Conception de Système Numérique

Modélisation VHDL de l'algorithme de déchiffrement AES

Table des matières

Table des figures

Liste des tableaux

1 Introduction - Chiffrement/Déchiffrement par bloc

Bob et Alice souhaitent garder secrète leur conversation vis-à-vis d'Eve jalouse. Ils décident donc de s'envoyer des messages dont ils seront les seuls à comprendre (cf. exemple de Table 1). Dans ce but, ils se basent sur la théorie de la cryptographie pour chiffrer/déchiffrer ces messages. Parmi les solutions existantes, ils choisissent un algorithme de chiffrement symétrique par bloc réputé mathématiquement inviolable : AES [1]. Dans le cas de l'AES, la taille des blocs de données est fixée à 16 octets soit 128 bits. La clé peut être néanmoins de taille 128 bits, 192 bits ou 256 bits.
L'objectif de ce projet est de développer en VHDL l'algorithme de déchiffrement AES avec une clé de 128 bits.

Clé	`2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c`
Message à déchiffrer	As-tu la COVID ?
Message à déchiffrer (hexadécimal)	`41 73 2d 74 75 20 6c 61 20 43 4f 56 49 44 20 3f`
Message chiffré (hexadécimal)	`8c 11 35 44 06 ad 44 88 de ca ec 83 aa 03 43 06`

Table 1 – Message envoyé par Bob et reçu chiffré par Alice

2 Description de l'algorithme AES 128bits

L'algorithme AES se compose de 4 transformations élémentaires : AddRoundKey, SubBytes, ShiftRows et MixColumns détaillées dans la figure 1 pour le chiffrement ; AddRoundKey, InvSubBytes, InvShiftRows et InvMixColumns pour le déchiffrement. L'algorithme se déroule en 11 rondes : 1 ronde initiale suivie de 9 rondes exécutant les 4 fonctions et 1 dernière ronde ne comprenant pas l'appel à la fonction *InvMixColumns*. Le déchiffrement s'exécute dans l'ordre inverse de l'algorithme de chiffrement.

Un bloc de données sur 16 octets (i.e. 128 bits) forme le message à chiffrer/déchiffrer en entrée de l'algorithme. Une clé ***secrète*** utilisée lors de la ronde initiale et dérivée ensuite dans les rondes suivantes constitue un paramètre de l'algorithme AES. Une fonction supplémentaire est nécessaire pour la génération des clés de ronde. Cette fonction est appelée *Key Expansion*.

Description des entrées / sorties
Dans le cadre de ce projet, vous développerez l'entité InvAES décrite dans la figure 2 pour déchiffrer les messages envoyés entre Bob et Alice selon l'algorithme de la figure 3.

L'entité comporte les entrées / sorties suivantes :
- Une entrée 'data_i' de 128 bits
- Une horloge 'clock_i'
- Un signal d'initialisation 'reset_i'
- Un signal 'start_i' indiquant la présence d'un message à chiffrer en entrée
- Une sortie 'data_o' sur 128 bits
- Un signal 'aes_done_o' indiquant que les calculs de déchiffrement sont finis

Chronogramme du fonctionnement de l'entité InvAES
Le chronogramme de l'entité est donné dans la figure 4. Il n'inclut pas les contraintes de *timing* de l'horloge comme des relations entre signaux de commande et de donnée présents sur les bus. Les signaux évolueront de manière synchrone avec l'horloge.

1. Advanced Encryption Standard

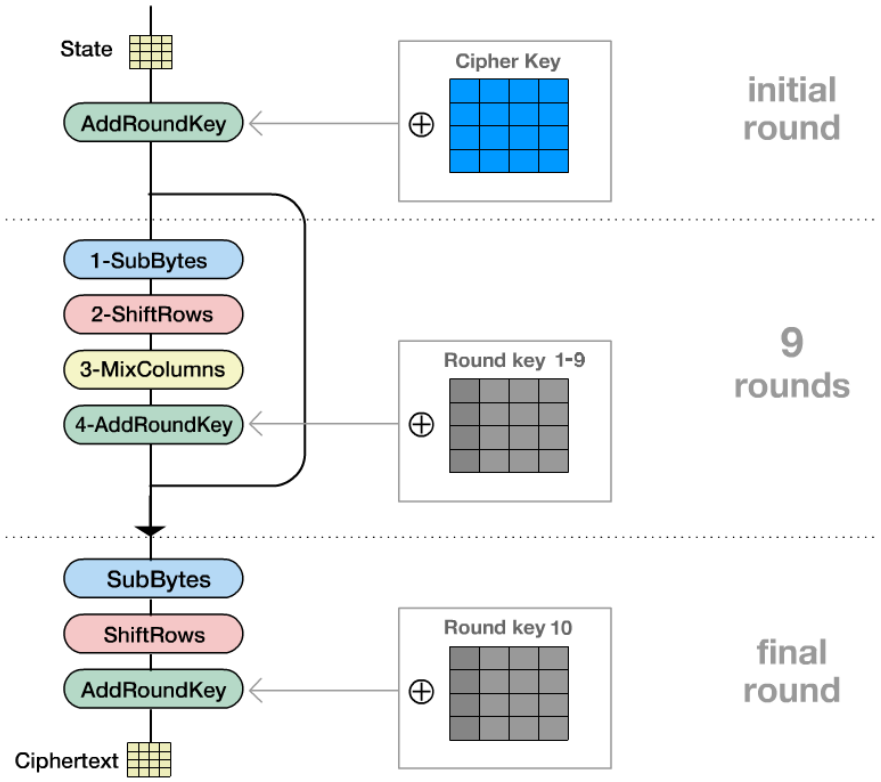

Figure 1 – Chiffrement AES 128bits

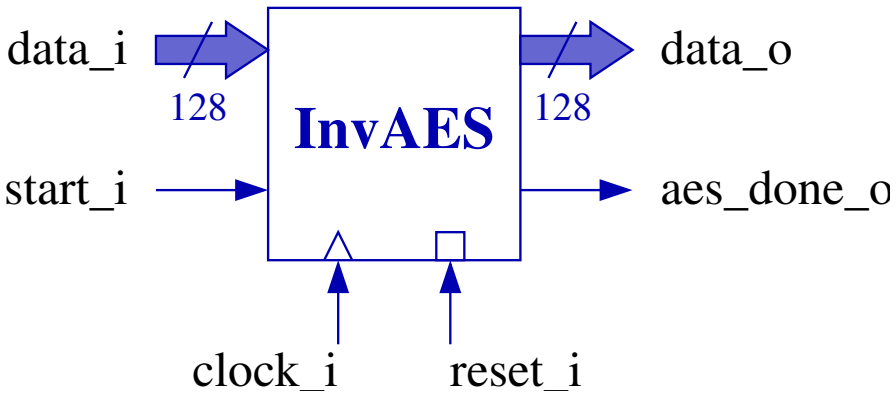

Figure 2 – Entrées / Sorties de l'entité InvAES

```
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 3 – Algorithme de déchiffrement

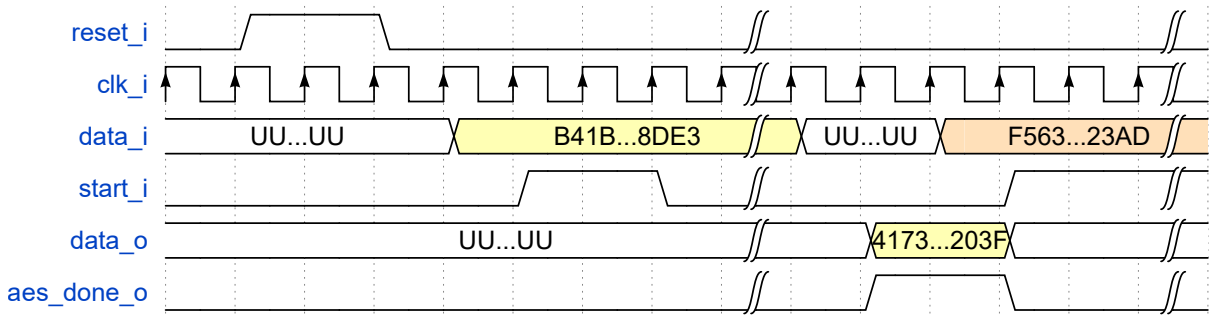

Figure 4 – Évolution des signaux de l'entité InvAES

2.1 Formalisme et notation

Pour être en adéquation avec la norme, vous utiliserez les notations suivantes :
— La numérotation des indices de la figure 5
— La représentation des octets (cf. figure 6)
— Un état peut également être considéré comme un tableau de colonnes, chaque colonne comprenant 4 octets soit un mot (word) de 32 bits (cf. figure 7)

Input bit sequence	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	...
Byte number	0								1								2								...
Bit numbers in byte	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	...

Figure 5 – Numérotation des vecteurs de bits

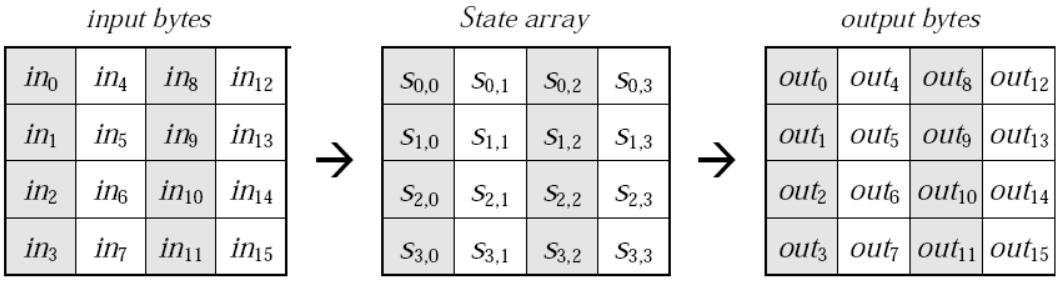

Figure 6 – Représentation de la matrice d'état de l'AES

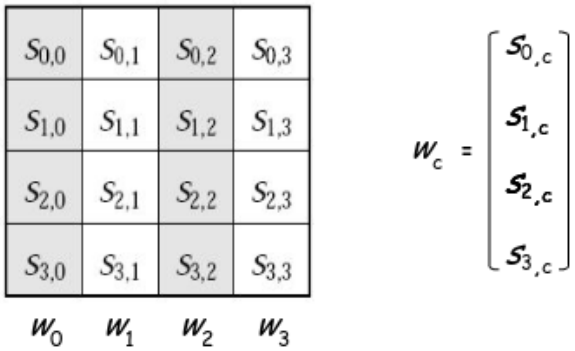

Figure 7 – Représentation d'une colonne de la matrice d'état

2.2 La transformation InvAddRoundKey

la fonction InvAddRoundKey ajoute la clef de ronde courante (Ronde 10 à Ronde 0) à l'état ; l'addition étant prise au sens ou-exclusif, elle est équivalente à la fonction AddRoundKey. Par conséquent, la fonction booléenne XOR est appliquée bit à bit entre les octets de l'état et les octets de la clef de ronde. La figure 8 montre l'application de la fonction. A noter que la clef de la ronde 0 est la clef de chiffrement alors que les clefs des rondes suivantes sont issues d'un processus de diversification des clefs dénommé 'Key Expansion'. Pour le projet, l'ensemble des clefs sera disponible à partir d'une mémoire adressable de $0x0$ à $0xA$ et compilé dans la librairie ***LIB_AES*** fournie.

2.3 La transformation InvShiftRows

La fonction InvShiftRows effectue une permutation cyclique (i.e. rotation) des octets des lignes de l'état. Le décalage des octets correspond à l'indice de la ligne considérée ($0 \leq r < 4$). Ainsi,

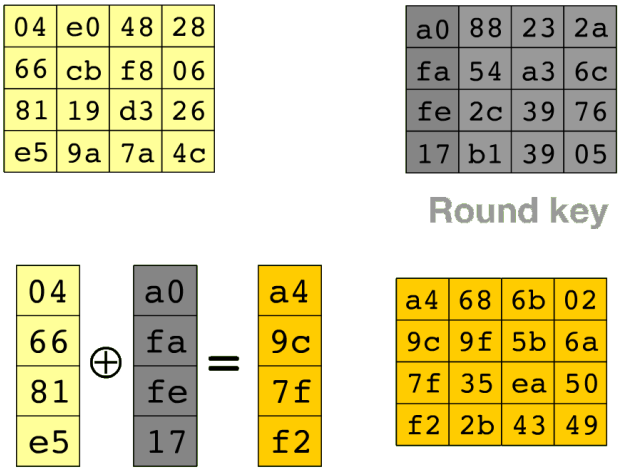

Figure 8 – Calcul de la fonction InvAddRoundKey

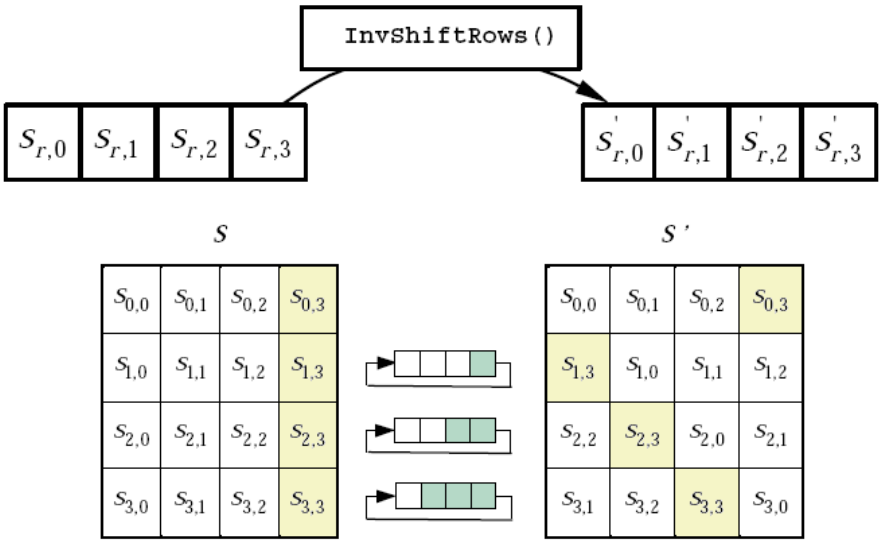

Figure 9 – Illustration de la fonction InvShiftRows

dans l'hypothèse ou $S_{1,x} = \{0x23, 0xeb, 0x5f, 0x99\}$, l'application de la fonction donnera $S'_{1,x} = \{0x99, 0x23, 0xeb, 0x5f\}$. Une illustration de la fonctionnalité est détaillée dans la Figure 9.

2.4 La transformation InvSubBytes

La fonction InvSubBytes consiste en une transformation non linéaire appliquée à tous les octets de l'état en utilisant une table de substitution appelée S-Box (cf. Figure 10). Vous implanterez la S-Box de la figure 11 sous la forme d'une mémoire (i.e. un tableau en VHDL) lors du développement de ce projet.
Pour exemple, soit $S_{2,1} = \{0x53\}$, l'application de la fonction InvSubBytes donnera l'écriture de $S'_{2,1} = \{0xED\}$ en sortie.

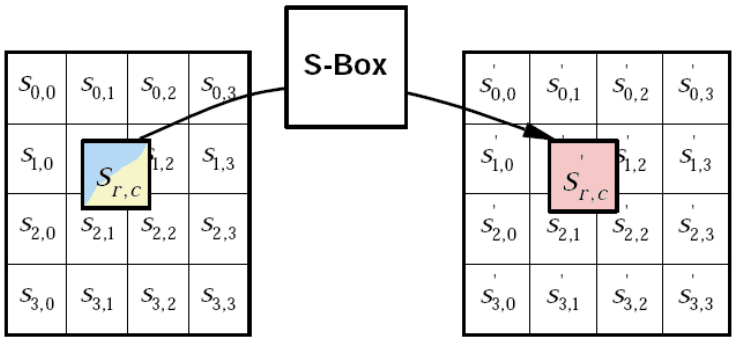

FIGURE 10 – Principe de la fonction InvSubBytes

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 11 – Table de substitution utilisé pour le projet

2.5 La transformation InvMixColumns

La fonction InvMixColumns applique une transformation colonne après colonne à un état. Cette transformation linéaire d'un produit matriciel utilisant les 4 octets d'une colonne (cf. Figure 12). Les colonnes sont traitées comme des polynômes dans GF(2^8) [2] et multipliées modulo x^4+1 (noté $\otimes$) avec les polynômes fixes de la Figure 12. Ces opérations réalisées dans le champs de Gallois sont calculées pour les additions à l'aide d'une fonction XOR.

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

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

FIGURE 12 – Produit matriciel de la fonction InvMixColumns

Illustrons la multiplication sur 2 exemples :

— Soit $S_{0,c} = 0xD4 = b'11010100$, $\{02\} \otimes S_{0,c}$ s'écrit sous la forme polynomiale :

$$(x^7 + x^6 + x^4 + x^2) \cdot x = x^8 + x^7 + x^5 + x^3$$

2. Galois Field ou champs de Galois de 2^8 = corps commutatif fini de cardinal 256

A la lecture de *Cryptography and Network Security* [1], la multiplication polynômiale par x peut être implanté à l'aide d'un décalage à gauche suivi d'un ou-exclusif avec la valeur b'100011011 conditionné par le bit de poids fort du polynôme. Ainsi nous écrivons l'opération telle que :

$$\begin{aligned} \{02\} \otimes S_{0,c} &= (\{02\} \odot S_{0,c}) \ mod \ b'100011011 && (1)\\ &= shiftleft(b'11010100) \ xor \ b'100011011 \ (0xD4_7 = 1 \Rightarrow ou-exclusif) && (2)\\ &= b'110101000 \ xor \ b'100011011 && (3)\\ &= b'10110011 = 0xB3 && (4) \end{aligned}$$

— Soit $S_{0,c} = 0x04 = b'00000100$, $\{02\} \otimes S_{0,c}$ s'écrit sous la forme polynomiale :
$$(x^2) \cdot x = x^3$$
idem, nous écrivons l'opération telle que :

$$\begin{aligned} \{02\} \otimes S_{0,c} &= (\{02\} \cdot S_{0,c}) \ mod \ b'100011011 && (5)\\ &= shiftleft(b'00000100) && (6)\\ &= b'00001000 = 0x08 && (7) \end{aligned}$$

Soit $W_2 = \{0x14, 0xE5, 0xFF, 0x00\}$, la sortie de la fonction InvMixColumns pour $S'_{0,2}$ sera :

$$\begin{aligned} S'_{0,2} &= (\{0xE\} \otimes S_{0,2}) + (\{0xB\} \otimes S_{1,2}) + (\{0xD\} \otimes S_{2,2}) + (\{0x9\} \otimes S_{3,2}) && (8)\\ &= (\{b'1110\} \otimes S_{0,2}) \ xor \ (\{b'1011\} \otimes S_{1,2}) \ xor \ (\{b'1101\} \otimes S_{2,2}) \ xor \ (\{b'1001\} \otimes S_{3,2}) && (9) \end{aligned}$$

Or

$$\begin{aligned} (\{b'1001\} \otimes S_{3,2}) &= (\{b'1000 \ xor \ b'0001\} \otimes S_{3,2}) && (10)\\ &= (\{08\} \otimes S_{3,2} \ xor \ \{01\} \otimes S_{3,2}) && (11)\\ &= (\{2^3\} \otimes S_{3,2} \ xor \ S_{3,2}) && (12)\\ &= (\{02\} \otimes (\{02\} \otimes (\{02\} \otimes S_{3,2}))) \ xor \ S_{3,2}) && (13)\\ &= ..... && (14) \end{aligned}$$

Par conséquent, le calcul successif des puissances de 2 des octets contenus dans la colonne permet d'effectuer toutes les opérations pour le calcul de la matrice inverse de la fonction MixColumns.
Avec $\oplus$ (i.e. xor) la fonction booléenne ou-exclusif définit selon :

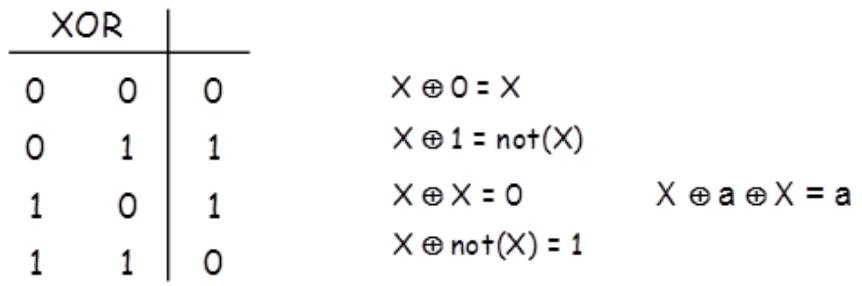

Figure 13 – Fonction Ou-exclusif

Notons également qu'une autre solution peut être implantée sous la forme de tables de substitution pré-calculées pour les multiplications par 0x9, 0xB, 0xD et 0xE.

2.6 Architecture globale de InvAES

L'architecture complète de l'InvAES est décrite dans la figure 14.
Elle contient :

— une machine d'états (ou Finite State Machine (FSM)) pilotant les signaux de contrôle de l'entité InvAES

— l'entité *KeyExpansion_ table* contenant toutes les clefs de rondes issues du standard NIST [2]
— un compteur *Counter* piloté par la FSM pour fixer l'adresse de la clef de ronde courante
— un composant *InvAESRound* intégrant les fonctions (AddRoundKey, InvShiftRows, InvSubBytes et InvMixColumns) pour le calcul d'une ronde
— un registre pour la sauvegarde du résultat du déchiffrement
— un multiplexeur pour choisir entre le message à déchiffrer en début de l'algorithme et le résultat intermédiaire issu du calcul d'une ronde

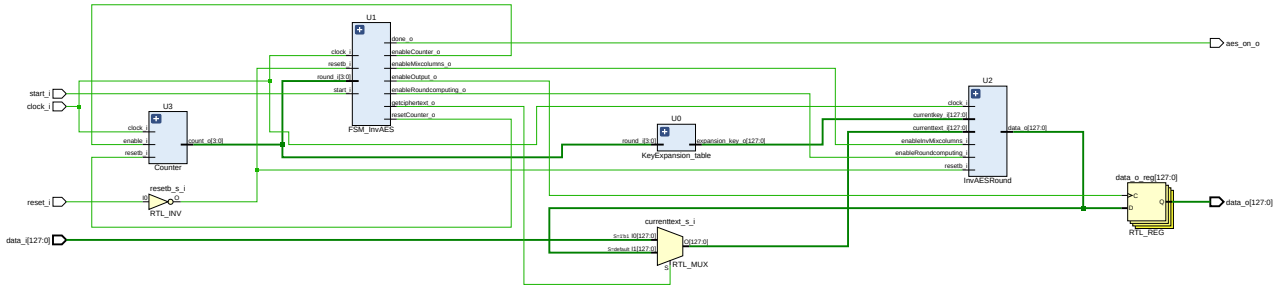

Figure 14 – Architecture globale de l'AES

3 Organisation du travail

3.1 Planning

Vous disposez de 5 séances pour modéliser l'entité InvAES en VHDL. Chacune des séances sera dédiée à la réalisation d'une composante de l'entité et sa validation. Il est par conséquent important de terminer cette composante avant le début de la séance suivante. Le découpage des séances est proposé comme suit :

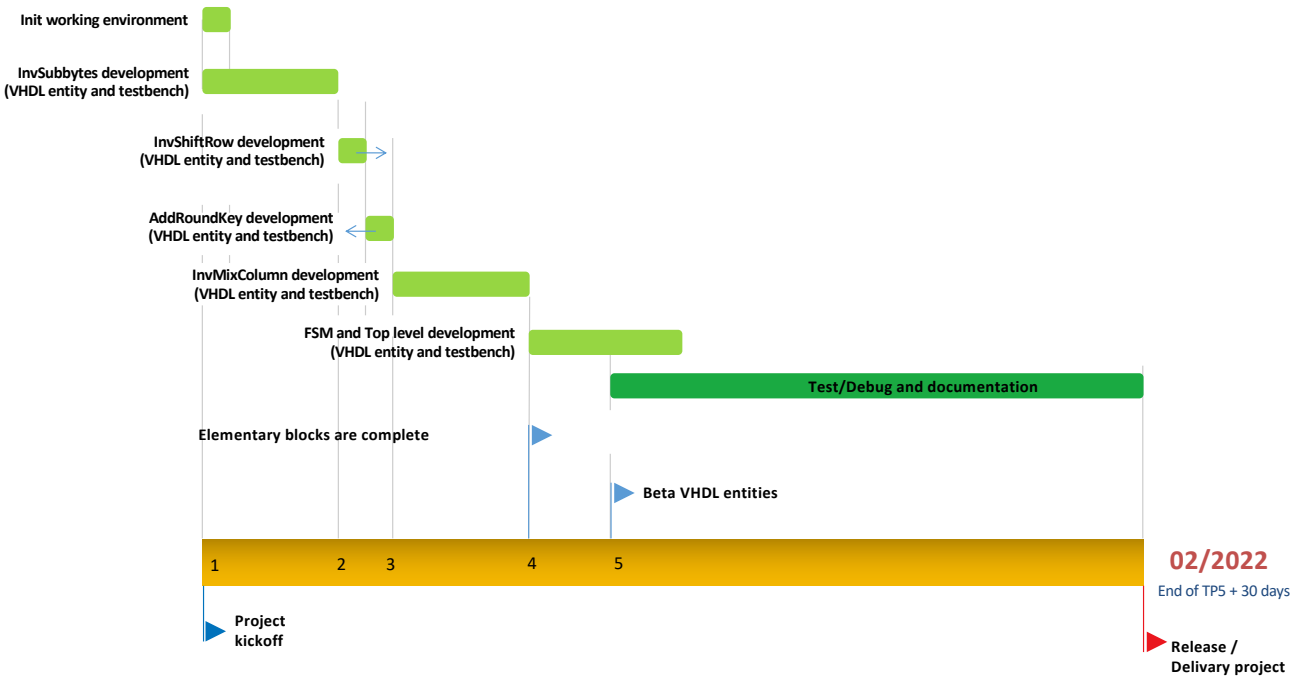

Figure 15 – Représentation Gantt du projet

1. Lecture et compréhension du sujet / Conception et test de la S-BOX pour la fonction InvSubBytes
2. Conception et test des modules *InvShiftRows* et *AddRoundKey*
3. Conception et test de la fonction *InvMixColumns*

4. Conception et test de la machine d'états (FSM) régissant le déchiffrement et du compteur
5. Suite de la conception de la FSM et intégration de tous les éléments dans l'entité InvAES (Top level) pour validation complète

Afin de coller à la réalité d'un projet industriel, le planning de la figure 15 décrit l'évolution que doit suivre votre travail. Ainsi un bilan d'avancement du projet en pourcentage sera demandé pour chacune des séances.

3.2 Contraintes de développement

Le projet sera développé à l'aide de l'outil Mentor Graphics ModelSim. Les fichiers de description VHDL (xxx.vhd) et de test (xxx_tb.vhd) seront enregistrés dans les répertoires nommés "SRC/RTL" et "SRC/BENCH" respectivement. Les bibliothèques associées seront nommées "LIB_RTL" et "LIB_BENCH" et créées dans un répertoire nommé "LIB" (donc respectivement LIB/LIB_RTL et LIB/LIB_BENCH). Un script automatisant la compilation vous sera demandé.

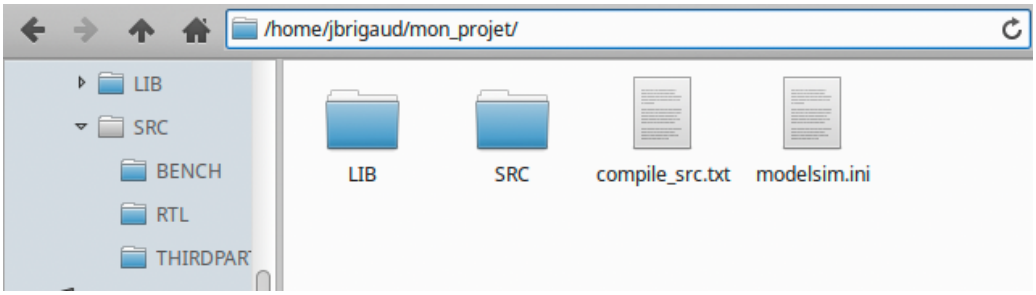

Figure 16 – Organisation des répertoires de travail

De plus, et afin de disposer d'un code VHDL lisible, une attention toute particulière sera apportée à l'écriture du code. Ainsi certaines règles devront être respectées :

1. Une architecture associée à une entité doit se trouver dans le même fichier VHDL. Le nom du fichier correspond au nom de l'entité.
2. Le nom de l'architecture reprend le nom de l'entité complété du suffixe "_arch".
3. Pour les machines d'états précisez si le modèle d'architecture est Mealy ou Moore. Exemple : "FSM_arch_Moore".
4. Les signaux entrant d'une entité doivent être complétés par le suffixe "_i"
5. Les signaux sortant d'une entité doivent être complétés par le suffixe "_o"
6. Les signaux internes d'une architecture doivent être complétés par le suffixe "_s"
7. Les commentaires sont "obligatoires"

3.3 Validation du fonctionnement

A titre d'évaluation du bon fonctionnement de votre développement, les états intermédiaires après chaque étape de calcul de l'AES pour le déchiffrement du message avec la clé `2b 7e 15 16 28 ae d2 a6 ab f7 15` vous sont listés ci-dessous.

```
Cipher text to decipher: (Hex) 8c 11 35 44 06 ad 44 88 de ca ec 83 aa 03 43 06

SetCiphertext : 8c 11 35 44 06 ad 44 88 de ca ec 83 aa 03 43 06
InitKey : 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c
KeyExpansion (round 0): 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c
KeyExpansion (round 1): 75 ec 78 56 5d 42 aa f0 f6 b5 bf 78 ff 7a f0 44
KeyExpansion (round 2): ca fb fe 2b 97 b9 54 db 61 0c eb a3 9e 76 1b e7
KeyExpansion (round 3): c1 bf 4e f4 56 06 1a 2f 37 0a f1 8c a9 7c ea 6b
```

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

```
Round 0
InvShiftRows : 58 f3 76 ab 8d e6 5a 31 ce c6 46 9c 72 ce 06 d5
InvSubBytes : 6a 0d 38 62 5d 8e be c7 8b b4 5a de 40 8b 6f 03
AddRoundKey : 41 73 2d 74 75 20 6c 61 20 43 4f 56 49 44 20 3f
Getplaintext : 41 73 2d 74 75 20 6c 61 20 43 4f 56 49 44 20 3f
Obtained plain text : (Hex) 41 73 2d 74 75 20 6c 61 20 43 4f 56 49 44 20 3f
 (ASCII) As-tu la COVID ?
```

La simulation de InvAES pour le déchiffrement du message de Bob est donnée dans la figure 17 ci-dessous.

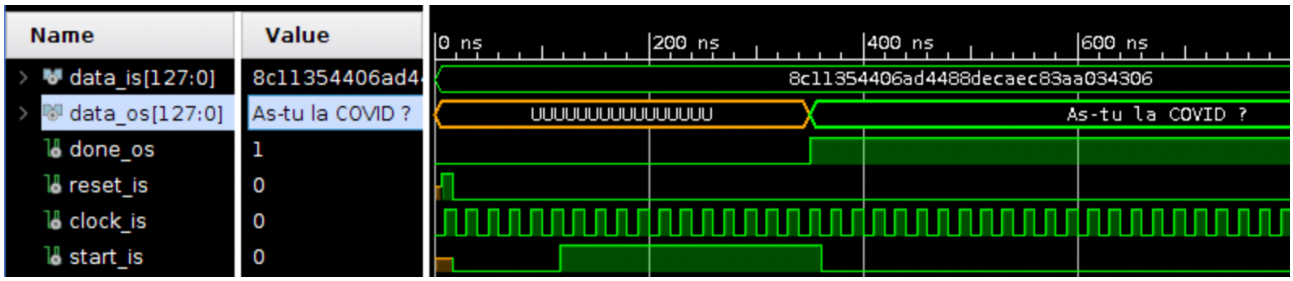

Figure 17 – Simulation d'un déchiffrement

3.4 Livrables

La note sera composée du PV de réception du projet complet. Le PV de réception sera validé après l'envoi par l'élève d'un fichier compressé (.zip ou .tar) à la date prévue par le Gantt. Il sera constitué :

1. Un rapport au format pdf contenant la description de votre entité InvAES et le chronogramme résultant de sa simulation
2. La description de la machine d'états gérant le déchiffrement
3. Les chronogrammes (et leur interprétation) issus de la simulation des éléments constituant InvAES
4. La description des points bloquants et les solutions envisagées
5. Les codes VHDL répartis dans les répertoires "sources" et "bench"
6. Le script de compilation de ces codes VHDL

La notation sera réalisée selon le barème suivant :

Modules						
InvSubBytes	InvShiftRows	AddRoundKey	InvMixColumns	InvAES FSM	Top	
1 pt	1 pt	1 pt	3 pts	3 pts	2 pts	
Règles d'écriture sources VHDL	Consignes VHDL		Rapport			Note / 30
	Script	Organisation	Contenu	Orthographe		
5 pts	1 pt	3 pts	7 pts	3 pts		

Table 2 – Notation du projet PCSN

Note 1 : Toute ressemblance avec un projet existant ou copié sur un autre élève sera sanctionnée d'un 0/20

Note 2 : Il n'y a pas de rattrapage pour cette UP

Références

[1] W. Stallings, *Cryptography and Network Security : Principles and Practice*. Upper Saddle River, NJ, USA : Prentice Hall Press, 5th ed., 2010.

[2] NIST, "Fips-197, announcing the advanced encryption standard (aes),"