Modélisation en SystemVerilog de l'algorithme ASCON-AEAD128

Farès OSMAN

10/05/2025

Table des matières

1 Introduction – Sécurisation des échanges par chiffrement authentifié

L'échange d'informations dans des environnements interconnectés, comme Internet ou les réseaux embarqués, nécessite des mécanismes cryptographiques robustes. Il ne suffit plus de garantir uniquement la confidentialité des messages, mais aussi leur authenticité et leur intégrité.

C'est dans ce contexte que s'inscrit l'algorithme **ASCON-AEAD128**, un algorithme de chiffrement authentifié avec données associées (AEAD, pour *Authenticated Encryption with Associated Data*). Ce type de chiffrement permet de sécuriser les données sensibles tout en permettant la vérification de leur origine et leur non-modification.

Utilisé notamment dans les protocoles de communication sécurisés (TLS, IPsec, WireGuard), ASCON permet d'assurer que :

Sélectionné dans le cadre de l'initiative CAESAR et retenu comme standard par le NIST pour la cryptographie légère, ASCON s'appuie sur une structure dite *fonction éponge* et des permutations cryptographiques efficaces (p^8 et p^{12}) pour chiffrer et authentifier les messages.

Dans ce rapport, nous présentons la version simplifiée de l'algorithme ASCON-AEAD128, son fonctionnement détaillé, ainsi que sa modélisation matérielle en SystemVerilog.

2 Présentation de l'algorithme ASCON-AEAD128

2.1 Structuration de l'état et notations retenues

2.1.1 Organisation interne de l'état S

L'algorithme ASCON-AEAD128 repose sur un état interne noté S, de taille 320 bits. Cet état est représenté comme une concaténation de cinq registres de 64 bits chacun :

$$S = \{S_0, S_1, S_2, S_3, S_4\}$$

Chaque registre S_i contient 64 bits, et les cinq registres sont indexés du plus faible (S_0) au plus fort (S_4) en termes de poids binaire.

L'état S peut être vu selon deux représentations complémentaires :

— **Vue ligne** : cinq lignes de 64 bits (chaque ligne correspond à un registre).

— **Vue colonne** : 64 colonnes de 5 bits (chaque bit d'une colonne provient d'un registre différent).

Par convention :

— La partie **externe** (notée S_r) correspond aux registres S_0 et S_1, soit les 128 bits de poids faible.

— La partie **interne** (notée S_c) correspond aux registres S_2, S_3 et S_4, soit les 192 bits de poids fort.

Cette séparation logique est essentielle pour distinguer les données visibles du monde extérieur (sortie, injection) de celles utilisées à des fins internes de diffusion et de sécurité.

2.2 Symboles et paramètres du chiffrement

Les notations suivantes seront utilisées tout au long du rapport :
— K : clé secrète de 128 bits, partagée entre l'émetteur et le récepteur.
— N : nonce de 128 bits, différent à chaque message pour garantir l'unicité.
— A : données associées de 96 bits (non chiffrées mais authentifiées).
— P : message clair à chiffrer, codé sur 376 bits (avant padding).
— C : texte chiffré résultant du chiffrement de P, même taille que P.
— T : tag d'authentification de 128 bits généré à la fin de l'algorithme.
— IV : vecteur d'initialisation fixe, codé sur 64 bits. Valeur utilisée :

$$\texttt{IV} = \texttt{0x00001000808c0001}$$

Toutes les opérations arithmétiques internes sont réalisées sur des blocs binaires de taille fixe (64 bits par registre), principalement via XOR, rotation circulaire et substitution.

2.2.1 Logique d'une permutation

Les permutations p^8 et p^{12} sont les transformations centrales de l'algorithme ASCON-AEAD128. Elles permettent de brouiller efficacement l'état interne S grâce à une série de transformations répétées appelées **rondes**.

Chaque permutation consiste en l'application successive d'un certain nombre de rondes :
— p^8 applique 8 rondes : utilisée pour le traitement des données associées et du texte clair.
— p^{12} applique 12 rondes : utilisée pour l'initialisation et la finalisation.
Une ronde est une fonction composée de trois opérations successives sur l'état S :

$$p = p_L \circ p_S \circ p_C$$

où :
— p_C : l'addition de constante.
— p_S : la substitution non-linéaire par S-box.
— p_L : la diffusion linéaire.
L'enchaînement de ces trois opérations permet de garantir à la fois confusion et diffusion sur l'état interne, ce qui est essentiel pour la sécurité cryptographique de l'algorithme.

2.2.2 Étape 1 : injection de constantes (pC)

La première étape d'une ronde consiste à ajouter une constante à l'état S, plus précisément au registre S_2. Cette opération est réalisée via un XOR :

$$S_2 \leftarrow S_2 \oplus \texttt{roundconstant}[i]$$

La constante de ronde dépend de l'indice i de la ronde (entre 0 et 11) et est définie dans le fichier `ascon_pack.sv` sous forme d'un tableau de 64 bits.

Cette étape sert à briser la symétrie et garantir que chaque ronde apporte une transformation unique. Elle joue un rôle fondamental dans la résistance de l'algorithme aux attaques par linéarisation ou réutilisation d'état.

2.2.3 Étape 2 : substitution non-linéaire (pS)

La deuxième transformation appliquée à chaque ronde est une **substitution non-linéaire** sur l'état S, réalisée à l'aide d'une **boîte de substitution** (S-box). Cette opération est notée p_S.

L'état S est vu ici comme un tableau de 64 colonnes de 5 bits (un bit par registre, de S_0 à S_4). Chaque colonne représente un mot de 5 bits, qui est ensuite transformé par la S-box selon une correspondance définie dans un tableau.

La S-box est une fonction :

$$\texttt{Sbox} : \{0,1\}^5 \to \{0,1\}^5$$

et son contenu est défini dans le fichier `ascon_pack.sv`, dans le tableau `sbox`.

L'opération effectuée est :

$$\forall i \in \{0,\ldots,63\}, \quad \{S_0[i], S_1[i], S_2[i], S_3[i], S_4[i]\} \leftarrow \texttt{Sbox}(\{S_0[i], \ldots, S_4[i]\})$$

Cette substitution est essentielle pour introduire de la non-linéarité dans l'algorithme, empêchant les attaques linéaires ou différentielles de prédire le comportement du système.

2.2.4 Étape 3 : diffusion linéaire (pL)

La troisième et dernière transformation d'une ronde est la diffusion linéaire, notée p_L. Elle est appliquée **indépendamment sur chaque ligne** de l'état S, c'est-à-dire sur chacun des registres S_0 à S_4.

L'objectif de cette étape est de propager localement les modifications apportées à certains bits de l'état vers d'autres bits du même registre, augmentant ainsi la diffusion des informations.

Pour chaque registre S_i, une fonction Σ_i est appliquée :

$$\begin{aligned}
\Sigma_0(x) &= x \oplus (x \gg 19) \oplus (x \gg 28) \\
\Sigma_1(x) &= x \oplus (x \gg 61) \oplus (x \gg 39) \\
\Sigma_2(x) &= x \oplus (x \gg 1) \oplus (x \gg 6) \\
\Sigma_3(x) &= x \oplus (x \gg 10) \oplus (x \gg 17) \\
\Sigma_4(x) &= x \oplus (x \gg 7) \oplus (x \gg 41)
\end{aligned}$$

où $(x \gg n)$ désigne une **rotation circulaire** de n bits vers la droite (et non un simple décalage logique).

Chaque registre est donc mis à jour selon sa propre fonction Σ_i :

$$S_i \leftarrow \Sigma_i(S_i)$$

Cette étape améliore significativement la sécurité globale du chiffrement en s'assurant qu'une modification locale d'un bit influence plusieurs bits après quelques rondes.

2.3 Déroulement complet de l'algorithme de chiffrement

L'algorithme ASCON-AEAD128 fonctionne en plusieurs phases successives qui transforment l'état interne S afin de produire un message chiffré C et un tag d'authentification T. Voici les étapes détaillées du processus :

1. **Initialisation**
   — L'état S est initialisé à partir du vecteur IV, de la clé K et du nonce N :

$$S \leftarrow \{IV, K[63:0], K[127:64], N[63:0], N[127:64]\}$$

   — L'état subit une permutation p^{12} :

$$S \leftarrow p^{12}(S)$$

   — Puis on applique un XOR avec la clé :

$$S \leftarrow S \oplus \{0^{192}, K[63:0], K[127:64]\}$$

2. **Traitement des données associées**
   — Les données A sont paddées et concaténées avec l'octet `0x01` pour former un bloc A_1 de 128 bits.
   — Ce bloc est injecté dans l'état par XOR :

$$S_r \leftarrow S_r \oplus A_1$$

   — On applique ensuite une permutation p^8 sur l'état.
   — Enfin, le bit de terminaison est ajouté à S_4 :

$$S_4 \leftarrow S_4 \oplus 0x8000000000000000$$

3. **Chiffrement du texte clair**
   — Le message P est découpé (après padding) en trois blocs P_1, P_2, P_3 de 128 bits chacun.
   — Pour P_1 et P_2 :

$$\begin{aligned} S_r &\leftarrow S_r \oplus P_i \\ C_i &\leftarrow S_r \\ S &\leftarrow p^8(S) \end{aligned}$$

   — Pour P_3 :

$$\begin{aligned} S_r &\leftarrow S_r \oplus P_3 \\ C_3 &\leftarrow S_r[119:0] \quad \text{(sur 15 octets)} \end{aligned}$$

4. **Finalisation**
   — On applique un XOR avec la clé partielle :

$$S \leftarrow S \oplus \{0^{64}, K[63:0], K[127:64], 0^{128}\}$$

   — Une permutation p^{12} est appliquée :

$$S \leftarrow p^{12}(S)$$

— Le tag T est calculé par XOR entre les deux derniers registres et la clé :

$$T \leftarrow \{S_4, S_3\} \oplus \{K[127:64], K[63:0]\}$$

À l'issue du chiffrement, les éléments transmis au destinataire sont :
— Les données associées A
— Le texte chiffré $C = \{C_1, C_2, C_3\}$
— Le tag d'authentification T
— Le nonce N

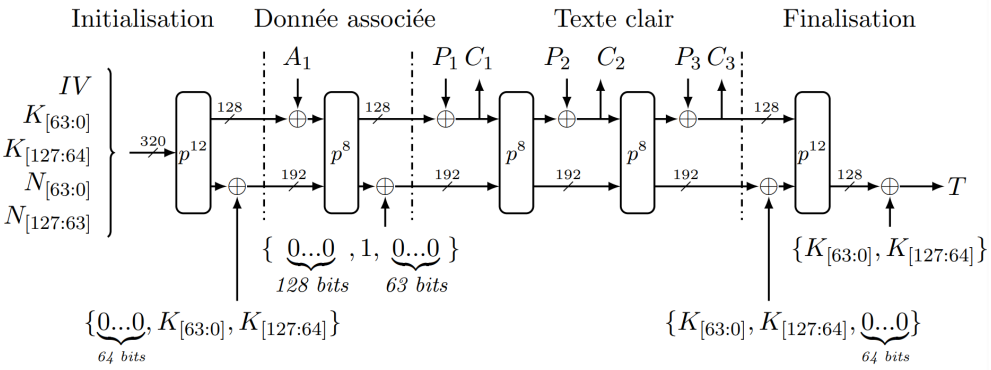

FIGURE 1 – Schéma du chiffrement Ascon-AEAD128

3 Modélisation numérique de l'algorithme

3.1 Cadre de la machine à états (FSM)

La machine à états finis (FSM — Finite State Machine) joue un rôle central dans la gestion de l'algorithme ASCON-AEAD128. Elle est responsable de l'enchaînement cohérent des différentes phases : initialisation, traitement des données associées, chiffrement du texte clair, et finalisation.

La FSM adoptée est de type **Moore**, c'est-à-dire que ses sorties (signaux de contrôle) dépendent uniquement de l'état courant, ce qui permet une meilleure stabilité des signaux.

Objectifs de la FSM :
— Contrôler les signaux de sélection de données (clé, nonce, données associées, texte clair).
— Gérer les phases successives de l'algorithme.
— Piloter les modules de permutation et les compteurs.
— Activer les modules d'enregistrement pour le texte chiffré et le tag.
— Générer les signaux de fin de chiffrement et de validité des sorties.

Chaque phase logique de l'algorithme est modélisée par un ou plusieurs états. Des transitions conditionnelles sont utilisées pour attendre la fin des opérations comme les permutations ou la disponibilité des données en entrée.

3.2 Vue d'ensemble de l'architecture matérielle

L'implémentation matérielle repose sur une architecture modulaire appelée Ascon Top. Elle est conçue pour respecter le découpage logique de l'algorithme. L'architecture globale comprend :

— **FSM** : contrôle l'ensemble du système en fonction des signaux d'entrée et des compteurs.
— **Module de permutation** : applique les transformations p_C, p_S et p_L en boucle, selon le nombre de rondes demandé.
— **Blocs XOR** : injectent les données nécessaires (clé, nonce, données associées, texte clair) au bon moment, et extraient le tag à la fin.
— **Registres de stockage** : mémorisent l'état S, le texte chiffré C, et le tag T.
— **Multiplexeur** : permet de sélectionner les entrées actives selon l'étape de l'algorithme.
— **Compteurs** : gèrent le nombre de rondes pour les permutations et le nombre de blocs à chiffrer.

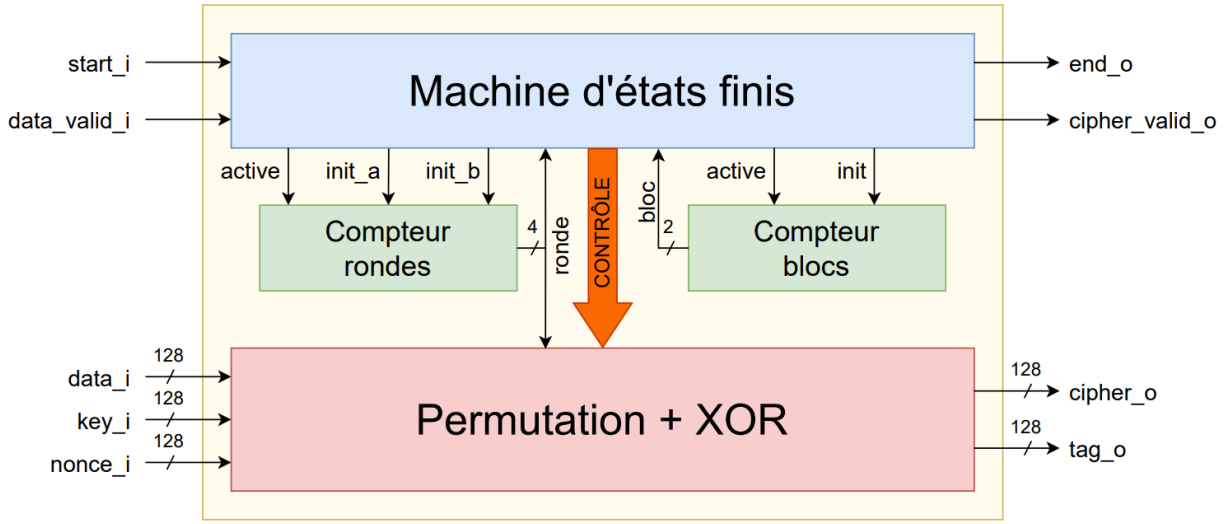

FIGURE 2 – Architecture globale de Ascon-AEAD128

Cette architecture permet de traiter l'ensemble du chiffrement ASCON-AEAD128 en flux contrôlé, tout en facilitant le test des sous-composants individuellement.

3.3 Permutation finale

3.3.1 Principe de fonctionnement

Le module `permutation_finale` est le cœur du traitement cryptographique. Il encapsule l'ensemble des opérations nécessaires à une ronde de permutation de l'état S, ainsi que les blocs périphériques permettant d'injecter ou d'extraire les données pertinentes.

Ce module regroupe les sous-modules suivants :

— **mux** : un multiplexeur qui sélectionne, selon l'étape de l'algorithme, l'état initial ou l'état présent (boucle).
— **xor_begin** : applique un XOR entre les données d'entrée sélectionnées par le multiplexeur et l'état courant.
— **addition_constante (pC)** : injecte une constante de ronde dans le registre S_2, selon l'itération en cours.

— **couche_substitution (pS)** : applique une transformation non-linéaire à chaque colonne de l'état via une S-box 5 bits.
— **couche_diffusion (pL)** : effectue des rotations cycliques et des XOR sur chaque ligne pour diffuser l'information.
— **xor_end** : réalise un XOR final avec la clé afin de générer le tag d'authentification.
— **Registres internes** :
  — `regstate` : enregistre l'état S à la fin de chaque ronde.
  — `reg_tag` et `reg_cipher` : stockent respectivement le tag final et les blocs de texte chiffré.

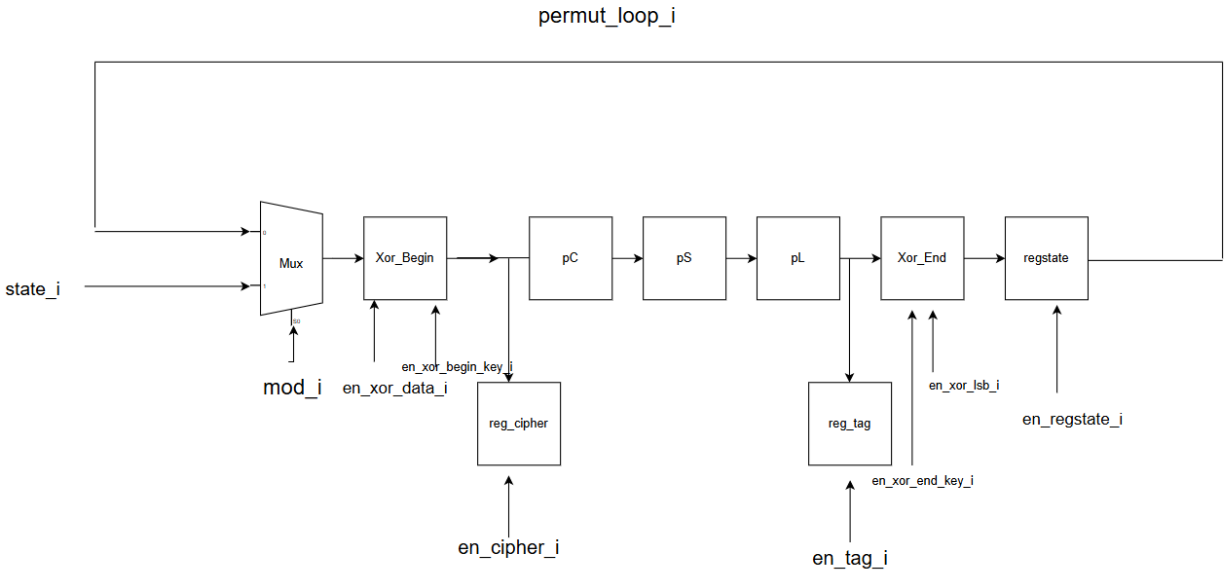

FIGURE 3 – Architecture permutation finale

3.3.2 Validation des sous-modules

Chaque sous-module du bloc `permutation_finale` a été validé indépendamment à l'aide de testbenchs dédiés. Ces simulations permettent de vérifier le bon fonctionnement logique de chaque étape composant la permutation.

Testbench du module `addition_constante` (pC) On vérifie ici que la constante est correctement XORée à S_2 selon la ronde en cours.

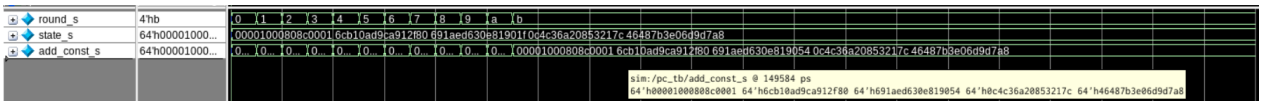

FIGURE 4 – Chronogramme de la simulation du testbench de l'addition de constante

Testbench du module `couche_substitution` (pS) Ce test valide la substitution de chaque colonne de l'état via la S-box .

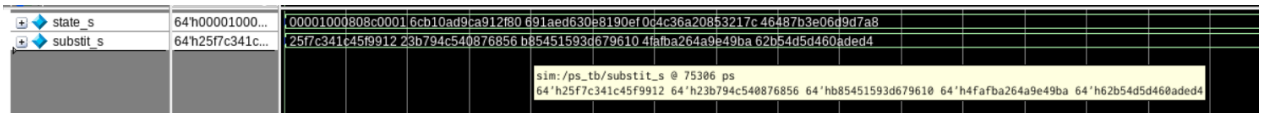

FIGURE 5 – Chronogramme de la simulation du testbench de l'addition de constante

Testbench du module `couche_diffusion` (pL) La simulation vérifie que chaque registre S_i subit correctement la fonction de diffusion Σ_i.

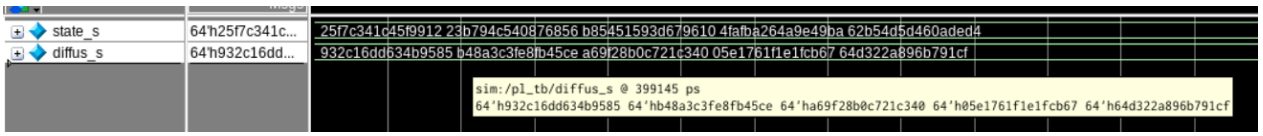

FIGURE 6 – Chronogramme de la simulation du testbench de l'addition de constante

Testbench du module `xor_end` Ce module réalise le dernier XOR de la boucle

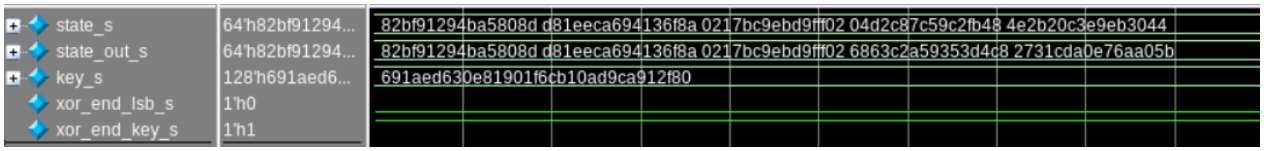

FIGURE 7 – Chronogramme de la simulation du testbench du xor_end

Testbench du module `permutation_finale` Une simulation complète a été réalisée pour tester l'enchaînement de plusieurs permutations p et . Les sorties du module tag_s et cipher_s sont conformes aux attentes.

3.4 Machine à états finie de contrôle (FSM)

3.4.1 Entrées et sorties de la FSM

La FSM contrôle le déroulement complet de l'algorithme ASCON-AEAD128, en activant les bons modules au bon moment via des signaux de commande. Elle utilise les signaux suivants :

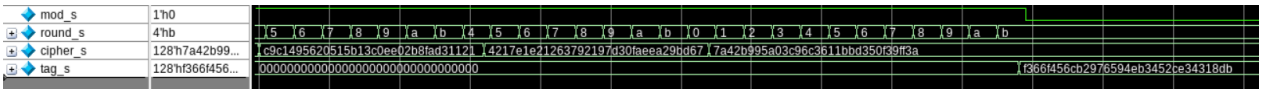

FIGURE 8 – Chronogramme de la simulation du testbench de la permutation finale

Entrées

- `start_i` : signal de démarrage de l'algorithme (1 bit)
- `clock_i` : horloge du système (1 bit)
- `resetb_i` : réinitialisation asynchrone active à 0 (1 bit)
- `round_i` : numéro de la ronde actuelle (4 bits)
- `data_valid_i` : indique qu'une donnée est disponible à l'entrée (1 bit)

Sorties

- `end_o` : indique la fin complète de l'algorithme
- `cipher_valid_o` : indique que la sortie `cipher_o` est valide
- `end_init_o` : fin de la phase d'initialisation
- `end_associate_o` : fin du traitement des données associées
- `end_cipher_o` : fin du chiffrement du texte clair
- `en_cpt_o` : active le compteur de rondes
- `init_a_o` : réinitialise le compteur de rondes à 0
- `init_b_o` : réinitialise le compteur à 4 (pour p8)
- `en_xor_begin_key_o` : active le XOR d'initialisation avec la clé
- `en_xor_data_o` : active le XOR pour injecter A ou P_i
- `en_xor_end_key_o` : active le XOR final pour calculer le tag
- `en_xor_lsb_o` : active l'ajout de 0x8000000000000000 dans S_4
- `mod_o` : sélectionne le mode de permutation (p8 ou p12)
- `en_tag_o` : active l'enregistrement du tag
- `en_cipher_o` : active l'enregistrement du bloc chiffré
- `en_reg_state_o` : active la mise à jour du registre de l'état S

L'ensemble de ces signaux permet de piloter les modules internes de façon fine, en fonction de la progression de l'algorithme dans ses différentes phases.

3.4.2 Diagramme d'état de la FSM

La machine à états finie (FSM) modélise l'enchaînement logique des différentes étapes de l'algorithme ASCON-AEAD128. Elle permet de synchroniser les opérations de permutation, les lectures et écritures dans les registres, ainsi que l'activation des blocs XOR selon la phase en cours.

Chaque état correspond à une phase précise du chiffrement :

- **Initialisation** : chargement du vecteur IV, de la clé K et du nonce N, suivi d'une permutation p^{12}.
- **Données associées** : injection du bloc A_1, permutation p^8, et ajout du bit de terminaison dans S_4.
- **Texte clair** : traitement successif des blocs P_1, P_2, P_3, avec permutations intermédiaires.
- **Finalisation** : XOR avec la clé, permutation p^{12} finale, et calcul du tag T.

Les transitions entre états dépendent de signaux internes (comme `round_i`) et externes (comme `data_valid_i`, `start_i`). La FSM produit à chaque état les signaux nécessaires à l'activation des blocs concernés.

Le diagramme d'état ci-dessous illustre la séquence exacte du déroulement logique de la FSM.

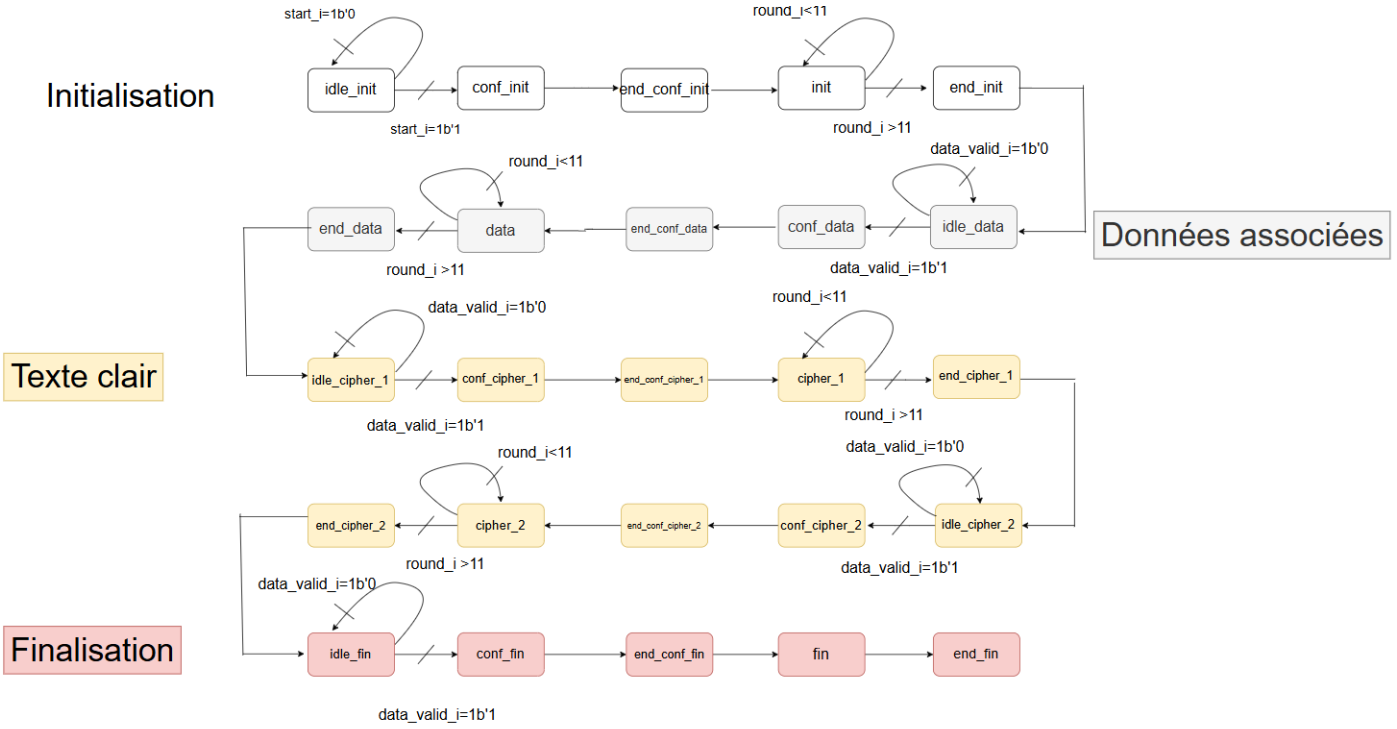

FIGURE 9 – Diagramme d'état de la machine à états finis

3.5 Module principal `Ascon_Top`

3.5.1 Principe de fonctionnement

Le module `Ascon_Top` constitue l'entité principale du système. Il encapsule l'ensemble des composants précédents : la machine à états finis (FSM), le module `permutation_finale`, les compteurs, ainsi que les registres et blocs de contrôle nécessaires.

Son rôle est d'exécuter le chiffrement authentifié ASCON-AEAD128 de bout en bout, à partir des données d'entrée (clé, nonce, données associées et texte clair) et de produire en sortie le texte chiffré et le tag.

L'enchaînement des opérations est entièrement automatisé par la FSM, sans intervention extérieure une fois le signal `start_i` activé.

3.5.2 Interface d'entrée / sortie du module `Ascon_Top`

Entrées

- `clock_i` : horloge principale du système
- `resetb_i` : réinitialisation globale (active à 0)
- `start_i` : déclenchement du chiffrement
- `data_i` : entrée de données sur 128 bits (données associées puis texte clair)
- `data_valid_i` : signale que `data_i` est prêt à être lu
- `key_i` : clé de chiffrement (128 bits)
- `nonce_i` : nonce de 128 bits

Sorties

- `cipher_o` : bloc de texte chiffré (128 bits)
- `cipher_valid_o` : indique que `cipher_o` est valide
- `tag_o` : tag d'authentification final (128 bits)
- `end_initialisation_o` : signal de fin de l'initialisation
- `end_associate_o` : signal de fin de données associées
- `end_cipher_o` : signal de fin du texte clair
- `end_o` : signal de fin du chiffrement (valide la sortie `tag_o`)

3.5.3 Résultat de la simulation du testbench

Le module a été validé à l'aide du testbench `ascon_top_tb`, qui simule un cas complet de chiffrement.

Les résultats de simulation montrent que :

- Les trois blocs de texte chiffré C_1, C_2 et C_3 sont bien générés séquentiellement sur `cipher_s`.
- Le tag d'authentification T est produit correctement sur `tag_s`.
- Les signaux `cipher_valid_o` et `end_s` s'activent exactement aux bons instants.

Ces résultats confirment que l'implémentation complète respecte la logique fonctionnelle de l'algorithme ASCON-AEAD128.

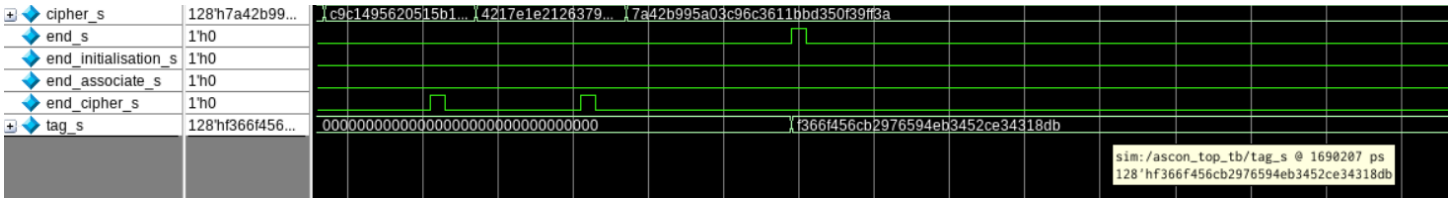

FIGURE 10 – Chronogramme de la simulation du testbench du module final

4 Difficultés rencontrées

Comme tout projet de conception numérique complexe, l'implémentation matérielle de l'algorithme ASCON-AEAD128 a soulevé plusieurs difficultés, aussi bien techniques que méthodologiques.

4.1 Problèmes rencontrés lors de la permutation finale

La partie la plus sensible du projet a sans doute été le développement du module `permutation_finale`. Malgré une architecture initialement cohérente, plusieurs anomalies ont été détectées au moment des premières simulations.

Après investigation, il s'est avéré que les erreurs provenaient principalement de deux sous-modules :

- **La S-box** contenue dans la `couche_substitution` ne substituait pas correctement les colonnes de l'état. Cela entraînait des sorties erronées dès la première ronde.

— **Le bloc** `xor_begin` posait également problème : l'injection des données dans l'état S ne se faisait pas correctement, ce qui biaisait l'ensemble des opérations suivantes.

Face à ces dysfonctionnements, il a été décidé de **reprendre la permutation finale depuis le début**, en reconstruisant et testant **chaque bloc indépendamment**. Cette démarche méthodique a permis de rétablir un comportement conforme à l'algorithme spécifié.

4.2 Difficultés liées à la machine à états finis (FSM)

Un autre point critique a été le développement de la FSM. Si les transitions générales ont pu être mises en place de manière structurée, la gestion fine des signaux de contrôle, en particulier pour la **phase d'initialisation**, s'est révélée particulièrement complexe.

Conclusion

Ce projet de modélisation matérielle de l'algorithme ASCON-AEAD128 en SystemVerilog a été l'occasion d'aborder de manière concrète la conception d'un système numérique sécurisé, en partant d'un algorithme cryptographique standardisé.

L'approche adoptée, fondée sur une architecture modulaire, a permis de structurer le développement de manière progressive : des blocs élémentaires (XOR, S-box, diffusion) jusqu'à l'intégration complète dans le module `ascon_top`. L'utilisation d'une machine à états finis pour piloter les opérations a également permis de mieux maîtriser l'enchaînement des phases de l'algorithme.

Malgré plusieurs difficultés rencontrées, notamment au niveau de la permutation finale et de la FSM, la validation par simulations a montré que l'implémentation est fonctionnelle : les blocs de texte chiffré et le tag d'authentification obtenus sont conformes aux attentes.

Ce projet m'a permis de renforcer mes compétences en logique, en architecture de systèmes embarqués et en conception SystemVerilog. Il m'a aussi permis d'expérimenter une méthodologie rigoureuse de test, essentielle pour garantir la fiabilité d'un chiffrement matériel.