a bien le comportement désiré.

3.4 addRoundKey

L'entité addRoundKey effectue simplement un XOR entre l'état actuel et la clef de la ronde actuelle.

3.4.1 Implémentation

L'addRoundKey prend en entrée l'état actuel du message sous forme d'une matrice de 4x4 octets ainsi que la clef sous forme d'une matrice de 4x4 octets. On XOR simplement un a un les octets de l'état et de la clef pour produire la sortie de l'addRoundKey (description comportementale).

3.4.2 Simulation

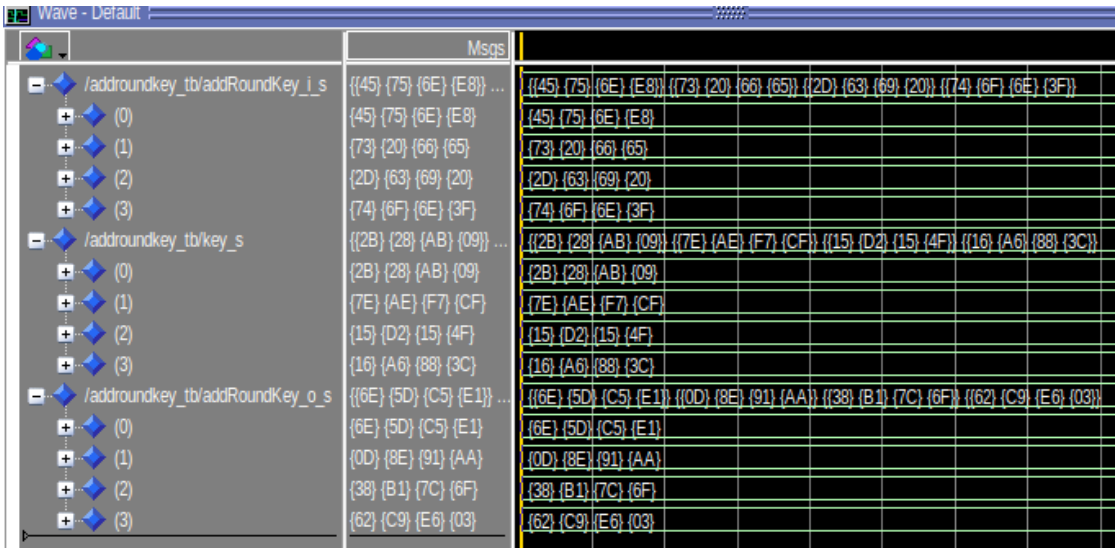

FIGURE 10 – Simulation d'addRoundKey

L'entité addRoundKey a bien le comportement désiré.

3.5 AESRound

3.5.1 Implémentation

L'entité AESRound converti le message et la clef fournies en bit128 en 2 matrices de 4x4 octets, puis branche les différents composants entre eux qui permettent d'effectuer les cycles afin de fournir les états intermédiaires. On y trouve aussi une boucle de rétro-action afin de calculer les cycles 1 à 9 (la sortie du cycle $n \in [1, 8]$ est branchée à l'entrée du cycle $n + 1$) (description structurelle).

3.5.2 Simulation

Mon testbench est composé de 3 entités AESRound, afin de tester le fonctionnement d'AESRound sur les 3 types de cycles différents. J'ai utilisé les données fournies par le sujet [2] des cycles 0, 1 et 10.

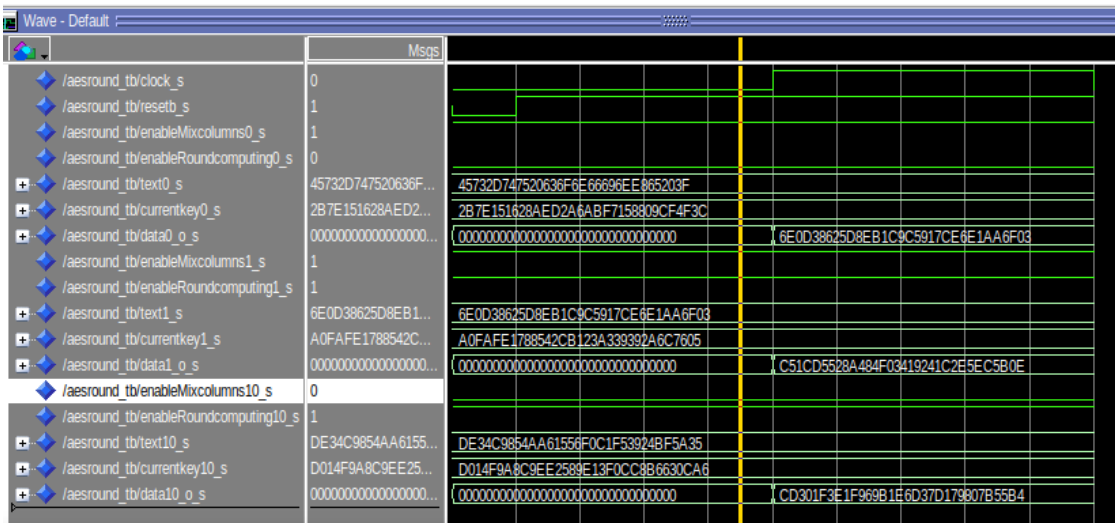

FIGURE 11 – Simulation des 3 types de cycles d'AESRound

L'entité AESRound a bien le comportement désiré pour chaque type de cycle.

4 KeyExpansion_I_O

L'entité KeyExpansion_I_O s'appuie sur une machine de Moore (FSM_KeyExpansion) et l'entité KeyExpander, pour générer une clef unique pour chaque ronde à partir de la clef secrète d'entrée puis de la clef générée à au cycle précédant, ainsi qu'à partir de la fonction subBytes et d'une matrice Rcon (figure 12).

4.1 KeyExpander

L'entité KeyExpander fonctionne à partir du cycle 1 et prend en entrée la clef du round précédant *key_i* ainsi que 8 bits : une colonne de la matrice Rcon (cf. figure 12). En sortie, le KeyExpander renvoit la clef pour le cycle actuel : $CurrentKey\_o$.

01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

Rcon

FIGURE 12 – Matrice Rcon

4.1.1 Implémentation

On construit les 4 colonnes de la sortie du KeyExpander une par une. Pour construire la première colonne (cf. figure 13) :

1. On applique une rotation vers le haut sur la dernière colonne de key_i.
2. On substitue chacun des octets de la colonne obtenue à l'aide d'une SBox.
3. On effectue l'opération :
   (colonne avec la $1^{ère}$ colonne de key_i) XOR (colonne obtenue précédemment) XOR (Rcon(numéro du cycle mod 4)).

Pour former les 3 autres colonnes, on XOR la colonne précédente de la clef avec la colonne de même indice de la clef du round précédent.

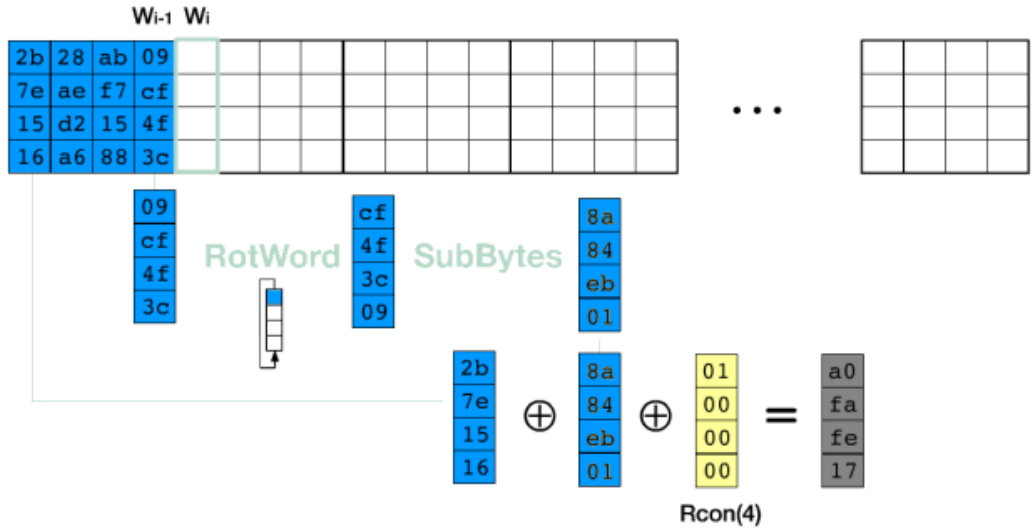

FIGURE 13 – Calcul de la première colonne (i mod 4 = 0)

4.1.2 Simulation

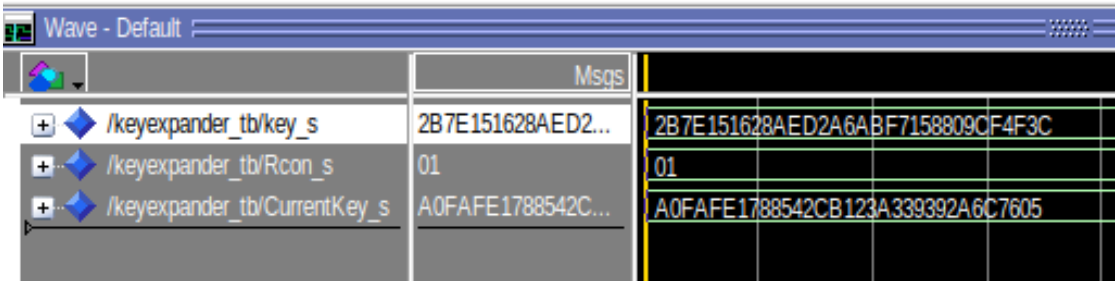

FIGURE 14 – Simulation du KeyExpander

L'entité KeyExpander a bien le comportement désiré.

4.2 FSM_KeyExpansion

La FSM_KeyExpansion est une machine de Moore qui détermine l'état actuel du KeyExpansion. En effet, le comportement du KeyExpansion varie selon le cycle actuel. L'état *init* correspond au cycle 0, l'état *count* aux cycles 1 à 9 et l'état *endState* au cycle 10.

4.2.1 Implémentation

La machine détermine l'état suivant (*next_state*) en fonction de l'état présent (*current_state* qui prend les valeurs : *init*, *count*, ou *endState*) et des signaux logiques (*start_i*, *resetb_i*, *counter_i*). A chaque front montant de l'horloge (*clock_i*), l'état actuel est mis à jour et prend la valeur de *next_state*. L'état présent permet de déterminer les valeurs que doivent prendre les signaux logiques *enable_o* et *resetb_o*, qui permettront à l'AESRound de déterminer quel type de cycle effectuer.

4.2.2 Simulation

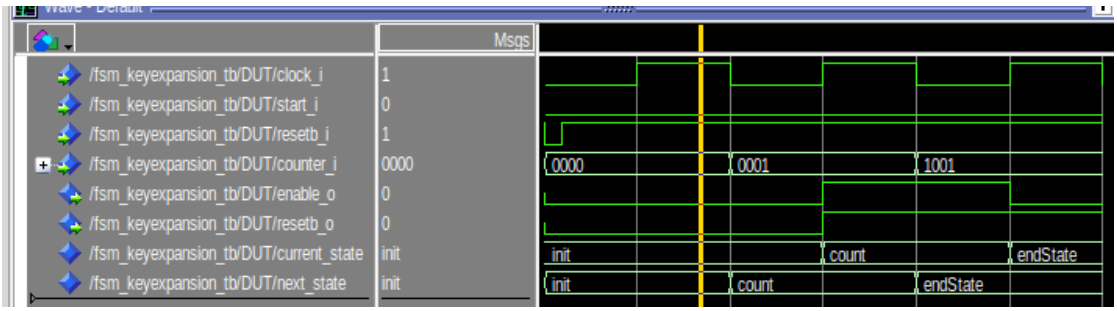

FIGURE 15 – Simulation de la FSM du KeyExpansion

L'entité FSM_keyExpansion a bien le comportement désiré.

4.3 KeyExpansion_I_O

KeyExpansion_I_O est l'entité qui produit les clefs d'encryption pour chaque ronde qui seront envoyé à l'AESRound. Pour cela, elle utilise FSM_KeyExpansion pour déterminer l'état actuel du chiffrement afin que le KeyExpander produise la bonne clef au bon moment.

4.3.1 Implémentation

Le KeyExpansion_I_O se découpe en 2 processus. Le premier processus détermine la colonne de Rcon qu'il faut envoyer au KeyExpander en fonction du compteur et de l'état indiqué par la FSM_KeyExpansion. Le deuxième processus recueille le résultat du KeyExpander de manière synchrone grâce à la clock.

4.3.2 Simulation

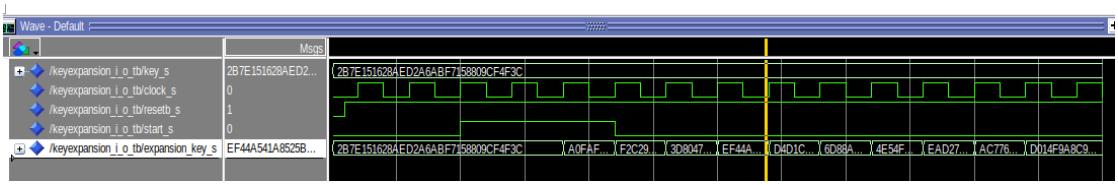

FIGURE 16 – Simulation du KeyExpansion_I_O

L'entité KeyExpansion_I_O a bien le comportement désiré, les clefs des différents cycles sont correctement construites.

5 AES

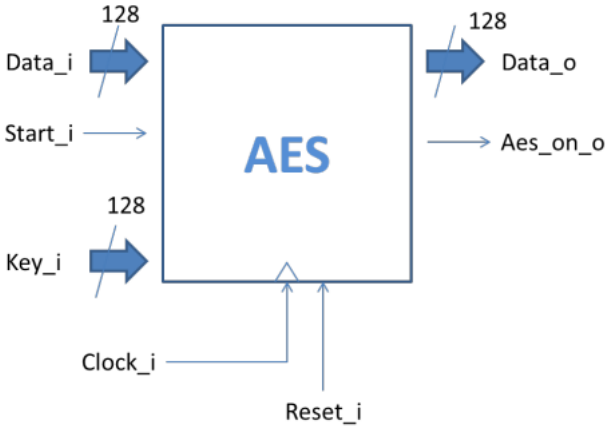

FIGURE 17 – Entrées/sorties de l'AES

L'entité AES prend en entrée une clef de 128 bits ainsi qu'un message de 128 bits et retourne le texte chiffré sur 128 bits. Elle coordonne les entités KeyExpansion_I_O et AESRound, à l'aide de la machine d'état principaux FSM_AES et du Counter. Ces deux dernières entités ne sont pas détaillées ici car founies par le sujet.

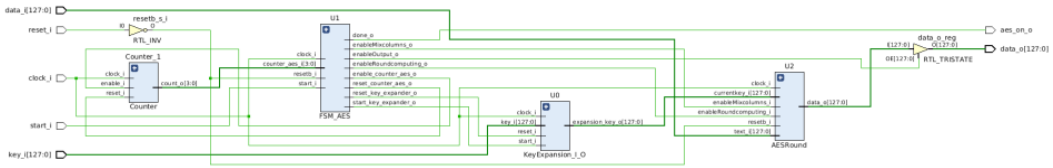

FIGURE 18 – Synoptique de l'AES

5.1 Configuration d'AES

Le fichier *aes_conf.vhd* permet d'assurer la cohérence entre les différentes entités contenues dans les différents fichiers *.vhd*, et notamment la cohérence entre les fichiers fournis par le sujet et les fichiers que j'ai rédigé.

6 Test et évaluation de l'AES

Pour évaluer l'AES, on utilise le message : "Es-tu confinèe ?" en hexadécimal ("45 73 2d 74 75 20 63 6f 6e 66 69 6e e8 65 20 3f"), et la clef : "cd 30 1f 3e 1f 96 9b 1e 6d 37 d1 79 80 7b 55 b4" en hexadécimal. Les données fournies dans le sujet permettent de vérifier le bon fonctionnement de l'AES-128 modélisé.

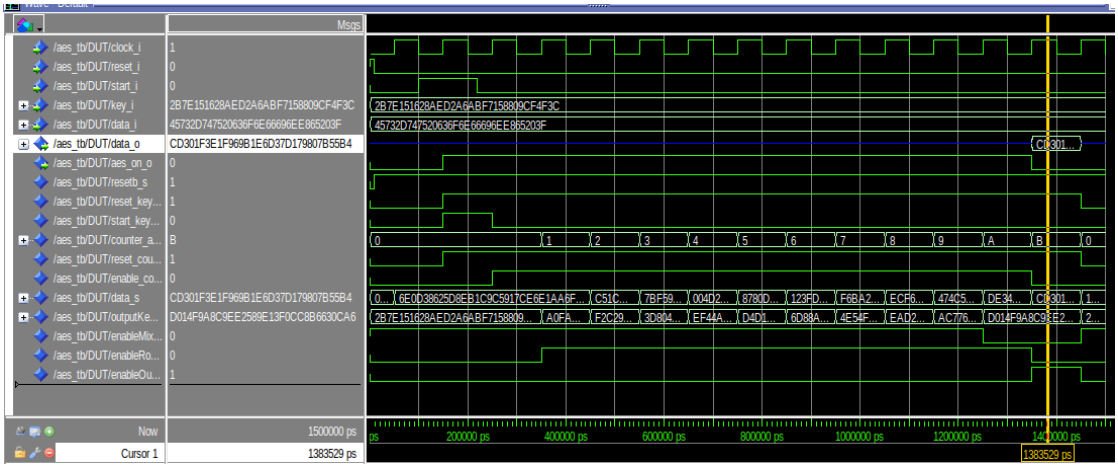

FIGURE 19 – Simulation de l'AES

L'entité AES a bien le comportement désiré : le message a été correctement chiffré.

6.1 Déchiffrement

Malgré qu'AES soit un algorithme de chiffrement symétrique, le système conçu ici ne permet pas de déchiffrer un message chiffré par AES-128. En effet, la clef utilisée pour déchiffrer le message est la même que celle utilisée pour le chiffrer, mais le déchiffrement consiste à appliquer les opérations inverses, dans l'ordre inverse et avec les clefs intermédiaires utilisées dans l'ordre inverse. Déchiffrer nécessiterait donc d'autres entités capables de faire les opérations inverses de mixColumns, shiftRows et subBytes [3].

7 Ouverture

Le chiffrement AES est actuellement considéré comme sûr. Cependant, le développement des attaques par canaux auxiliaires ces dernières années montre qu'une implémentation d'AES peut parfois être cassé grâce à des attaques par analyse de consommation [4], ou encore des attaque par faute [5][6]. L'implémentation d'AES dans ce projet est très certainement sensible à ce genre d'attaques, étant donné que ces possibles failles n'ont pas du tout été prises en compte lors de la conception.

8 Difficultées rencontrées lors du développement

Ce projet m'a permis de me familiariser avec un langage de description matériel, ici le VHDL. La logique de développement m'a demandé un effort d'adaptation au vu des différences avec les logiques de développement habituels aux langages informatique comme le C.

La manipulation du format des données qui change selon le contexte en bit128, en matrice, en ligne, ou en colonne m'a donné du fil à retordre. Cependant, j'ai trouvé le débuggage très efficace grâce aux outils de visualisation fournis par SystemVision.

9 Annexes

Vous trouverez en annexe de ce rapport tout les codes sources utiles pour faire fonctionner ce projet, ainsi qu'un certain nombre de bancs d'essais permettant de tester les entités qui composent le projet individuellement (certains dont il n'est pas fait mention autre part qu'ici dans le présent rapport).

Liste des codes fournis par le sujet (dans le dossier SRC) :
- $THIRDPARTY/CryptPack.vhd$
- $RTL/AES.vhd$
- $RTL/FSM\_AES.vhd$
- $RTL/Counter.vhd$

Liste des codes des entités (dans le dossier SRC/RTL) :
- $addRoundKey.vhd$
- $AESRound.vhd$
- $FSM\_KeyExpansion.vhd$
- $keyExpander.vhd$
- $KeyExpansion\_I\_O.vhd$
- $mixColumns.vhd$
- $mixMul.vhd$
- $SBox.vhd$
- $shiftRows.vhd$
- $subBytes.vhd$

Liste des codes des bancs d'essais (dans le dossiers SRC/BENCH) :
- $addRoundKey\_tb.vhd$
- $AESRound\_tb.vhd$
- $AES\_tb.vhd$
- $FSM\_KeyExpansion\_tb.vhd$
- $keyExpander\_tb.vhd$
- $KeyExpansion\_tb.vhd$
- $mixColumns\_tb.vhd$
- $mixMul\_tb.vhd$
- $SBox\_tb.vhd$
- $shiftRows\_tb.vhd$
- $subBytes\_tb.vhd$

Configuration de l'AES (dans le dossier SRC/RTL) : $aes\_conf.vhd$.
Script de compilation : $compile\_src.sh$.

Références

[1] Contributeurs de Wikipédia, *Advanced Encryption Standard*, Wikipédia, l'encyclopédie libre, (Page consultée le octobre 16, 2020).

[2] Olivier POTIN, *Projet Conception d'un Système Numérique - Modélisation VHDL de l'algorithme de chiffrement AES*, 2020.

[3] Saïda BEKHOUCHE, *Fondements mathématiques et fonctionnement du standard de chiffrement avance (AES)*, 2012. Thèse de doctorat.

[4] Loıc DUFLOT, *Rapport de projet : Differential Power Analysis*, 2003

[5] Yifan LU, *Attacking Hardware AES with DFA*, 2019

[6] Thomas FUHR, Eliane JAULMES, Victor LOMNE and Adrian THILLARD, *Fault Attacks on AES with Faulty Ciphertexts Only*, 2013