Decipher VHDL AES

Ecole des Mines de Saint-Etienne

Martin Guyard – 07/01/2019

Sommaire

Abstract

Ce rapport finalise mon projet Decipher VHDL AES. L'objectif de ce projet est de décrire une architecture matérielle, grâce au VHDL, qui implémente un déchiffrement AES. Ce rapport détail mes choix de conceptions, mes difficultés, mes résultats. Les simulations sont effectuées sous Mentor Graphics ModelSim.

AES, Advanced Encryption Standard, est un standard de chiffrement symétrique paru en 2000 à la suite d'un concours lancé par la NSA dans le but de remplacer un ancien standard vieillissant et plus assez sécurisé, le DES (Data Encryption Standard). AES est actuellement un des standards les plus sûrs. Il est très populaire et il n'existe pas à ce jour de cryptanalyse efficace pour le casser.

AES est un algorithme de chiffrement par blocs de 128 bits (16 octets). La version décrite dans ce rapport est un déchiffrement AES 128 bits (i.e., la taille des clés est de 128 bits).

Algorigramme

AES est un algorithme de chiffrement par ronde, c'est-à-dire que l'algorithme effectue plusieurs fois les mêmes fonctions (rondes) en calculant une nouvelle clé de ronde à partir de l'ancienne. On distingue les fonctions fondamentales suivantes :

InvSubByte, InvShiftRow, InvMixColumn, AddRoundKey (RoundKeyExpansion).

A savoir que la version décrite dans ce rapport ne calcule pas les clés de ronde (RoundKeyExpansion n'est pas décrite). J'utilise des clés de rondes déjà calculées pour une clé secrète donnée. Le but du rapport est de fournir une preuve de concept et de travail, pas de fournir une architecture prête pour l'industrialisation.

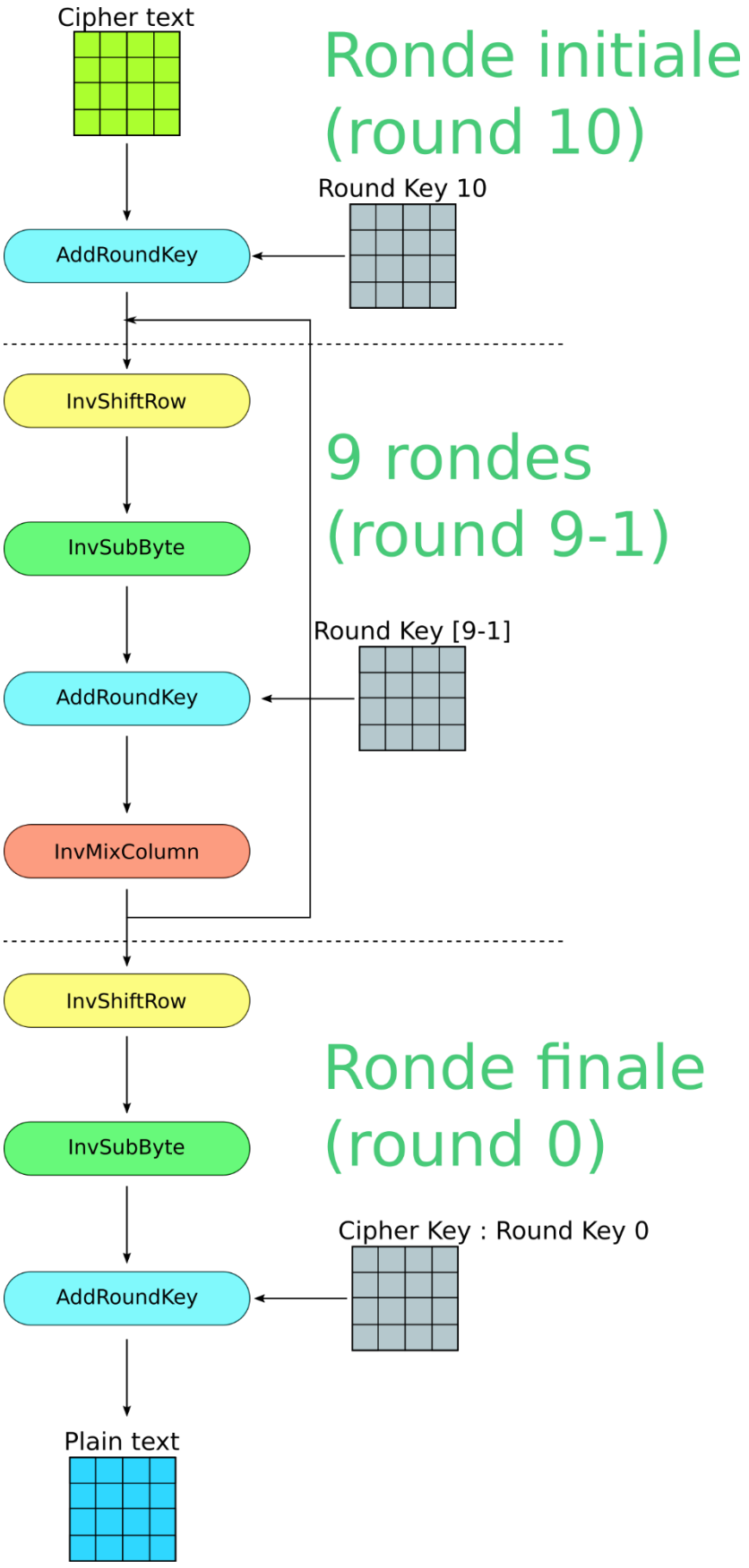


Fig1. Algorigramme AES Decipher

Routage & architecture InvAES

La vision haut niveau du projet nous donne la vue ci-dessous. Data_in et data_out sont un type bit128 défini dans un package standard donné pour le projet. Data_in représente le message chiffré, data_out le texte clair. Il n'y a pas d'entrée pour la clé secrète, elle est déjà intégrée dans la « boite noire InvAES », avec toutes les clés de rondes associées et déjà calculées.

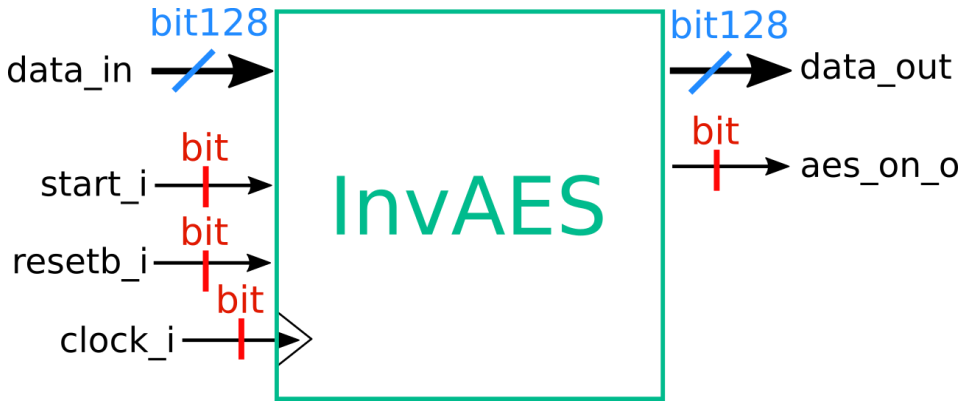


Fig2. Input / Output Inv AES

InvAES est constitué d'une **FSM Moore** pour cadencer les transitions entre les différents états de l'algorithme de déchiffrement (ronde initiale 10, ronde 9-1, ronde finale 0). D'un composant **counter** qui donne le numéro de ronde courant. D'un composant **KeyExpansionTable** qui donne la clé de ronde correspondante. Et enfin d'un composant **InvAESRound** qui représente une ronde de déchiffrement, ce composant est commandé par la FSM Moore.

Les modules/composants : **InvSubByte**, **InvShiftRow**, **InvMixColumn**, **AddRoundKey** sont tous instanciés dans InvAESRound.

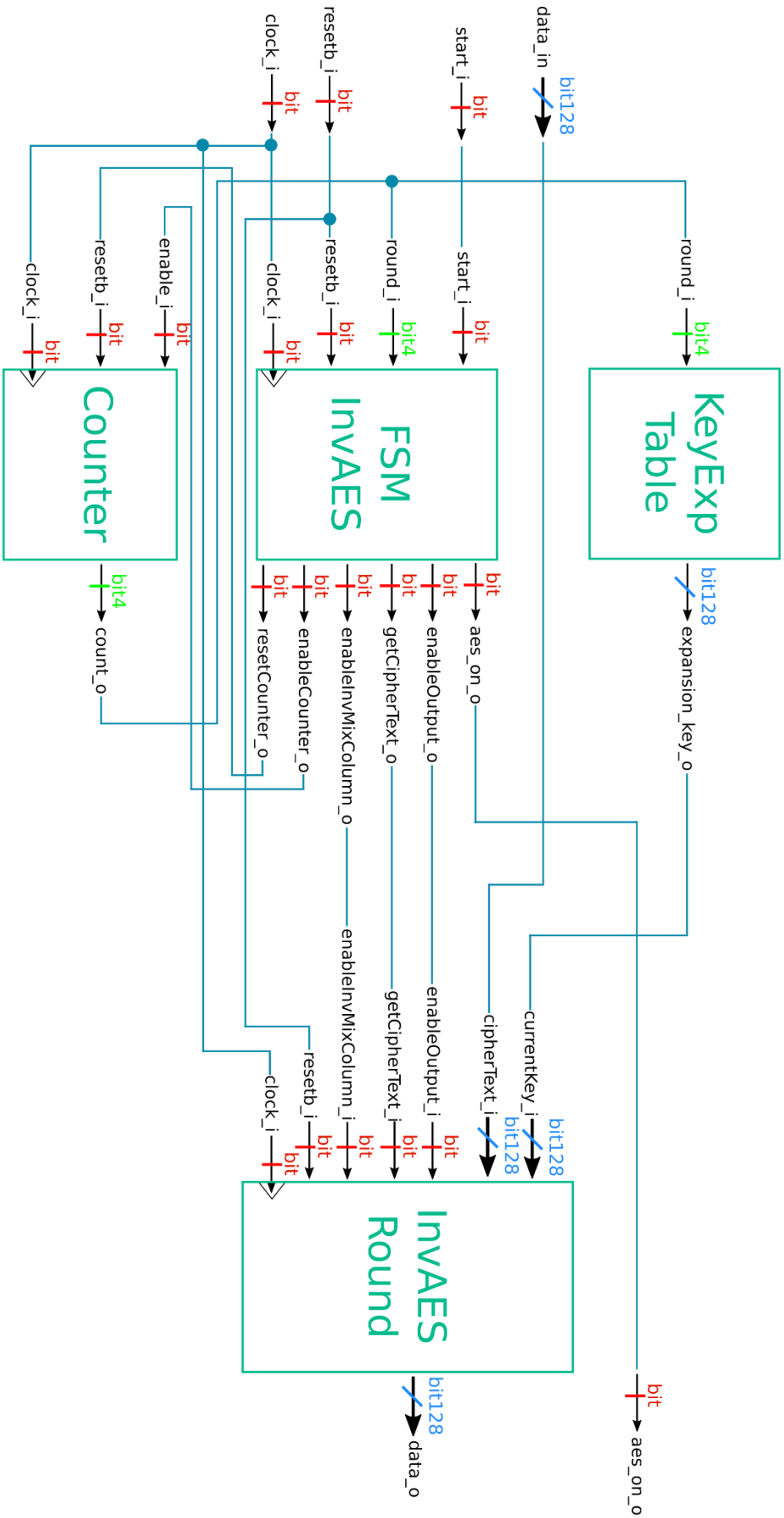


Fig3. Routage InvAES

Routage & architecture InvAESRound

InvAESRound est le composant qui déchiffre une ronde d'AES (la ronde peut être une ronde courante 9-1, une ronde initiale 10 ou une ronde finale 0)

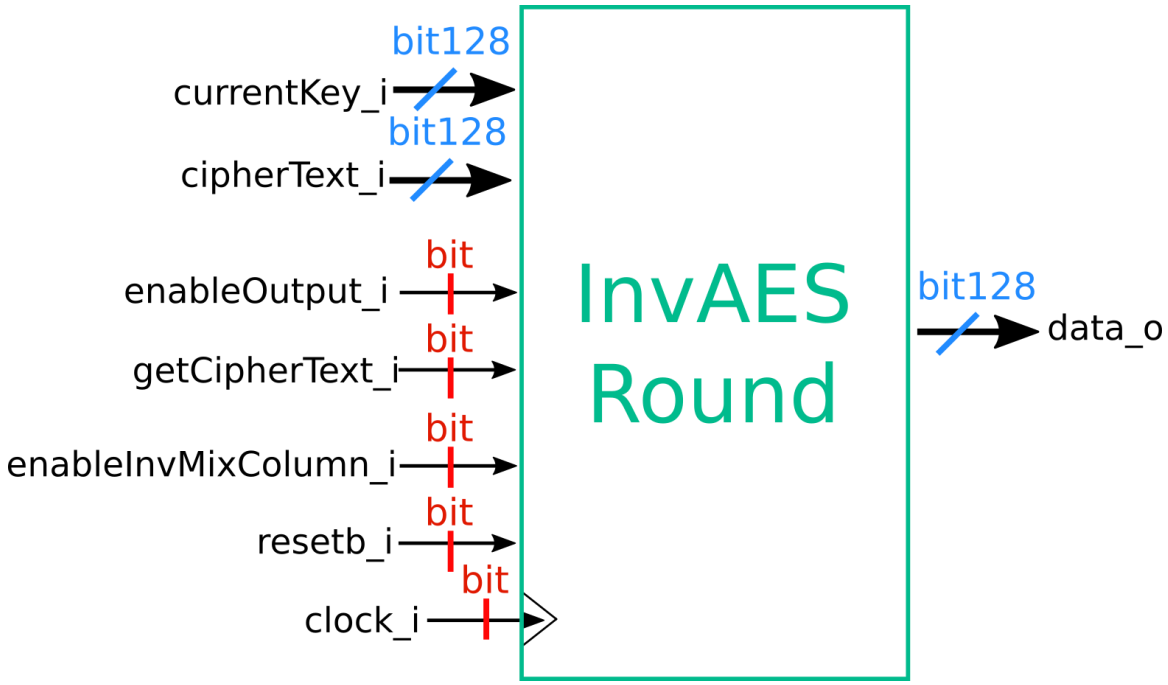


Fig4. Input / Output Inv AES Round

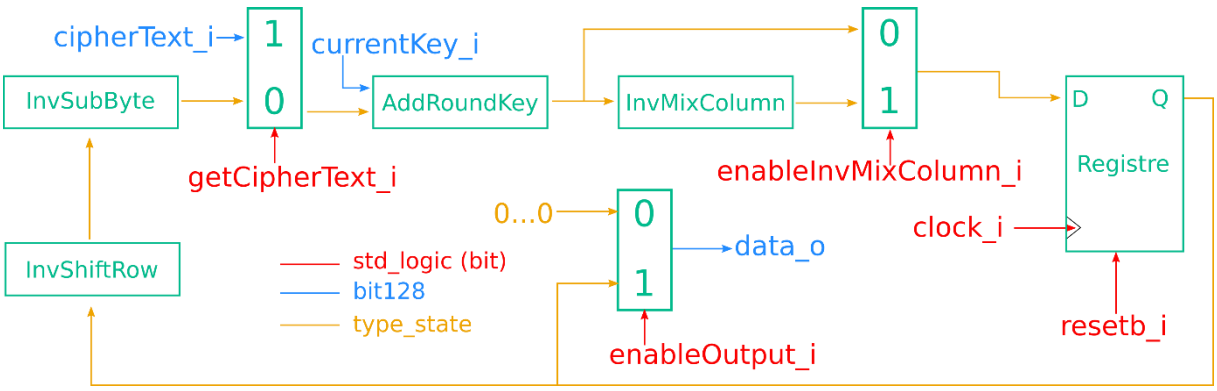


Fig5. Routage InvAESRound

Les données d'entrées/sorties sont des types bit128, elles sont converties en type_state en entrée et en sortie puisque tous nos modules « Inv » prennent des entrées et sorties type_state. data_o correspond au plain text (message déchiffré).

Tous les composants sont instanciés, peu importe l'ordre, je déclare également les signaux input/output de chaque composant, puis c'est le routage des signaux qui est conditionnel, en fonction de getCipherText_i, enableInvMixColumn_i, enableOuput_i, qui sont commandés par la FSM en fonction des états. Le routage ci-dessus correspond à mon implémentation.

Prenons un exemple : le premier multiplexeur qui est commandé par getCipherText_i. Voici comment je le décris en VHDL dans l'architecture de InvAESRound_arch :

```
addroundkey_in_s <= data_in_s when (getCipherText_i = '1') else
                    invsubbyte_out_s;
```

Le signal addroundkey_in_s prend donc bien entrée soit data_in_s (cipherText_i converti en type_state) lorsque getCipherText_i vaut '1' sinon le signal prend la sortie du module InvSubByte.

En bref, j'ai créé les signaux input/output de chaque module (InvSubByte, InvShiftRow, InvMixColumn, AddRoundKey) puis j'ai « mappé » ces signaux sur les composants correspondants, et enfin je « route » les signaux entre eux et en fonction des multiplexeurs.

Enfin l'architecture implémentée est celle que Mr. Dutertre nous a demandé d'implémenter, mais il aurait été tout-à-fait possible de proposer une autre architecture. L'avantage de celle proposée par rapport à celle présentée dans l'énoncé du projet, est qu'elle est plus compacte et simplifie le routage final de InvAES global. C'est une architecture qui me convient.

FSM Moore

La FSM Moore est instanciée dans InvAES global. Ce composant représente la machine d'état qui cadence les transitions d'états et les entrées de InvAESRound. En effet InvAESRound permet de calculer une ronde de l'inverse AES dans plusieurs cas de figure : ronde initiale, ronde courante et ronde de fin. Cependant il faut un composant qui le commande (qui commande la valeur des interrupteurs des multiplexeurs), ce composant c'est la FSM Moore. La FSM prend en entrée en plus de l'horloge et du bit de reset, un bit de start qui initie le déchiffrement et un quartet (bit4) qui indique, en format unsigned, le numéro de ronde courante.

A partir de ses entrées et des états, la FSM Moore fait évoluer ses sorties et commande le composant InvAESRound.

Dans mon cas j'ai identifié 5 états pour ma FSM :

- 1 état initial / veille,
- 3 état de déchiffrement (ronde initiale, ronde courante, ronde finale),
- 1 état final ¨affichage de la sortie.

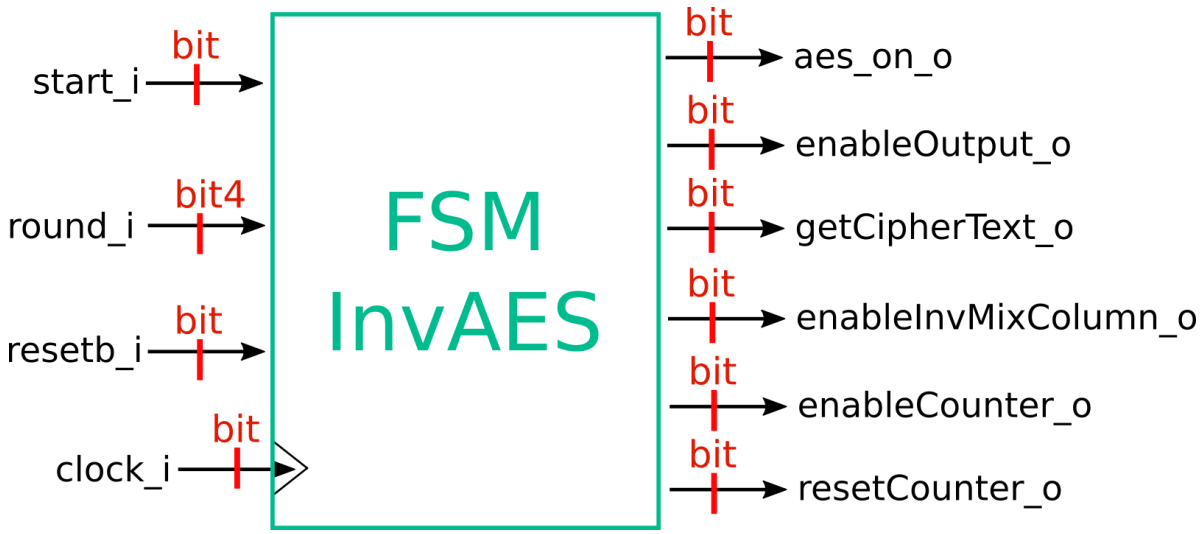


Fig6. Input / Output FSM InvAES

Dans la machine décrite ci-dessous, le reset est asynchrone, c'est-à-dire qu'il n'attend pas le prochain coup d'horloge pour que l'état courant repasse sur l'état0. Enfin les sorties de la machine de Moore sont synchrones, elles évoluent sur des états, c'est-à-dire au moment des coups d'horloges. Ainsi, si la condition de transition est respectée, l'état courant passe à l'état suivant au prochain coup d'horloge, sinon l'état courant reste inchangé.

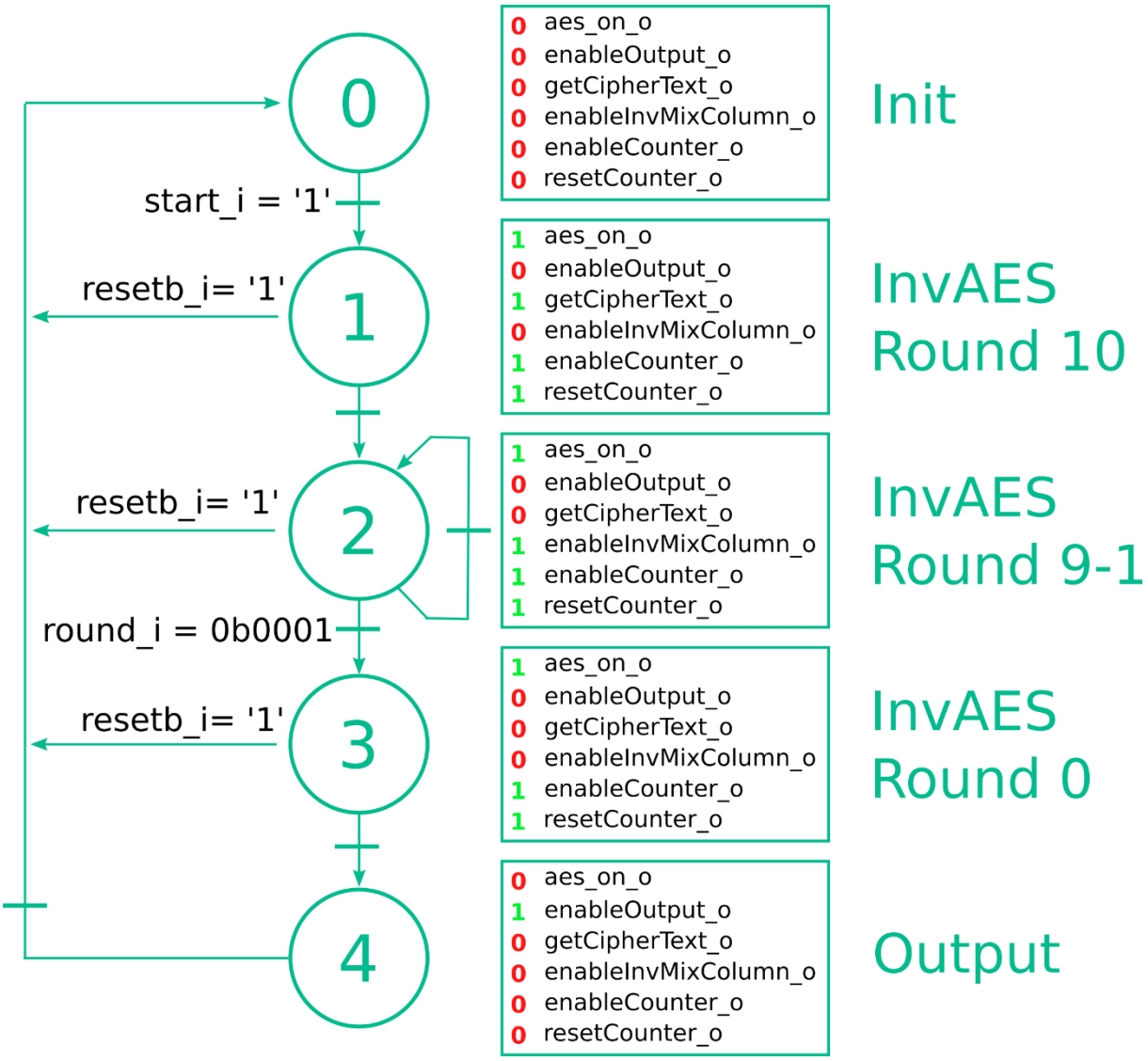


Fig7. FSM Moore InvAES

J'ai eu l'occasion de discuter du modèle de FSM avec mes camarades. Il est apparemment possible de réaliser une FSM à seulement 4 états en combinant mes état0 et état4, mais cela me semble moins précis. Ensuite j'ai préféré réaliser une machine de Moore car cela me parait plus simple à mettre en œuvre qu'une machine de Mealy.

Concernant les états, le schéma est explicite. A partir de l'algorigramme *Fig1* j'explicite les évolutions des sorties et je regroupe en états des évolutions de sorties similaires. C'est seulement à partir de l'algorigramme que j'ai pu expliciter la machine de Moore.

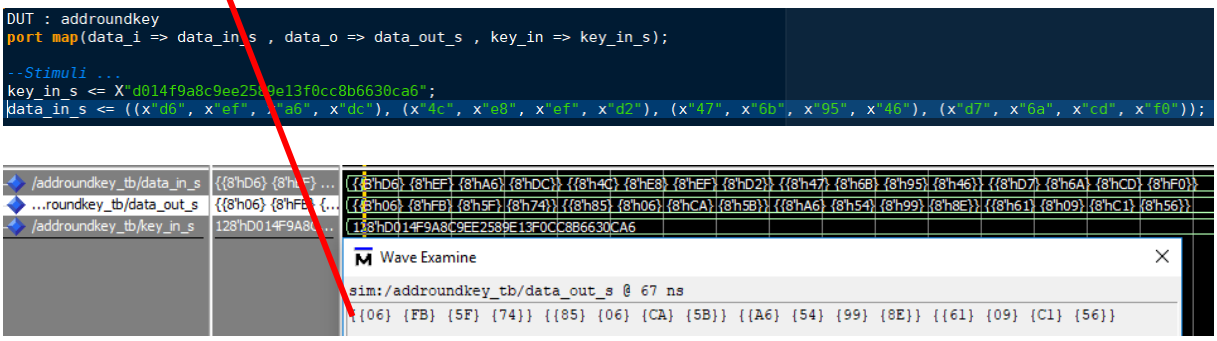
Certains choix que j'ai fait sont discutables. Par exemple sur l'état4 je reset le counter et j'indique que l'aes est fini, on pourrait aussi bien laisser aes_on_o à 1 et ne pas déclencher le reset du counter puisque cela sera de toute façon fait au prochain coup d'horloge sur l'état0**. En réalité peu importe, le plus important sur l'état4 est d'avoir enableOutput_o sur '1'.**

Test unitaire : AddRoundKey

Fichiers : AddRoundKey.vhd, AddRoundKey_tb.vhd

```
SetCiphertext : d6 ef a6 dc 4c e8 ef d2 47 6b 95 46 d7 6a cd f0

KeyExpansion (round 10): d0 14 f9 a8 c9 ee 25 89 e1 3f 0c c8 b6 63 0c a6
Round 10
AddRoundKey : 06 fb 5f 74 85 06 ca 5b a6 54 99 8e 61 09 c1 56
```

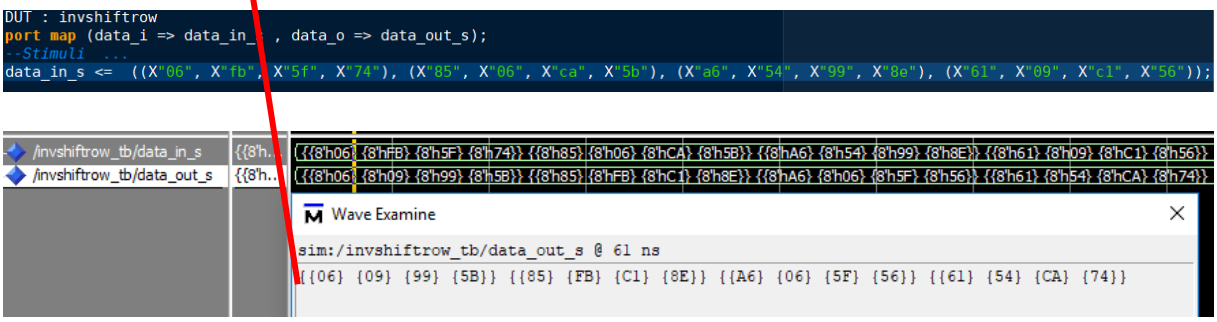


Le test unitaire est validé. Pour le round10, je prends le cipherText, et la clé10 en entrée.

Test unitaire : InvShiftRow

Fichiers : InvShiftRow.vhd, InvShiftRow_tb.vhd

```
Round 10
AddRoundKey : 06 fb 5f 74 85 06 ca 5b a6 54 99 8e 61 09 c1 56
Round 9
InvShiftRows : 06 09 99 5b 85 fb c1 8e a6 06 5f 56 61 54 ca 74
```

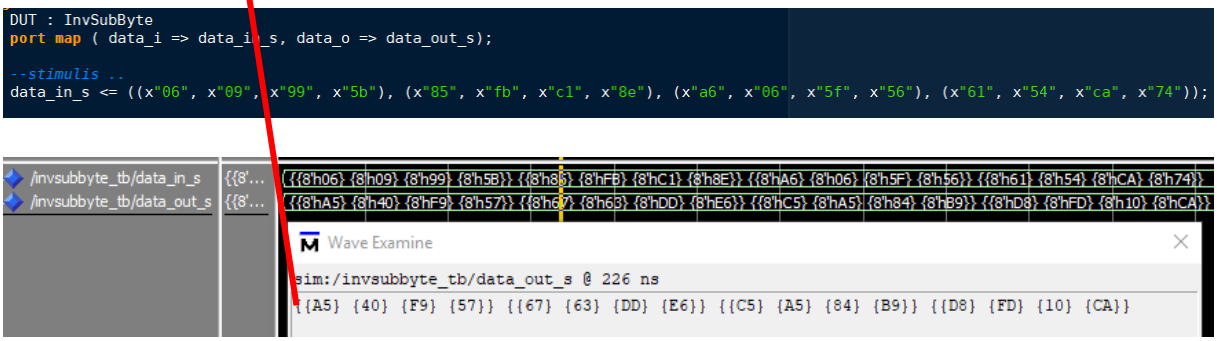


Le test unitaire est validé. Pour le round9, je prends la sortie de AddRoundKey comme stimuli en entrée de InvShiftRow.

Test unitaire : InvSubByte

Fichiers : InvSubByte.vhd, SBOX.vhd, InvSubByte_tb.vhd

```
Round 9
InvShiftRows : 06 09 99 5b 85 fb c1 8e a6 06 5f 56 61 54 ca 74
InvSubBytes : a5 40 f9 57 67 63 dd e6 c5 a5 84 b9 d8 fd 10 ca
```

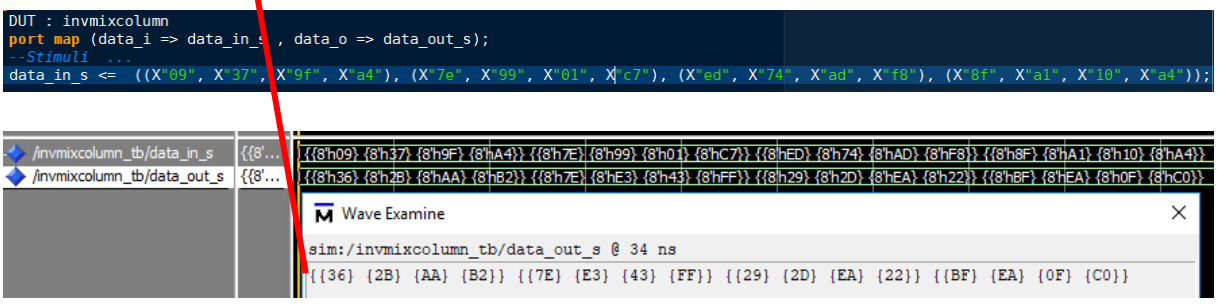


Le test unitaire est validé. Pour le round9, je prends la sortie de InvShiftRow comme stimuli en entrée de InvSubByte, j'ai bien le résultat attendu.

Test unitaire : InvMixColumn

Fichiers : InvMixColumnElementary.vhd, InvMixColumn.vhd, InvMixColumn_tb.vhd

```
Round 9
InvShiftRows : 06 09 99 5b 85 fb c1 8e a6 06 5f 56 61 54 ca 74
InvSubBytes : a5 40 f9 57 67 63 dd e6 c5 a5 84 b9 d8 fd 10 ca
AddRoundKey : 09 37 9f a4 7e 99 01 c7 ed 74 ad f8 8f a1 10 a4
InvMixColumns : 36 2b aa b2 7e e3 43 ff 29 2d ea 22 bf ea 0f c0
```

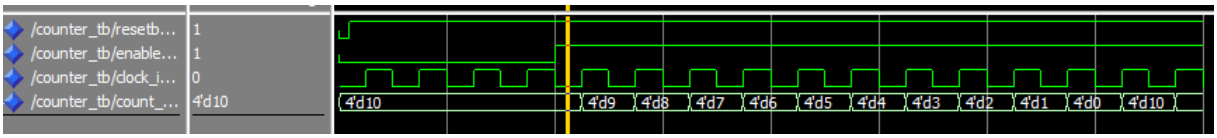


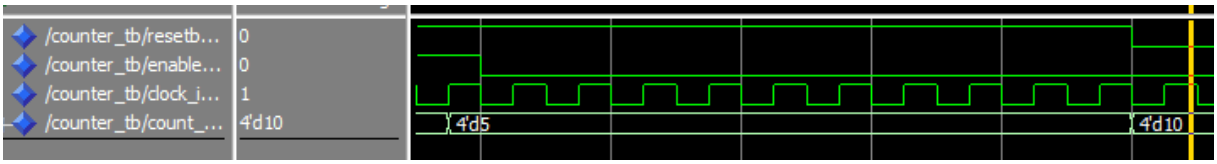
Le test unitaire est validé. Pour le round9, je prends la sortie de AddRoundKey comme stimuli en entrée de InvMixColumn.

Test unitaire : Counter

Fichiers : counter.vhd, counter_tb.vhd

```
DUT : counter
port map (  resetb_i => resetb_in_s,
            enable_i => enable_in_s,
            clock_i => clock_in_s,
            count_o => count_out_s);
--Stimuli...
resetb_in_s <= '0', '1' after 10 ns, '0' after 1500 ns;
enable_in_s <= '0', '1' after 200 ns, '0' after 1000 ns;
clock_in_s <= not(clock_in_s) after 25 ns;
```





Le test unitaire est validé. Le counter décrémente de 10 à 0, à chaque coup d'horloge lorsque enable et reset sont à 1. Enable_i à 0 bloque le compteur, resetb_i à 0 bloque le compteur sur la valeur 10.

Test unitaire : InvAESRound

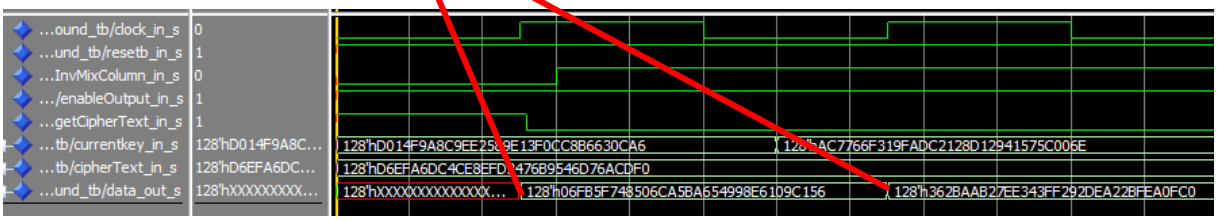
Fichiers : InvAESRound.vhd, InvAESRound_tb.vhd, RegisterComp.vhd

```
DUT : InvAESRound
port map (
   clock_i => clock_in_s,
   resetb_i => resetb_in_s,
   enableInvMixColumn_i => enableInvMixColumn_in_s,
   enableOutput_i => enableOutput_in_s,
   getCipherText_i => getCipherText_in_s,
   currentkey_i => currentkey_in_s,
   cipherText_i => cipherText_in_s,
   data_o => data_out_s );

--Stimuli ...
clock_in_s <= not(clock_in_s) after 25 ns;
resetb_in_s <= '1', '0' after 500 ns;
enableOutput_in_s <= '1';
enableInvMixColumn_in_s <= '0', '1' after 30 ns;
getCipherText_in_s <= '1', '0' after 26 ns;

currentkey_in_s <= x"d014f9a8c9ee2589e13f0cc8b6630ca6", x"ac7766f319fadc2128d12941575c006e" after 60 ns;
cipherText_in_s <= x"d6efa6dc4ce8efd2476b9546d76acdf0";
```

```
Round 10
AddRoundKey : 06 fb 5f 74 85 06 ca 5b a6 54 99 8e 61 09 c1 56
Round 9
InvShiftRows : 06 09 99 5b 85 fb c1 8e a6 06 5f 56 61 54 ca 74
InvSubBytes : a5 40 f9 57 67 63 dd e6 c5 a5 84 b9 d8 fd 10 ca
AddRoundKey : 09 37 9f a4 7e 99 01 c7 ed 74 ad f8 8f a1 10 a4
InvMixColumns : 36 2b aa b2 7e e3 43 ff 29 2d ea 22 bf ea 0f c0
```



Le test unitaire est validé. En entrée de cipherText_in il y a le cipherText de l'énoncé, puis il y a 2 clés de rondes, la 10 et la 9. Le resetb_i est à '1', enableOutput_i est à '1' pour observer la sortie en continu. GetCipherText_i est initialement à '1' puis à '0', au premier front montant d'horloge on obtient le résultat de la ronde 10. Puis la clé change dans le testbench et au coup d'horloge suivant la ronde 9 est effectuée et le résultat est valide.

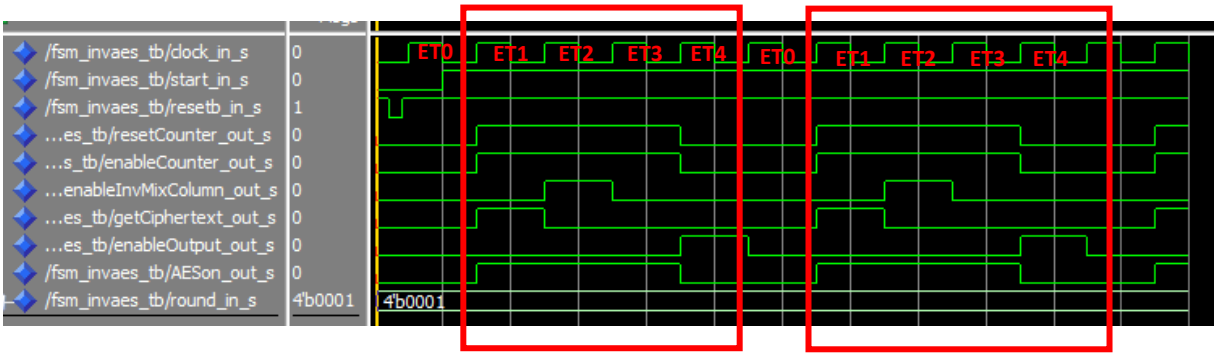
Ici les signaux : resetb_in_s, InvMixColumn_in_s, enableOutput_in_s et getCipherText_in_s sont stimulé dans le testbench. Dans InvAES global ces signaux sont commandés par la FSM Moore en fonction des états.

Test unitaire : FSM_InvAES

Fichiers : FSM_InvAES.vhd, FSM_InvAES_tb.vhd

```
DUT : FSM_InvAES
port map (  resetb_i => resetb_in_s,
            start_i => start_in_s,
            clock_i => clock_in_s,
            round_i => round_in_s,
            resetCounter_o => resetCounter_out_s,
            enableCounter_o => enableCounter_out_s,
            enableInvMixColumn_o => enableInvMixColumn_out_s,
            getCiphertext_o => getCiphertext_out_s,
            enableOutput_o => enableOutput_out_s,
            AES_on_o => AESon_out_s );

-- Stimuli ...
start_in_s <= '1'after 50 ns;
resetb_in_s <= '0' after 10 ns, '1' after 20 ns;
clock_in_s <= not (clock_in_s) after 25 ns;
round_in_s <= std_logic_vector(to_unsigned(1,4));
```



Le test unitaire est validé. A noter que dans cet exemple round_in_s est fixe à 0b0001, on observe bien seulement 4 coup d'horloge front montant par cycle de déchiffrement ie lorsque AESon_out_s vaut '1'. Le déchiffrement commence bien au premier coup d'horloge suivant le passage de start_i à '1'. On voit aussi les évolutions de getCipherText_o, enableInvMixColumn_o et enableOutput_o, qui permettent de distinguer dans quelle état de la FSM on se trouve.

Explication d'un déchiffrement : boite rouge

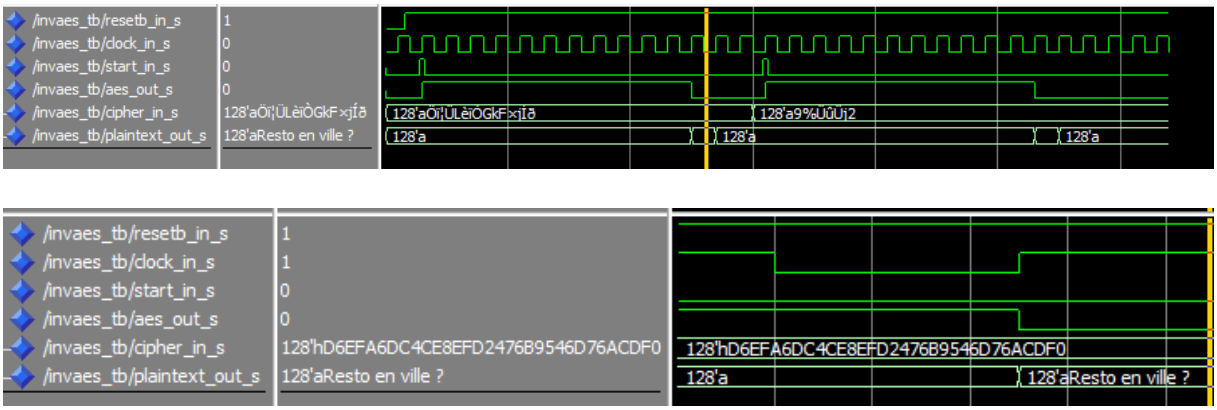
Au premier coup d'horloge front montant suivant le passage de start_i à '1', getCipherText_o passe à '1', il s'agit de la ronde10. Puis au coup suivant enableInvMixColumn_o passe à '1', il s'agit d'une ronde courante (ronde 9 à ronde 1). Comme round_in_s est fixé à 0b0001 par la simulation, au coup suivant on entre dans la ronde 0, enableInvMixColumn_o repasse donc à '0'. Puis enfin au coup suivant le déchiffrement est fini et enableOutput_o passe à '1'.

Résultat InvAES

Les tests unitaires de tous les modules constituants InvAES sont validés. Après tout le travail fait, écrire l'architecture de InvAES est une formalité. J'instancie les composants : InvAESRound, counter, FSM_InvAES et KeyExpansion_Table. Puis je mappe les signaux input / outpur de InvAES et déclare des signaux internes nécéssaires.

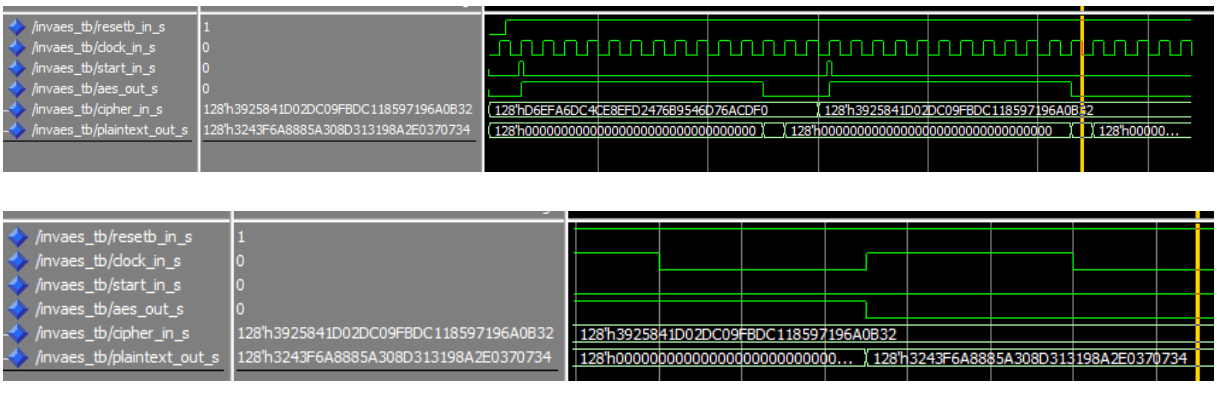
```
--Sitmuli ...
clock_in_s <= not (clock_in_s) after 50 ns; --cadencement
start_in_s <= '0', '1' after 140 ns, '0' after 160 ns, '1' after 1540 ns, '0' after 1560 ns;
resetb_in_s <= '1'after 80 ns;
            --Set cipher text
cipher_in_s <= x"d6efa6dc4ce8efd2476b9546d76acdf0", x"3925841d02dc09fbdc118597196a0b32" after 1500 ns;
```

Dans le testbench de InvAES je lance 2 déchiffrements, qui correspondent aux 2 donnés dans l'énoncé du projet.



Le premier déchiffrement nous renvois bien « Resto en ville ? ».

avec le message chiffré $0x3925841d02dc09fbdc118597196a0b32$
donnant le message en clair $0x3243f6a8885a308d313198a2e0370734$.



Le second déchiffrement renvoie bien la valeur attendue.

En conclusion, mon InvAES global fonctionne sur les deux exemples donnés.

J'ai eu des difficultés principalement sur le module InvAESRound car je n'avais pas adopté une vision matérielle. Après une entrevue avec Mr. Potin j'ai rapidement compris comment développer ce module. Le plus difficile selon moi a été de comprendre la source des erreurs de simulation : soit un problème dans le module, soit un problème dans le testbench.

Les points bloquants habituels que sont l'appropriation du sujet, de la technologie et de l'outils de simulation ont rapidement été résolus.

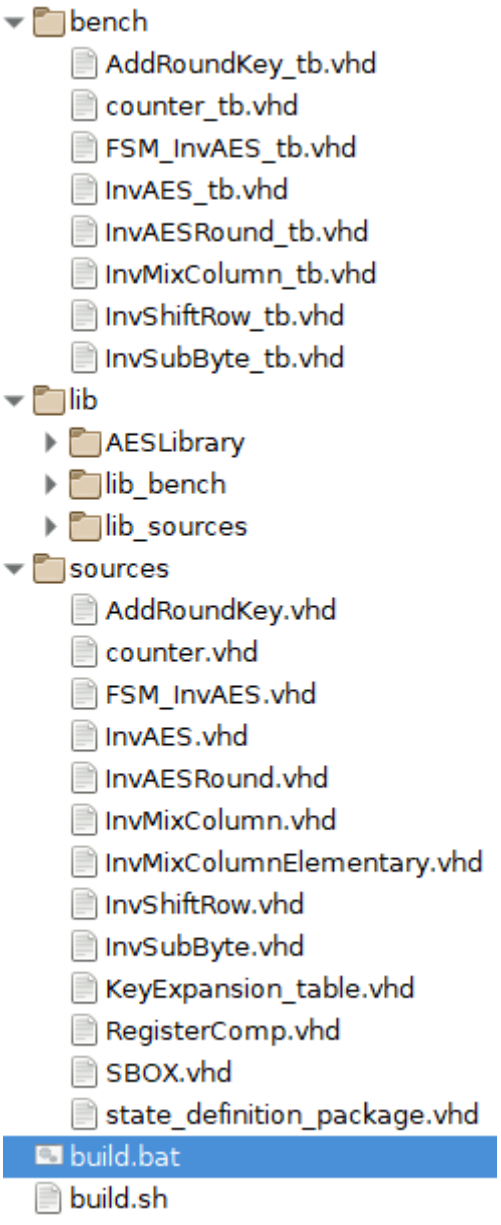


Fig8. Organisation du projet