ECOLE DES MINES DE SAINT ETIENNE
CAMPUS G. CHARPAK PROVENCE

Rapport

Projet Conception de Système Numérique

Maïssa ELAFANI

17 novembre 2024

Table des matières

1 INTRODUCTION

Dans le cadre du Projet de Conception de Système Numérique, l'objectif était de s'appuyer sur la théorie de la cryptographie pour chiffrer et déchiffrer des messages en utilisant une clé secrète partagée en amont.
Pour ce faire, nous avons utilisé le chiffrement authentifié avec données associées (AEAD) à travers l'algorithme ASCON128.

1.1 L'algorithme de chiffrement AEAD

L'algorithme AEAD (Authenticated Encryption with Associated Data) a été conçu pour répondre aux besoin de confidentialité et d'intégrité des données. Depuis son apparition, il est majoritairement utilisé comme solution pour protéger les échanges d'informations sensibles.

L'AEAD est souvent employé dans des contextes où la sécurité des données est cruciale tels que les transactions en ligne ou encore les communications via Internet comme c'est le cas pour ce projet.

L'algorithme AEAD prend en entrée un bloc de données, une clé de chiffrement, des données associées et un vecteur d'initalisation, et le résultat final est un texte chiffré qui peut être transmis en toute sécurité.
De plus, l'un des aspects importants de l'AEAD est son utilisation de mécanismes d'authentification qui permettent de vérifier l'intégrité des données lors de leur déchiffrement.

Dans le cadre du projet, nous réaliserons l'implémentation de l'algorithme ASCON128 en utilisant le langage SystemVerilog.

2 Description de l'algorithme ASCON128

2.1 Fonctionnement de l'ASCON128

L'algorithme de chiffrement ASCON128 prend en entrée un message sur 256 bits et retourne un message chiffré sur 248 bits. Le fonctionnement de l'algorithme se base sur la propagation d'un état courant qui va subir un certain nombre d'opérations élémentaires. L'évolution au cours du temps de cet état courant nous permettra par la suite d'obtenir le message chiffré aisi que le tag qui sert à l'authentification du message chiffré.
Le chiffrement ASCON128 se décompose en 4 parties :

- Initialisation : l'état courant est généré à partir de la clé de chiffrement du Nonce et du vecteur d'initialisation.
- Données associées : l'état courant interagit avec la donnée associée qui caractérise le message à chiffré (IP, date, etc...).
- Chiffrement du message : le message est chiffré en utilisant l'état courant du système.
- Finalisation : un tag est généré et servira par la suite pour authentifier et vérifier la validité du message.

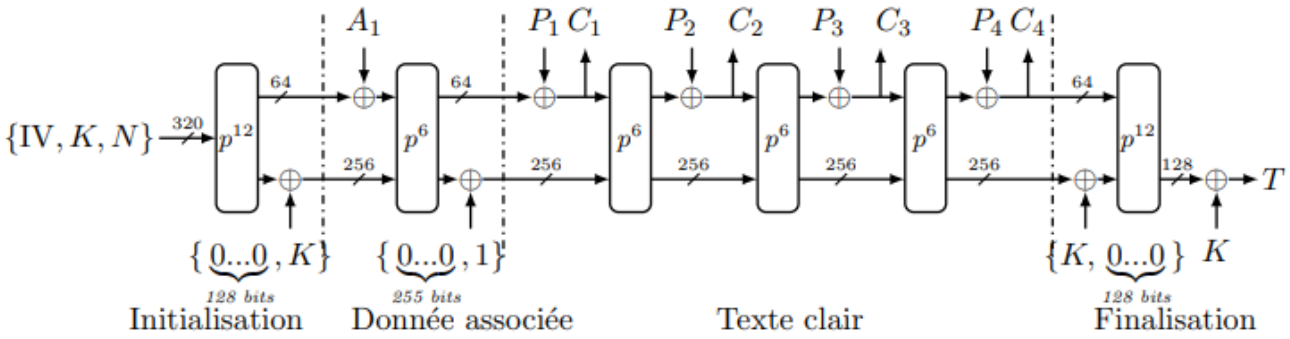

FIGURE 1 – Chiffrement ASCON128

2.2 Découpage des élements

Un processus de chiffrement ASCON128 peut être décomposé en plusieurs parties différentes :

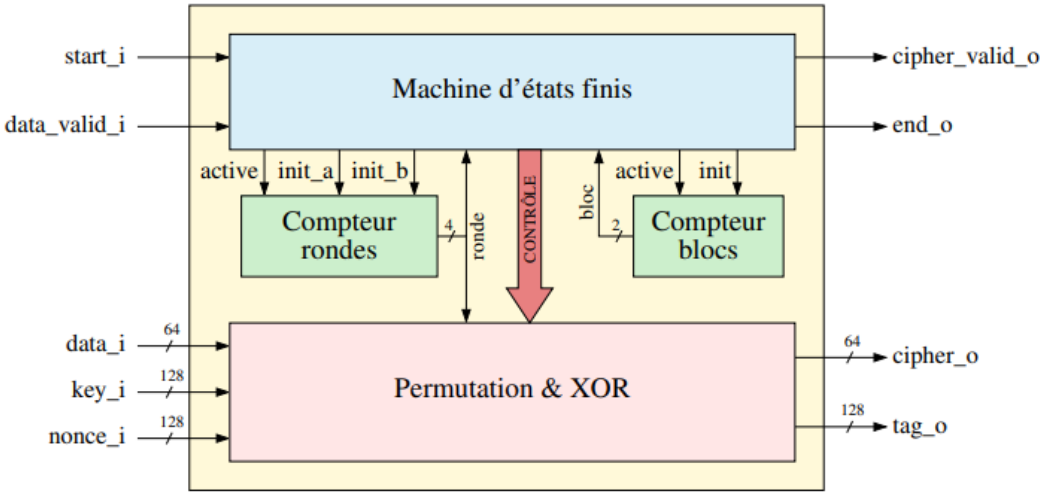

FIGURE 2 – Architecture globale de ASCON128

Dans un premier temps, il s'agit de coder les transformations permettant la réalisation du composant "Permutation with XOR".
Dans un second temps, il sera nécessaire de gérer la partie "haute" de l'algorithme avec les composants qui permettront de réaliser l'ensemble de l'algorithme de chiffrement. Et en parallèle, une machine d'état associée à un compteur permettront de mettre en place l'ensemble du processus.

2.3 Structure du projet

2.3.1 Organisation du code

Le code pour la réalisation de ce projet a été separé en deux parties : la partie modules et la partie testing (Test Bench).
L'ensemble des modules se trouve dans le dossier (/SRC/LIB/RTL). Par soucis de clarté, les noms des modules crées seront les mêmes que ceux du fichier de code correspondant.
Les fichiers de testing se trouvent dans (/SRC/LIB/BENCH) et ils sont reconnaissables puisqu'ils sont de la forme *nom_tb.sv* .
Dans le cadre du projet, nous utilisons le package "$ascon_pack$" fourni dans les documents de référence et qui définit les constantes utiles. Son fichier se trouve dans (/SRC/LIB/RTL).

2.3.2 Formalisme

Pour favoriser la lisibilité, la clarté et la réutilisation du code, un certain nombre de formalismes ont été mis en place :

— Un module doit impérativement avoir le même nom que le fichier dans lequel il est défini.

— Tout signal correspondant à une entrée sera de la forme "xxxxxx_i" .
— Tout signal correspondant à une sortie sera de la forme "xxxxxx_o" .
— Tout signal correspondant à un signal de test sera de la forme "xxxxxx_s" .

3 Permutations et XOR

Au cours du chiffrement ASCON128, l'état courant de 320 bits subira un certain nombre de permutations ainsi que l'action des portes XOR, que ce soit sur sa partie haute (donc les premiers 64 bits) ou sur sa partie basse (donc les derniers 256 bits).
Chaque permutation peut être décomposée en 3 opérations élémentaires consécutives : l'addition de constante, la couche de substitution et la diffusion linéaire.

3.1 Permutations

3.1.1 Addition de constante

Principe L'état courant est composé de 5 registres de 64 bits (x0 x1 x2 x3 x4). L'addition constante n'intervient que sur le registre x2 auquel elle va ajouter une constante C_r. Le choix de cette constante est défini par une ronde i .

$$x_2 \leftarrow x_2 \oplus C_r \tag{1}$$

Ronde r de p^{12}	Ronde r de p^6	Constante c_r
0		00000000000000000f0
1		00000000000000000e1
2		00000000000000000d2
3		00000000000000000c3
4		00000000000000000b4
5		00000000000000000a5
6	0	00000000000000000096
7	1	00000000000000000087
8	2	00000000000000000078
9	3	00000000000000000069
10	4	0000000000000000005a
11	5	0000000000000000004b

FIGURE 3 – Définition des constantes C_r

Modèle Pour la modélisation de l'addition constante, nous allons créer un module qui prend en entrée l'état courant **Etat_i** sur 320 bits et la ronde **Round_i** sur 4 bits puisqu'on peut avoir au maximum i=11 et nous avons en sortie l'état courant modifié **Etat_o**.

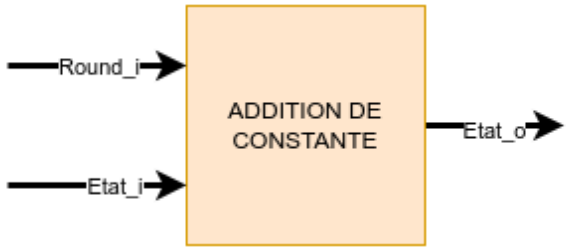

FIGURE 4 – Modèle addition constante

Simulation On simule le bloc "addition constante" à l'aide du logiciel *Modelsim*. On prend en entree l'état courant initial du chiffrement ASCON128.

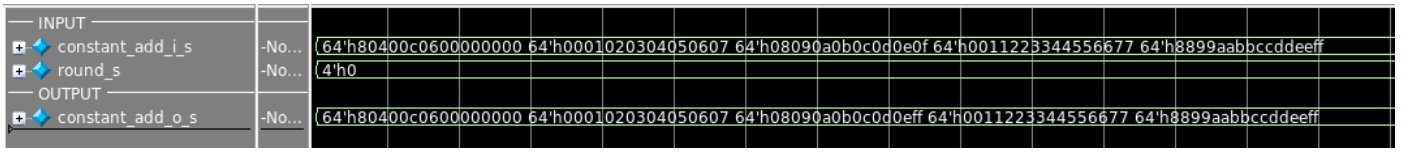

FIGURE 5 – Résultat simulation addition de constante

La simulation est bien concluante puisque seule la partie x_2 de l'état a été modifié et c'est bien la valeur 0x0000000000000000f0 qui a été ajoutée.

3.1.2 Couche de substitution

Principe La couche de substitution modifie l'état courant en appliquant en parallèle la substitution de 5 bits en colonne en utilisant une table de substitution. Cette opération agit de façon indépendante sur chaque colonne de l'état courant, donc elle agit sur $[x_1(i), x_2(i), x_2(i), x_3(i), x_4(i)]$ et cela pour i allant de 0 à 63.

x	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
$S(x)$	04	0B	1F	14	1A	15	09	02	1B	05	08	12	1D	03	06	1C	1E	13	07	0E	00	0D	11	18	10	0C	01	19	16	0A	0F	17

FIGURE 6 – Table de substitution

Modèle Cette couche de substitution agit sur les 64 colonnes de 5 bits qui composent l'état courant, il faut donc effecteur 64 opérations pour mener à bien l'opération. Pour gagner en performance, nous décidons de concevoir la couche de substitution comme la juxstapposition de 64 composants qui travailleront de façon concurrente. Chacun fera le calcul uniquement sur une colonne. Cette approche nous permet donc de gagner en performance.

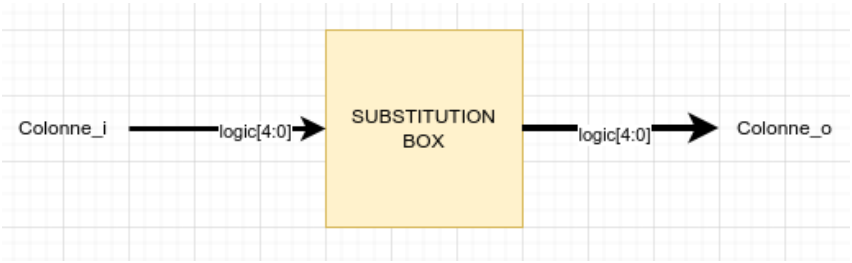

FIGURE 7 – Substitution élementaire

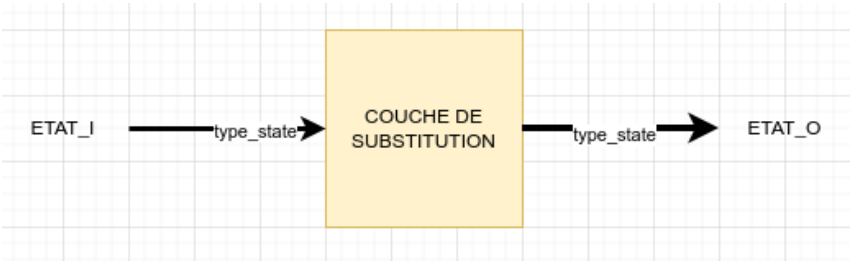

FIGURE 8 – Couche de substitution

Simulation Nous avons tout d'abord, simulé le fait que la substitution élémentaire qui elle n'agit que sur une colonne en faisant bien attention à la cohérence avec le tableau de correspondance.

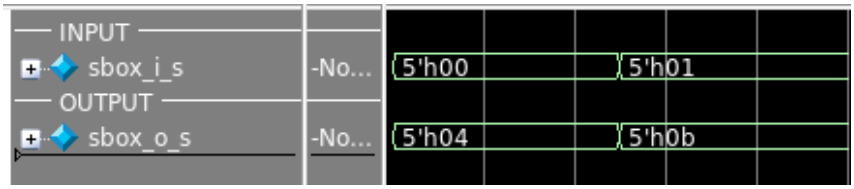

FIGURE 9 – Simulation substitution élémentaire

Enfin, nous avons simulé l'action de la couche de substitution sur l'ensemble de l'état courant.

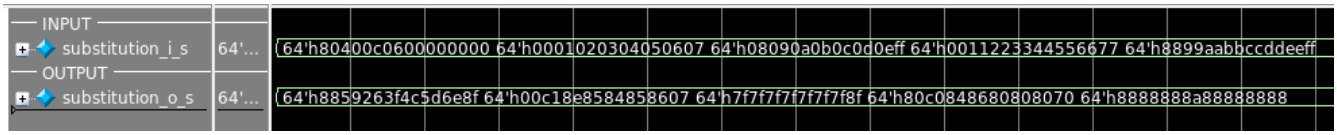

FIGURE 10 – Simulation couche de substitution

Les simulations sont bien cohérentes avec les résultats attendus.

3.1.3 Couche de diffusion linéaire

Principe Cette couche effectue une opération de décalage et XOR sur chaque x_i qui compose l'état courant : on lui applique une diffusion.

$$
\begin{array}{lllllllllllll}
x_0 & \leftarrow & \Sigma_0(x_0) & = & x_0 & \oplus & (x_0 & \ggg & 19) & \oplus & (x_0 & \ggg & 28) \\
x_1 & \leftarrow & \Sigma_1(x_1) & = & x_1 & \oplus & (x_1 & \ggg & 61) & \oplus & (x_1 & \ggg & 39) \\
x_2 & \leftarrow & \Sigma_2(x_2) & = & x_2 & \oplus & (x_2 & \ggg & 1) & \oplus & (x_2 & \ggg & 6) \\
x_3 & \leftarrow & \Sigma_3(x_3) & = & x_3 & \oplus & (x_3 & \ggg & 10) & \oplus & (x_3 & \ggg & 17) \\
x_4 & \leftarrow & \Sigma_4(x_4) & = & x_4 & \oplus & (x_4 & \ggg & 7) & \oplus & (x_4 & \ggg & 41)
\end{array}
$$

FIGURE 11 – Simulation couche de substitution

Modèle Pour modéliser cette diffusion linéaire, nous avons conçu un bloc qui prend en entrée l'état courant **Etat_i** dans sa totalité. Et nous retrouvons par la suite cet état en sortie du bloc **Etat_o**.

FIGURE 12 – Modèle couche de diffusion linéaire

Simulaion Pour la simulation nous avons mis en entrée l'état courant.

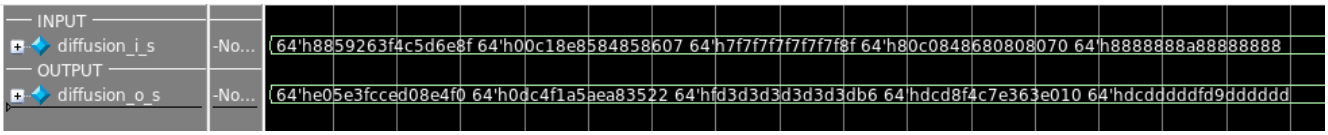

FIGURE 13 – Simulation de la couche de diffusion linéaire

3.1.4 Permutation

Principe Nous avons regroupé les 3 opérations élémentaires en un seul bloc qui représentera la permutation. Cette permutation représente l'opération que va subir notre état. Il y a deux cas possibles : soit il va la subir 6 fois p soit il va la subir 12 fois p^{12}.

Simulation Pour la simulation, nous avons utilisé l'état courant initial auquel nous avons appliqué 12 permutations en modifiant à chaque fois la valeur de la ronde i.

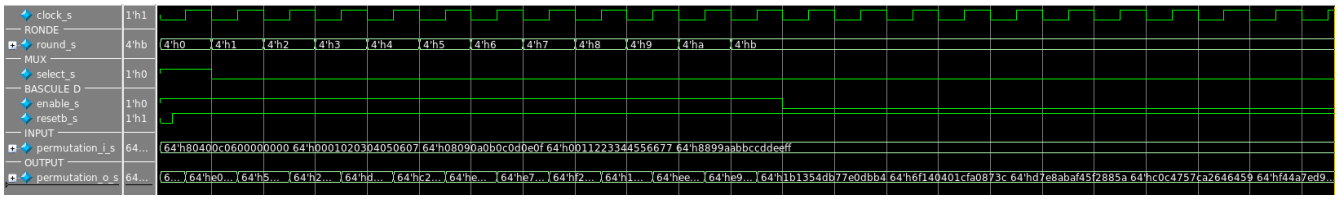

FIGURE 14 – Simulation permutation

3.2 Opération XOR

Entre deux rondes de permutation l'état courant subit des opérations de type XOR qui modifient sa partie haute ou sa partie basse. En analysant de plus près la structure de l'algorithme de chiffrement de l'ASCON128, on a identifié deux types d'opérations XOR différentes.

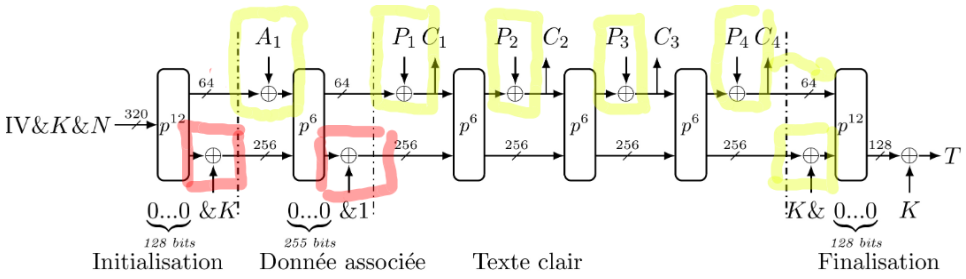

FIGURE 15 – Analyse de l'algorithme

Nous distinguons les "xor_up" qui interviennent en amont des permutation et les "xor_down" qui interviennent après une permutation.

3.2.1 XOR UP

Principe Ce type de XOR effectue des opérations principalement avec les données associées sur 64 bits à l'exception de la dernière opération qui elle utilise la clé de chiffrement sur 128.

Modèle Le composant xor_up prend en entrée l'état courant **Etat_i** mais prend aussi un signal d'allumage en entrée **Enable_i** qui sert à éteindre ou allumer la porte XOR, nous avons enfin le signal **Mode_i** qui sert a sélectionnner le mode de fonctionnnement (avec les données associées ou avec la clé de chiffrement).

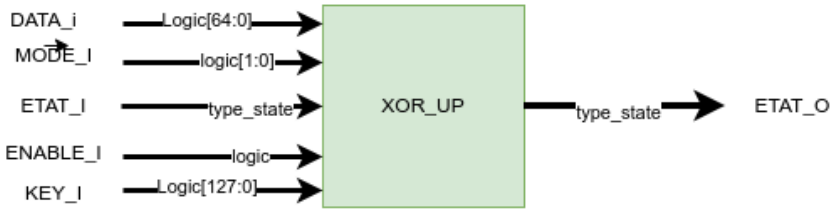

FIGURE 16 – MOdèle xor_up

Simulation Nous avons simulé l'action de l'opération XOR avec les données associées sur un état courant défini.

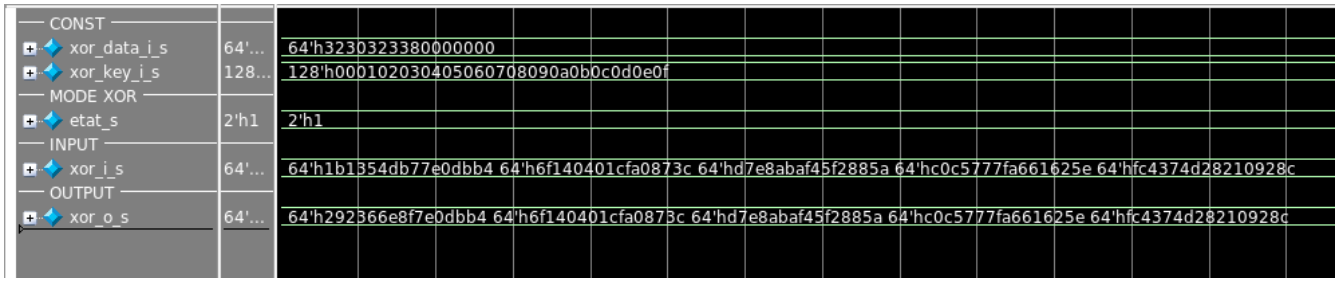

FIGURE 17 – Simulation Xor_up

3.2.2 XOR DOWN

Principe Ce composant est très similaire au "xor_up", il se différencie par le fait d'agir exclusivement sur la partie basse de l'état courant donc les opérations d'effectuent sur 256 bits.

Modèle Le modèle est très similaire à celui de xor_up à l'exception qu'on a pas besoin de connaître les données associées.

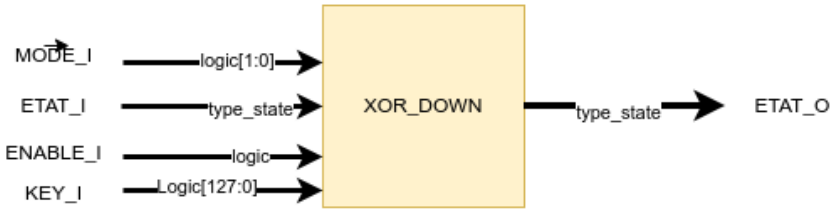

FIGURE 18 – Simulation Xor_up

3.2.3 Permutaion avec les portes XOR

Pour gagner en efficacité les portes xor_up et xor_down ont été directement intégrées à l'intérieur du bloc permutation.

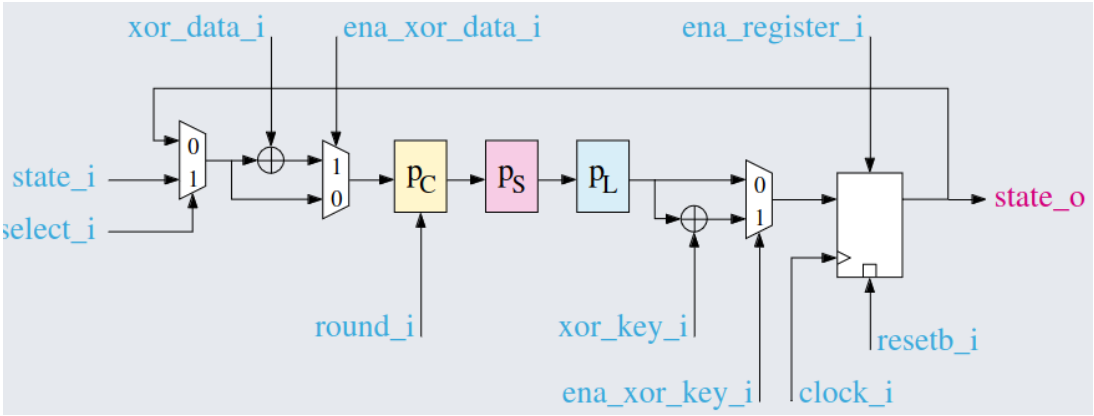

FIGURE 19 – Simulation Xor_up

Avec les signaux de contrôle des portes XOR crées nous pouvons donc les allumer que au début pour le xor_up et seulement à la fin pour le xor_down.

Simulation Pour cette simulation un contexte plus exhaustif à été mis en place en simulant le chiffrement à partir du tout debut avec l'état courant initial jusqu'à l'opération avec les données associées, ce qui nous a permis de tester à la fois les xor_up et xor_down.

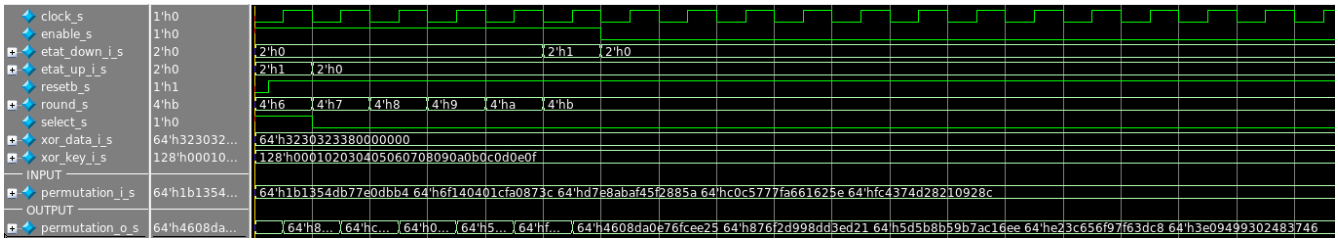

FIGURE 20 – Simulation permutation avec les xors

3.2.4 Cipher et TAG

Principe Le but de ce chiffrement ASCON128 est de génénrer le message chiffré et le TAG, pour cela nous allons introduire dans la permutation deux registres à bascule D qui nous permettront de lire les données utiles pendant que le ASCON128 effectue le chiffrement.

Modèle Le composant qui nous pemettra de récupérer ces données est une bascule D soit de 64 bits pour le cipher soit de 128 bits pour le TAG.

Simulation Nous avons simulé ici la capacité de notre permutation à lire les cipher générés par chaque partie du message.

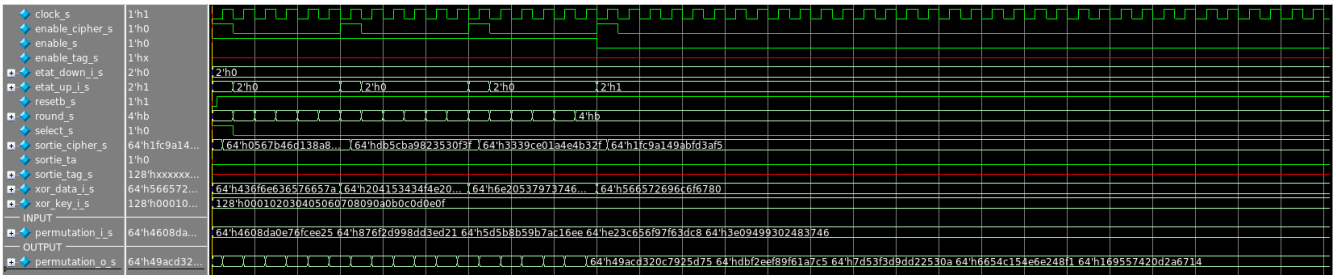

FIGURE 21 – Simulation permutation complète

La simulation est encore ici cohérente avec les résultats attendus.

4 Machine d'etats finis

Pour synchroniser l'ensemble des opérations que doit effectuer le bloc "permutation & XOR" nous allons essayer de concevoir une machine de Moore qui décrit les différents états du chiffrement. Pour ce faire, nous aurons besoin de deux indicateurs principaux la valeur de la ronde et le rang du bloc de message qu'on est en train de traiter pour cela en plus de la machine nous aurons besoin de deux compteurs.

4.1 Compteurs

4.1.1 Compteur de ronde

Principe Ce compteur doit être capable de compter au moins jusqu'à 11 qui correspond à la valeur maximale de la ronde, il sera donc sur 4bits. L'incréméntation du compteur se fera selon le signal de la clock. Il permettra aussi de selectionnner le départ 0 pour $p12$ ou 6 pour $p6$.

Modèle Comme on peut voir dans ce modèle en entrée, on retrouve le signal **clock_i** celui de reset **rest_i** mais aussi un signal d'activation **enable_i** qui contrôle l'allumage du compteur enfin les deux input **inita_i** et **initb_i** règlement le début du compteur. En sortie nous avons un chiffre sur 4 bits **cpt_o**.

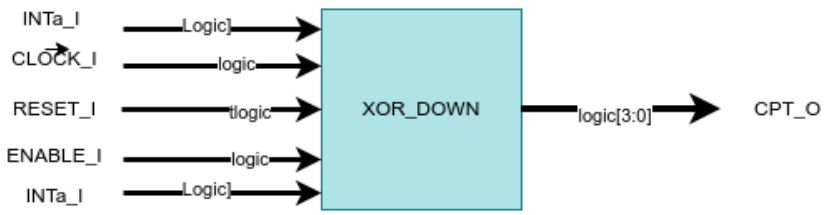

FIGURE 22 – Modèle compteur de ronde

4.1.2 Compteur de blocs

principe Ce composant sera très similaire au compteur vu précedemment à l'exception du fait qu'il sera sur 2 bits et commencera forcément à zéro donc pas besoin de signaux de contrôle supplémentaires.

Modèle Modèle du compteur de blocs :

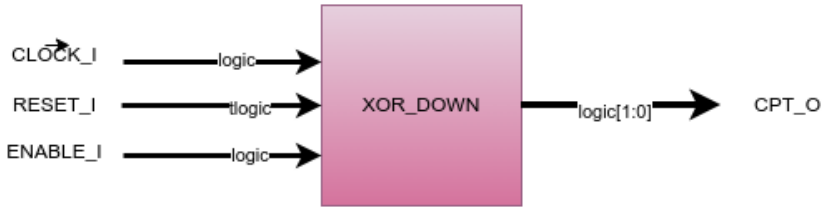

FIGURE 23 – Modèle compteur de blocs

5 Machine d'état de Moore

5.1 Description

La machine d'état doit être capable de choisir la bonne configuration des variables de contrôle pour un état donné. Elle sera majoritairement commandé par la variable **start_i** qui déclenche le debut du chiffrement et la variable **data_valid_i** qui indique la validité des données et permet ainsi a la machine de poursuivre, dans lecas contraire l'ensemble de l'algorithme est arreté. De plus la machine d'état permettra de commander les compteurs ainsi que tous les signaux qui contrôlent les permutations.

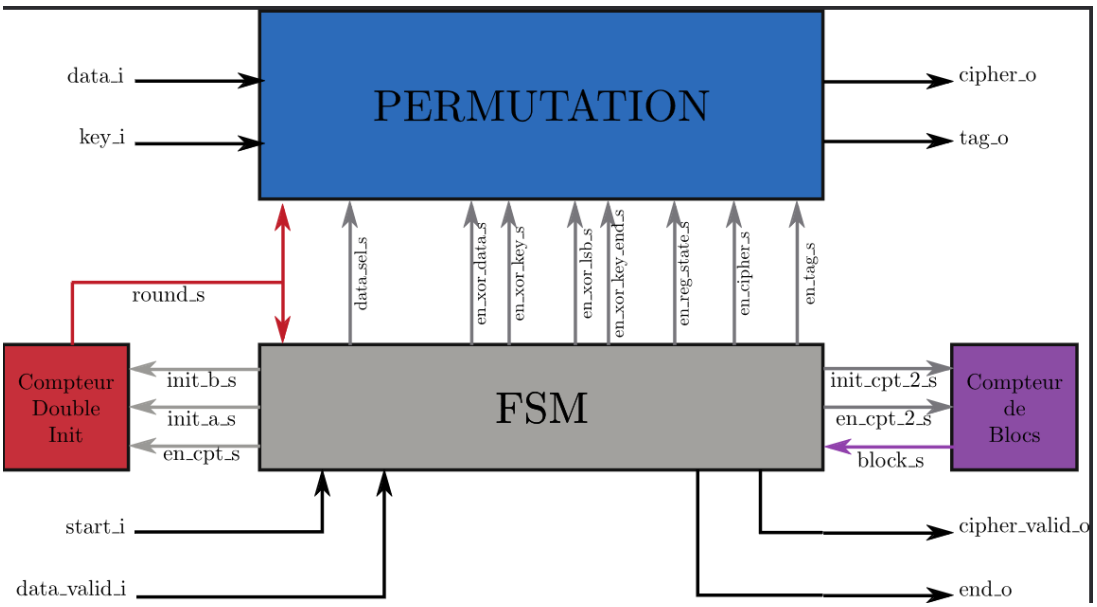

FIGURE 24 – Machine d'état

5.2 Graphe d'état

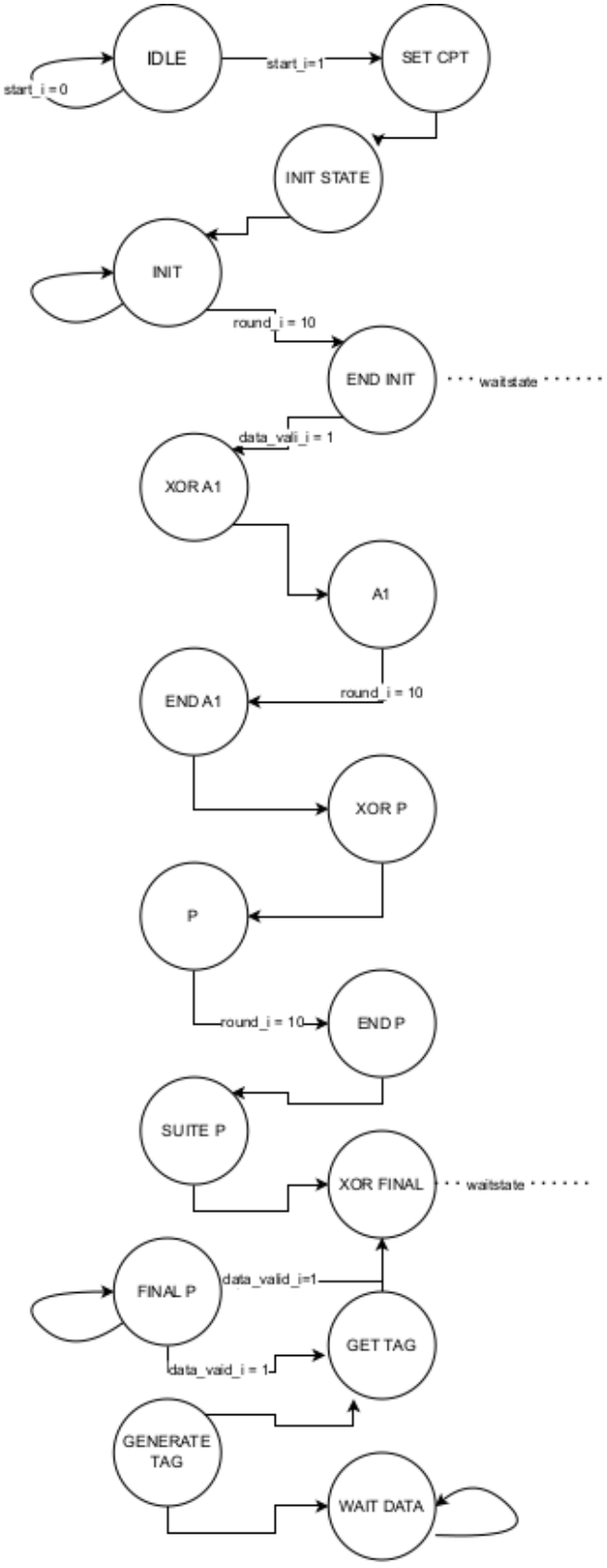

FIGURE 25 – Graphe d'état

6 Conclusion

Ce projet a été enrichissant tant sur le plan de la compréhension et l'implémentation de l'algorithme que sur l'apprentissage à travers le SystemVerilog.

Par ailleurs, j'ai trouvé ce projet difficile et long à réaliser avec beaucoup de travail hors période de cours mais l'enchaînement des cours permettant une évolution échelonnée de la difficulté des programmes à mettre en place, a tout de même représenté un accompagnement important et nécessaire pour ce projet.

Finalement, j'ai apprécié travailler sur ce projet car il a été très formateur.