Conception de Système Numérique

Modélisation SystemVerilog de l'algorithme de chiffrement ASCON

Auteur

SEBTI Adam

December 17, 2023

Contents

1 Introduction

Le chiffrement authentifié avec données associées (AEAD), largement utilisé dans les protocoles réseau, assure la confidentialité et l'authentification des communications. Pour préserver la confidentialité, le corps du message est chiffré, laissant l'en-tête du fichier (header) en clair. Cet en-tête contient des informations cruciales telles que les adresses IP source et destination, la taille du message, etc., nécessaires pour la redirection des paquets au sein de l'infrastructure réseau.

L'algorithme ASCON128 joue un rôle essentiel en garantissant non seulement la confidentialité du contenu des messages par le chiffrement, mais aussi l'authenticité de l'en-tête. Il réalise cela en associant un tag au message, que le destinataire doit vérifier. Si le destinataire ne parvient pas à recalculer la même valeur de tag, le message est rejeté, assurant ainsi l'intégrité et l'authenticité des communications.

L'objectif sera ici d'essayer d' implémenter l'algorithme de chiffrement ASCON128 en utilisant le langage de description matérielle Systemverilog pour représenter le comportement et la structure d'un circuit électronique dédié à ce chiffrement.

2 Description de l'algorithme ASCON128

2.1 Fonctionnement de l'algorithme

L'algorithme est initialisé avec un vecteur d'initialisation, la clé et le nonce. Le nonce protège lescommunications des attaques par rejeu. Dans le cas d'usage normal, il est différent à chaque chiffrement.

La seconde étape permet d'introduire la donnée associée dans le calcul de l'algorithme. Cette donnéepermet d'éviter qu'un attaquant remplace un message chiffré par un autre message chiffré avec la mêmeclé. La troisième étape consiste à introduire le message à chiffrer et à obtenir le message chiffré. La génération du tag est réalisée lors de la dernière étape. Le tag correspond à un hachage unique desdonnées chiffrées et il permet de vérifier l'intégrité du chiffrement. En cas d'attaque par rejeu ou dans le cas où le message chiffré est remplacé, si le nonce et la donnée associée sont différents à chaque chiffrement, alors le tag obtenu sera incorrect.

La taille du chemin de donnée est de 320 bits, divisé en 5 parties x0, x1, x2, x3, x4 de 64 bits. La permutation de ASCON est composée de 3 opérations : un ajout de constante sur le vecteur x2, une couche de substitution où, 64 Sbox 5 bits, sont appliquées en parallèle et une couche de diffusion.

Dans le cadre de notre projet , la donnée associée, le message à chiffrer et le message chiffré ont une taille de 64 bits et le message entier contiendra 4 sections de 64 bits, une ronde de la permutation s'effectue en un coup d'horloge.

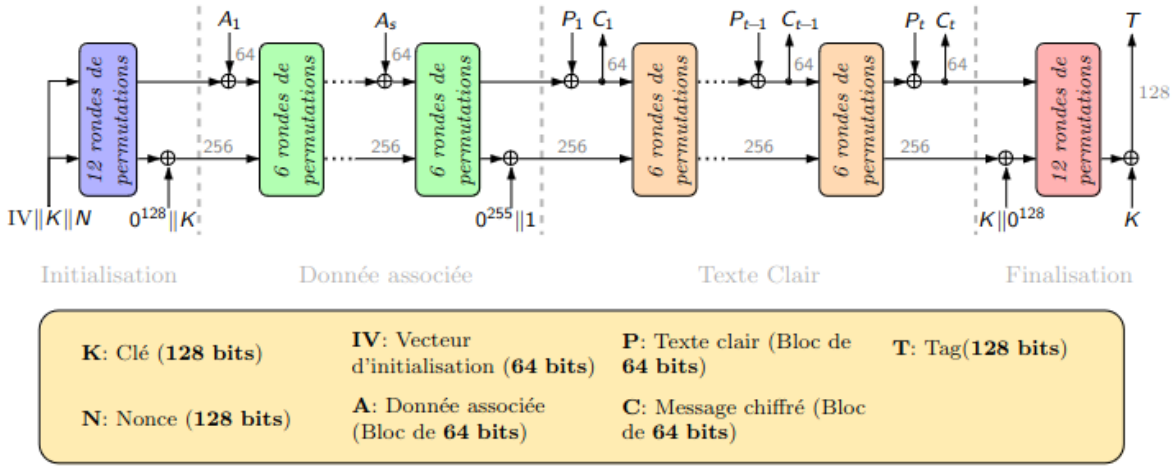

Figure 1: Schéma de ASCON-128

2.2 Notations et formalismes

2.2.1 Organisation du projet

Les fichiers liés aux architectures Systemverilog sont situés dans le répertoire "/SRC", pour ce projet deux bibliothèques associées on étés mises en place pour faciliter la compilation et l'exécution : "LIB_RTL" et "LIB_BENCH".

Les fichiers de testbench se trouvent dans le répertoire "SRC/BENCH". Les fichiers de testbench sont facilement identifiables grâce à leur nom du type xxx_tb.sv.

De plus, il existe un répertoire (SRC/RTL) qui contiennen les fichiers décrivant l'architecture du circuit des différents composants.

Enfin le projet comprend le pakcage "ascon_pack",ce dernier englobe les définitions de types couramment utilisés, tels que la structure type_state.

2.2.2 Formalismes

Afin d'assurer une meilleure lisibilité et le debug du code, les directives suivantes ont étés mises en place:

- Les signaux entrants sont complétés par le suffixe "_i"
- Les signaux sortants sont complétés par le suffixe "_o"
- Les signaux intermédiaires et de test ont le suffixe "_s"
- Les fichiers de testbench seront nommés : "nom_tb.sv"
- Un fichier de compilation sera mis en place

3 Implantation matérielle de l'algorithme

Avant d'entrer dans le vif du projet il est primordial de bien analyser le système électronique qui s'occupera du chiffrement ASCON128 afin de l'analyser plus dans le détail et proposer un découpage plus fin des différentes fonctions pour pour pouvoir tirer avantage de la modularité du langage Systemverilog et structurer notre travail.

3.1 Vue générale du module

Entrées/Sorties du module Suite à la description de l'algorithme faite précédemment on peut intuiter les ports d'entrées et de sortie dont notre module aura besoin :

- Une horloge **clock_i**
- Un signal d'initialisation **resetb_i**
- Une entrée **data_i** de 64 bits
- Une entrée **data_valid_i** indiquant la présence d'une donnée valide
- Une entrée pour la clé **key_i** de 128 bits
- Une entrée pour le nombre arbitraire **nonce_i** de 128 bits
- Une entrée **start_i** pour commencer le chiffrement
- Une sortie **cipher_o** sur 64 bits
- Une sortie **cipher_valid_o** indiquant la validité de la sortie cipher_o
- Une sortie **tag_o** sur 128 bits
- Une sortie **end_o** indiquant la fin du chiffrement du message

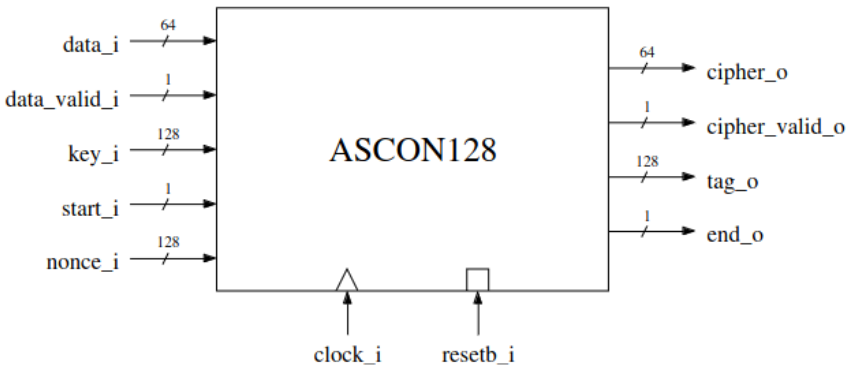

Figure 2: I/O du module ASCON128

3.2 Sous-modules

La structure d'ASCON est constituée de 4 sous-modules : une machine d'états (FSM) pour générer les signaux de contrôles, un compteur 4 bits pour exécuter le bon nombre de rondes (6 ou 12), la permutation et un compteur de blocs lors du chiffrage du message.

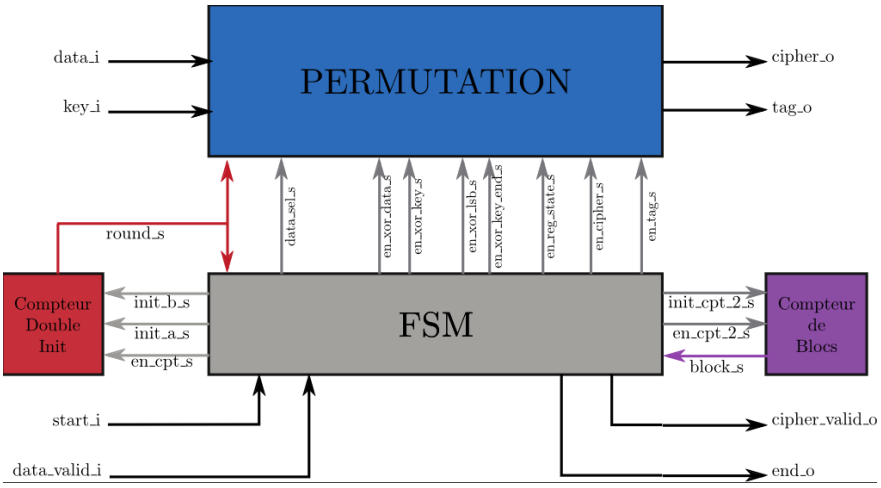

Figure 3: Sous modules du circuit ASCON128

Le sous-bloc de la permutation permet d'appliquer les opérandes sur l'état pour cela il contient un registre d'état courant sur 320 bits, un registre pour stocker le message chiffré sur 64 bits et un autre pour stocker la valeur du TAG. Ce sous-module contient aussi 3 multiplexeurs : le premier permettant de sélectionner l'entrée au choix parmi un état S calculé où un état initial déduit du nonce et du vecteur d'initialisation. Les deux autres multiplexeurs permettent de commander l'activation de deux portes XOR qui sont nécessaires lors du chiffrement.

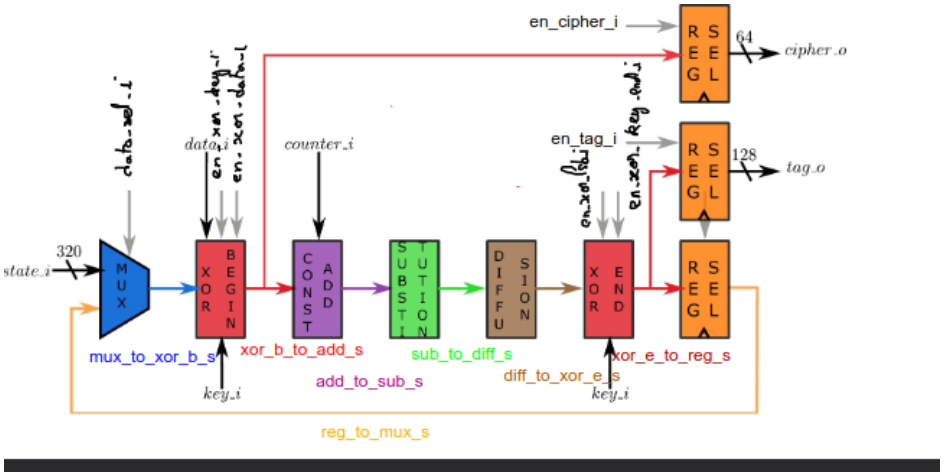

Figure 4: Structure du bloc permutation

4 Bloc permutation

4.1 Permutation

Le chiffrement de type ASCON128 se base sur l'application d'une permutation à l'état S courant du système qui est représentes par 5 registres de 64 bits x_0, x_1, x_2, x_3, x_4. Cette permutation est le résultat de opérandes élémentaires concaténées.

4.1.1 L'addition de constante

L'addition constante ajoute un constante C_r au registre x_2 de l'état courant, le choix de la constante est directement lié au rang de la ronde qui varie entre 0 et 11.

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

Donc notre sous-module prendra en entrée **Etat_i**(320 bits) et la **ronde_i**(4 bits) et nous donnera en sortie **Etat_o** qui sera l'état modifié.

Résultat de la simulation :

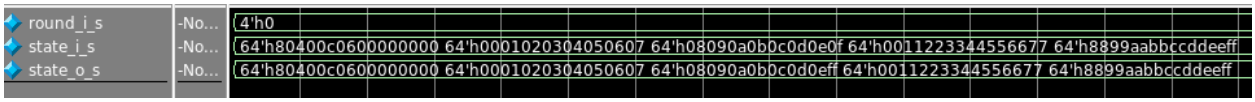

Figure 5: Simulation addition constante

4.1.2 La couche de substitution

Si on interprète l'état courant S comme une matrice 5*64 bits, on peut dire que la couche de substitution s'applique à chaque colonne de cette dernière, en effet à l'aide d'une table de correspondance elle affecte à chaque colonne une nouvelle valeur. Nous avons conçu ce module comme la juxtaposition de 64 sous-modules agissant chacun sur une colonne et travaillant de façon concurrente.

Avec cette approche nous avons privilégié la performance au détriment de l'utilisation de matière première.

Donc en entrée on aura que l'état **Etat_i**(320 bits) que l'on retrouvera modifié en sortie **Etat_o**.

Résultat de la simulation :

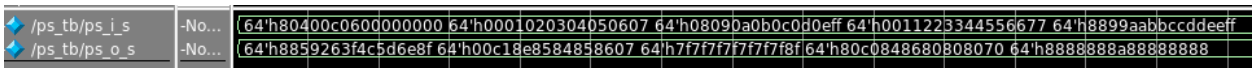

Figure 6: Simulation couche substitution

4.1.3 Couche de diffusion linéaire

Cette opérande applique une diffusion à chaque registre composant l'état courant S en entrée.

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

Figure 7: Diffusion linéaire

Résultat de la simulation:

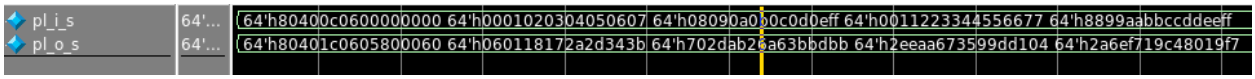

Figure 8: Simulation couche diffusion

4.2 Portes XOR

Pour fonctionner l'algorithme ASCON128 a besoin d'effectuer des opération de type XOR avec l'état courant et soit les données où la clé de chiffrement.

Dans optique d'optimisation du code et d'utilisation des ressources nous avons, en analysant le circuit, identifié deux catégories différentes de XOR: **xor_up** et **xor_down**.

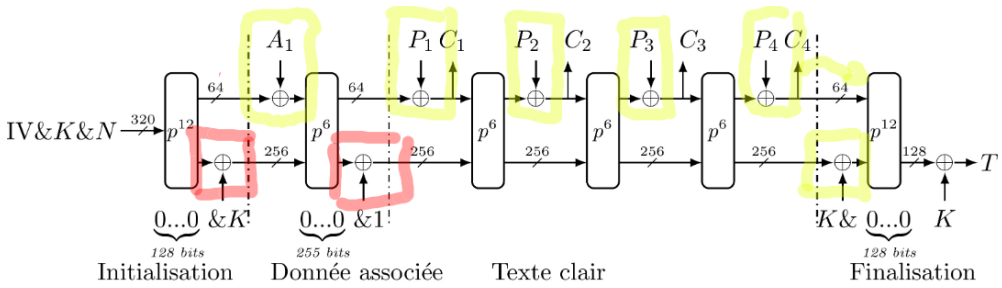

Figure 9: xor_up en jaune et xor_down en rouge

4.2.1 Fonctionnement

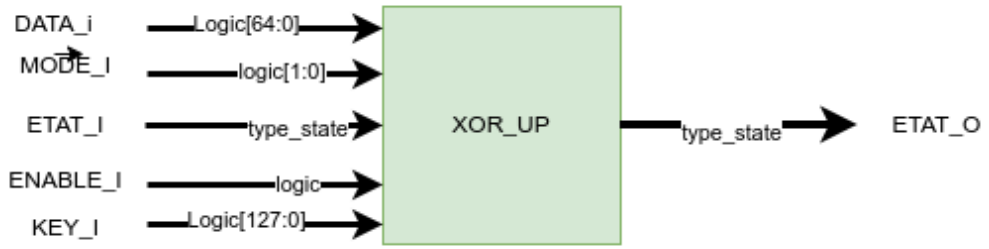

Figure 10: xor_up (begin) se trouvant en amont de la permutation

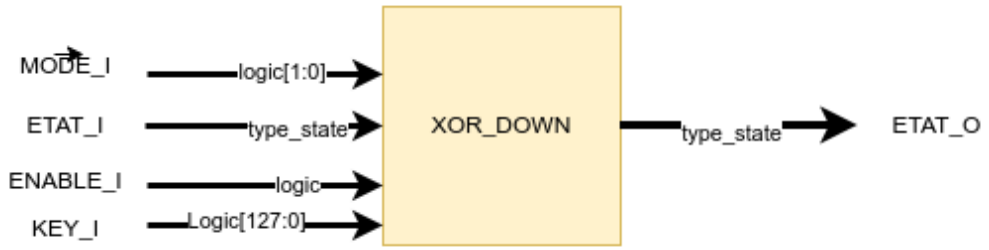

Figure 11: xor_down (end) se trouvant après la permutation

Comme on peut le constater les deux composants sont très similaires : à chaque fois on retrouve un signale **enable_i** pour contrôler le ON/OFF du composant, nous avons à chaque fois en entrée la clé et la données associées qui nous servirons pour réaliser l'opération XOR et enfin un signal **Mode_i** qui permet de sélectionner avec qui l'état S doit faire le XOR.

Cette implémentation des signaux de contrôle nous permet de rendre notre circuit moins gourmand en silicium car on aura besoin de moins de composants.

4.3 Simulation complète

Nous avons reliés tous ces composant dans un même module et nous y avons ajouté des bascules D pour pouvoir sauvegarder l'état courant de notre système et d'éventuelles sorties tel que les messages chiffrés (cipher).

Pour simplifier la phase de testing l'ajout des composants s'est fait de manière progressive, cela nous permet aussi de détecter plus facilement la source d'une erreur.

Uniquementt permutation p Résultat de la simulation :

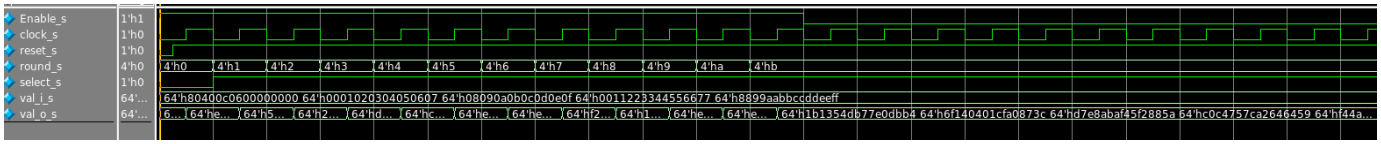

Figure 12: Simulation permutation v1

Ajout du composant xor_up Résultat de la simulation :

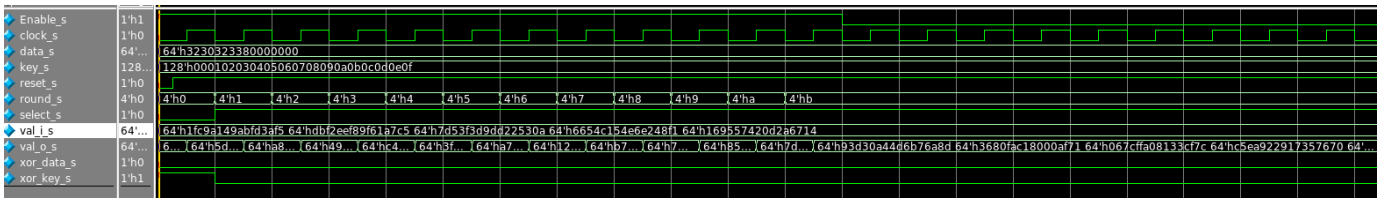

Figure 13: Simulation permutation v2

Ajout du composant xor_down Résultat de la simulation :

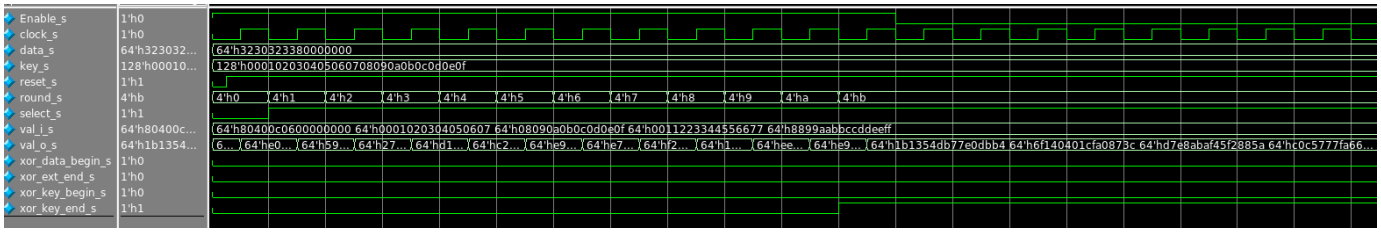

Figure 14: Simulation permutation v3

Ajout des bascules D pour sauvegarder le message chiffré et le tag Résultat de la simulation :

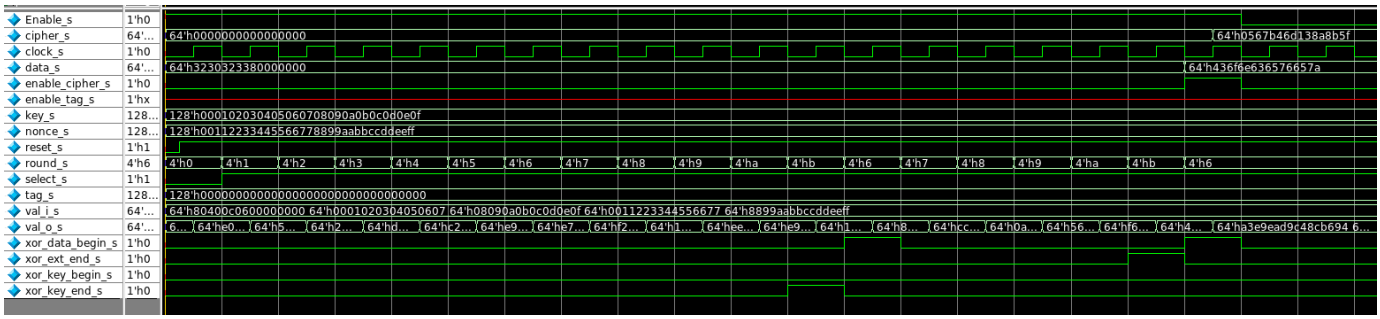

Figure 15: Simulation permutation v4

5 Machine de Moore

Afin de coordonner de manière synchronisée toutes les opérations à effectuer par le bloc "permutation", nous envisageons la création d'une machine de Moore qui détaille les divers états du processus de chiffrement. Pour ce faire, deux indicateurs essentiels seront utilisés : la valeur de la ronde en cours et le rang du bloc de message en traitement c'est pour ça que en plus de la machine, l'utilisation de deux compteurs sera nécessaire pour suivre ces indicateurs.

5.1 Compteurs

Compteur de ronde Ce compteur nous permet d'avoir un suivi sur la ronde actuelle et sera utilisé notamment par la permutation lors de l'étape de l'addition de la constante.

Ce compteur doit pouvoir être contrôlé par la machine d'état c'est pour cela que nous y ajouteront des signaux de ON/OFF et des signaux d'initialisation car nous avons deux cas soit on va de 0 à 11 pour $P12$ ou de 6 à 11 pour $P6$.

Nous avons en entrée le signal **clock_i** celui de reset **reset_i** mais aussi un signal d'activation enable_i**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**.

Compteur de bloc Ce composant n'est pas essentiel au fonctionnement du chiffrement ASCON128, mais son ajout nous permettrait de réduire considérablement le nombre d'états de notre Machine d'état. En effet ce compteur permet d'avoir un suivi sur quelle partie du message nous sommes en train de déchiffrer, il nous suffira donc seulement de répéter les derniers états tant que nous avons pas consommé la totalité du message.

Nous avons en entrée le signal **clock_i** celui de reset **reset_i** mais aussi un signal d'activation enable_i**enable_i** qui contrôle l'allumage du compteur. En sortie nous avons un chiffre sur 2 bits **cpt_o** car le message contient 4 partie de 64 bits.

5.2 Machine d'état

La machine d'état doit pouvoir piloter l'ensemble des composant de notre circuit. Elle sera principalement pilotée par la variable "**start_i**", qui déclenche le début du chiffrement, et la variable "**data_valid_i**", qui indique la validité des données. Cette dernière permet à la machine de progresser, sinon l'algorithme est entièrement interrompu. En outre, la machine d'état se doit de commander les compteurs et de les synchroniser avec les opérations de la permutation.

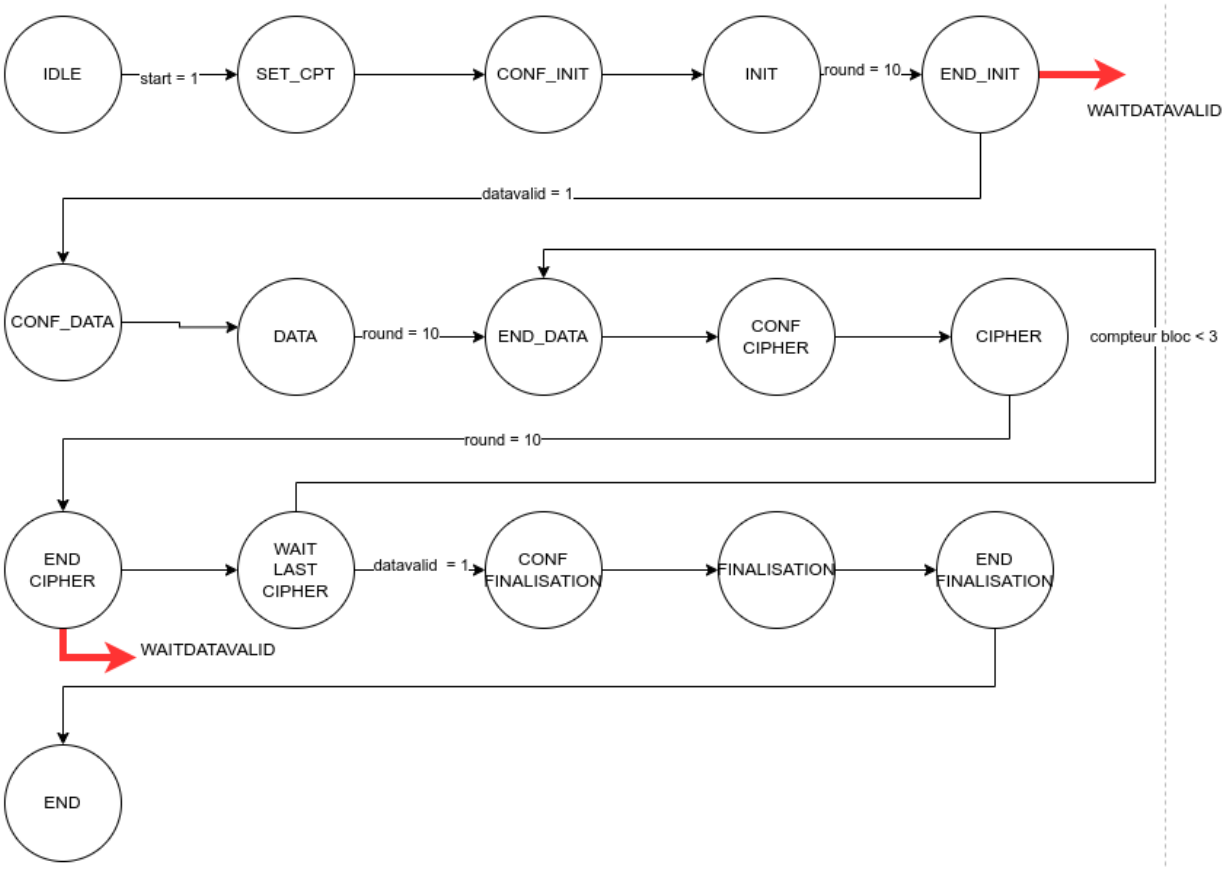

Figure 16: Graphe d'état

5.3 Top level

Pour finir nous avons exécuté l'ensemble du programme en simulant bien tous le branchements entre la FSM et la permutation. Résultats de la simulation :

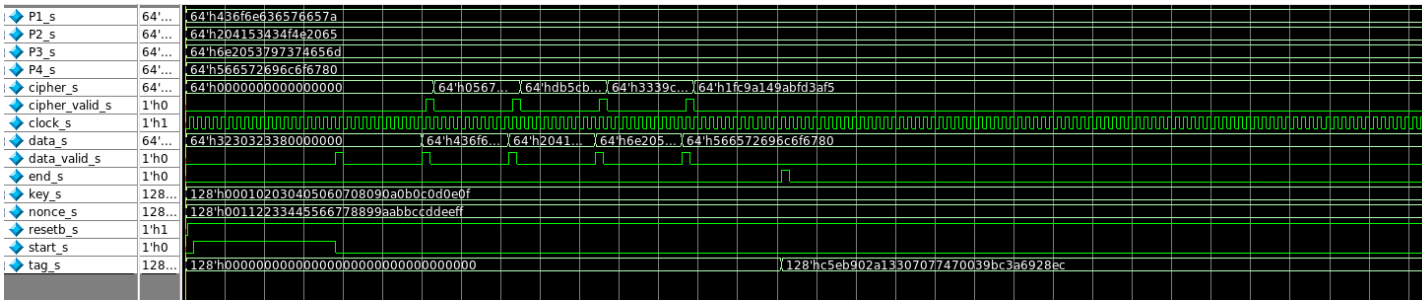

Figure 17: Simulation totale

6 Conclusion

Ce projet a constitué une expérience enrichissante, tant du point de vue de la compréhension et de la mise en œuvre de l'algorithme que de l'apprentissage à travers le langage SystemVerilog. Néanmoins, il faut noter que ce projet s'est révélé exigeant et chronophage, nécessitant un investissement considérable en dehors des heures de cours. Cependant, la progression graduée de la complexité des programmes à mettre en place au fil des cours a joué un rôle significatif et nécessaire en offrant un soutien continu tout au long du projet.