MODELISATION SYSTEMVERILOG DE L'ALGORITHME DE CHIFFREMENT ASCON

CONCEPTION DE SYSTÈME NUMÉRIQUE

SEMESTRE 6

Auteur :
Youssef Ennouri

Professeur :

Jean-Max Dutertre

Date :

Mai 2024

SOMMAIRE

INTRODUCTION

PRÉSENTATION DU PROJET

ASCON 128 est un algorithme de chiffrement authentifié avec données associées qui est à la fois léger, rapide et facile à mettre en œuvre. Cet algorithme permet d'assurer la confidentialité des messages, mais également l'authenticité de ceux-ci à l'aide d'un tag qui vérifie que le message reçu est bien celui envoyé.

Ce projet consiste en la modélisation en SystemVerilog de l'algorithme de chiffrement ASCON. Il est intéressant d'utiliser un langage de description matériel comme le SystemVerilog dans le cadre de notre projet afin de concevoir un circuit dédié, car celui-ci rendra l'algorithme plus performant, mais également permettra de libérer des ressources qui pourront être utilisées ailleurs.

Dans le cadre de notre projet, nous utiliserons le logiciel ModelSim de Mentor Graphics pour simuler nos différents composants et notre circuit final.

Le projet sera composé du présent rapport, mais également d'une archive au format .zip contenant les différents codes .sv et le script de compilation de ces codes.

HISTORIQUE DE L'ALGORITHME ASCON 128

Ascon a été développé en 2014 par une équipe de chercheurs pluridisciplinaire et a terminé finaliste de la CAESAR Competition, un concours organisé par un groupe de chercheurs internationaux en cryptologie pour encourager la conception de systèmes de cryptage authentifiés.

En 2023, l'institut national des normes et de la technologie (NIST) américaine a sélectionné l'ensemble d'algorithmes Ascon comme norme cryptographique légère. Cette sélection a été basée sur plusieurs critères, notamment la capacité de l'algorithme à fournir une sécurité adéquate, ainsi que sa performance et sa flexibilité en termes de vitesse, de taille et de consommation d'énergie.

BIBLIOTHÈQUES UTILISÉES

Pour ce projet, plusieurs fichiers nous ont été fournis pour faciliter la mise en œuvre du projet et réduire les contraintes temporelles. Ces fichiers sont les suivants :

- ***ascon_pack.sv :*** Ce fichier contient la déclaration d'un tableau de 5 lignes de 64 bits ainsi que d'autres tableaux qui nous seront utiles dans les différents modules de ce projet.
- ***mux_state.sv :*** Ce fichier modélise le comportement d'un multiplexeur à 2 entrées.
- ***register_w_en.sv :*** Cet algorithme représente un registre qui va nous permettre de stocker temporairement des valeurs.
- ***state_register_w_en.sv :*** Ce module représente un registre comme ***register_w_en.sv*** à l'exception que les entrées et sorties ont un type différent.
- ***xor_begin_perm.sv et xor_end_perm.sv :*** Ces fichiers permettent d'effectuer une opération de type XOR sur différents registres de l'état courant et ont des conditions d'activation différentes.
- ***compteur_simple_init.sv et compteur_double_init.sv :*** Ces modules permettent d'implémenter des compteurs pour compter le nombre de rondes. Le compteur double peut être réinitialiser à 6.

ASCON 128

FONCTIONNEMENT DE L'ALGORITHME ASCON 128

L'algorithme Ascon que nous allons étudier dans ce projet est une version simplifiée de l'algorithme original, afin d'être réalisable dans la durée du projet. La description suivante tient compte de ces simplifications.

Dans notre version de l'algorithme, les données associées sont codées sur 48 bits soit 64 bits après le padding. Le texte en clair sera codé sur 184 bits et la clé de chiffrement sur 128 bits. L'algorithme opérera sur un état courant (ou state S) de 320 bits. Cet état sera mis à jour avec une opération appelée permutation. La permutation comprend soit 6 itérations, soit 12 itérations, et sera notée p^6 (respectivement p^{12}).

L'état S de 320 bits est divisé en 5 registres x_i de 64 bits chacun :

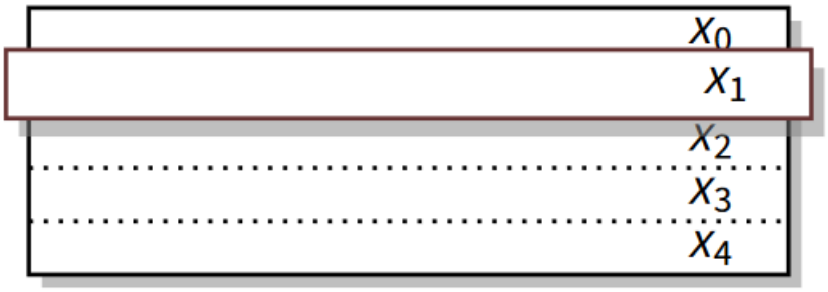

Figure 1 – Représentation en 5 registres de 64 bits

L'état peut aussi être interprété comme 64 colonnes de 5 bits chacune :

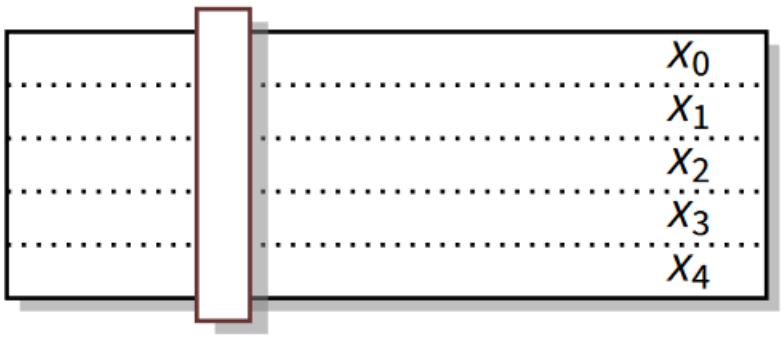

Figure 2 - Représentation en 64 colonnes de 5 bits

Le processus de chiffrement se décompose en quatre étapes :

1. La phase d'initialisation de l'état S en fonction d'un vecteur d'initialisation IV donné, de la clé secrète K et du nombre arbitraire N

2. Le traitement des données associées A, consistant à injecter un bloc de données de 64 bits

3. Le traitement du texte en clair P, par blocs de données de 64 bits, avec l'injection des blocs de texte en clair dans l'état courant, ainsi que la récupération des blocs de texte chiffré

4. La finalisation pour récupérer le tag T utilisé pour l'authentification

Après chaque bloc injecté dans l'état S (à l'exception du dernier bloc de texte clair), la permutation p^6 est appliqué à l'état S. Pendant les phases d'initialisation et de finalisation, la permutation p^{12} est utilisée, contenant 12 rondes au lieu de 6.

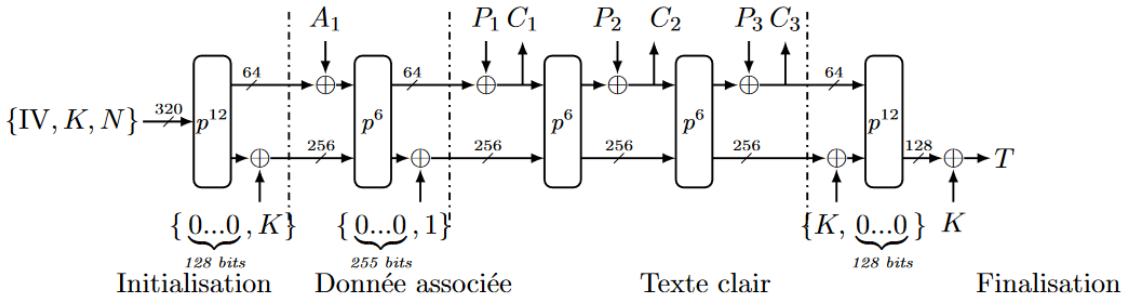

Figure 3 - Schéma du chiffrement ASCON 128

Ainsi, notre circuit Ascon présentera plusieurs entrées et sorties :

- 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 sur ***data_i***
- 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

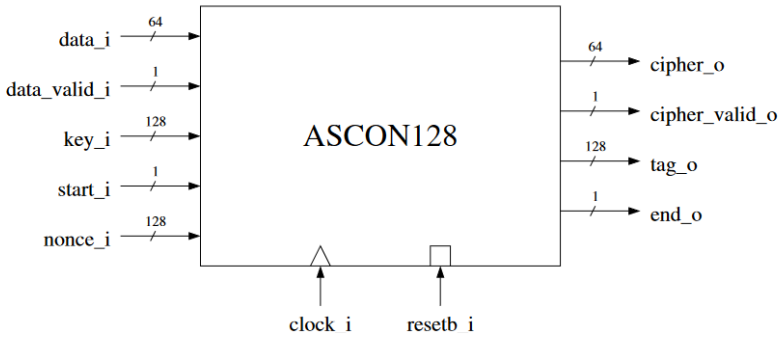

Figure 4 - Entrées/Sorties du module ASCON 128

MODÉLISATION DES FONCTIONS ÉLEMENTAIRES

Addition de Constante

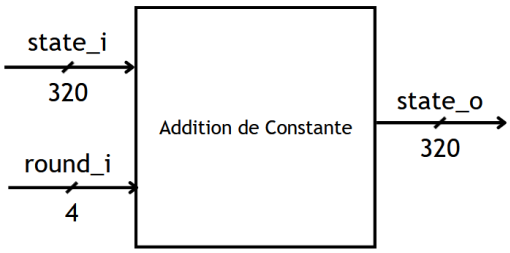

Figure 5 - Module Addition de Constante

L'addition de constante est une étape importante de notre modélisation car elle est la première phase de notre permutation. Elle consiste principalement en l'ajout d'une constante de ronde c_r au registre x_2 de l'état courant S au cours de la ronde i. Il suffit alors d'effectuer une opération de type XOR ($x_2 = x_2 \oplus c_r$) pour déterminer notre nouvelle valeur de x_2. Les constantes sont définies dans le tableau suivant :

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	0000000000000000096
7	1	0000000000000000087
8	2	0000000000000000078
9	3	0000000000000000069
10	4	000000000000000005a
11	5	000000000000000004b

Figure 6 - Liste des constantes pour p6 et p12

Nous utilisons dans notre implémentation le tableau *round_constant* qui est un tableau contenu dans le fichier ***ascon_pack*** et qui nous renvoie la bonne valeur de constante selon le numéro de ronde.

Les registres 0,1,3 et 4 de l'état d'entrée *state_i* ne sont pas modifiés et sont directement assignés la sortie *state_o*. L'octet de poids faible du registre 2 est l'unique entité modifiée dans ce module. L'octet de poids faible du registre 2 de l'état de sortie est calculé en effectuant un XOR (ou exclusif) entre les bits de l'octet de poids faible de l'état d'entrée et une constante *round_constant* déterminé par *round_i*.

Couche de Substitution

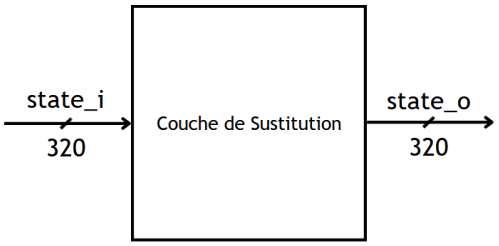

Figure 7 - Module Couche de Substitution

La couche de substitution est la 2ème étape de la permutation. Elle modifie alors l'état courant S en modifiant en parallèle 5 bits à l'aide d'une table de substitution. L'état courant peut alors être vu comme 64 colonnes de 5 bits.

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 8 - Table de Substitution

Pour effectuer la substitution sur chaque colonne, nous allons implémenter un module intermédiaire ***ascon_sbox*** qui prend en entrée 5 bits et effectue la substitution.

Tout comme pour l'addition de constante, nous utilisons *sbox_t* un tableau présent dans ***ascon_pack*** qui contient les valeurs de substitution.

Pour créer notre couche de substitution, il suffit alors de parcourir notre état d'entrée *state_i* à l'aide d'une boucle. À chaque itération, la *sbox* substitue les bits de la ligne correspondante par les nouvelles valeurs. La sortie de la *sbox* est alors stockée dans l'état de sortie *state_o*.

Couche de Diffusion Linéaire

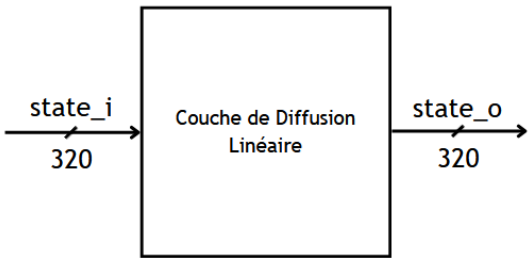

Figure 9 - Module Couche de Diffusion Linéaire

La couche de diffusion linéaire est la 3ème et dernière étape de la permutation. Elle modifie l'état courant S en appliquant une diffusion à chacune des lignes de 64 bits ou registres x_i. Elle modifie chacune des lignes du registre d'entrée en décalant les valeurs à l'aide d'une rotation cyclique de chaque ligne.

L'opération suivante est appliquée :

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

Figure 10 - Opération de Diffusion Linéaire

Chaque registre de l'état d'entrée *diffusion_i* est soumis à une série de rotation cyclique suivie d'une opération de type XOR. Ces opérations sont effectuées de manière individuelle pour chaque registre.

Le résultat de chaque modification est ensuite stocké dans l'état de sortie *diffusion_o*.

Simulation et Validation

Pour valider le bon fonctionnement de chacun de nos modules, il est essentiel de les tester et de vérifier leur validité. Pour cela, nous avons créé des testbenchs qui nous permettent de leur indiquer les différentes valeurs d'entrée de notre système et nous fournit les sorties.

Pour chaque module, les différentes valeurs initiales et finales nous ont été fournies, il est donc aisé de vérifier le bon fonctionnement des modules.

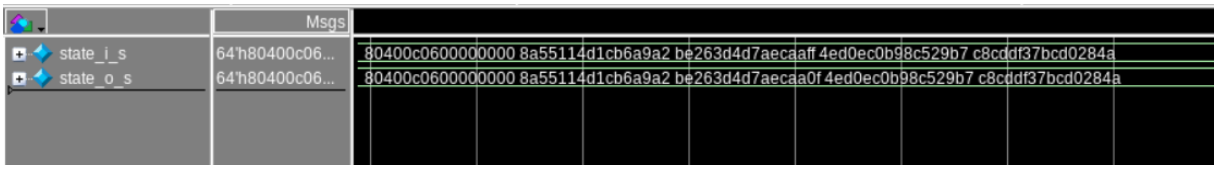

Figure 11 - Simulation du module Constant_Addition

Le résultat attendu et le résultat final sont les mêmes, le module est validé.

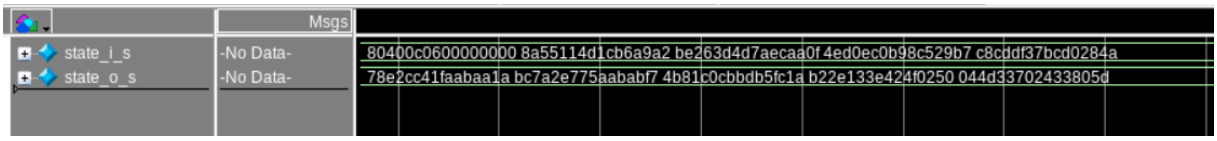

Figure 12 - Simulation du module Substitution_Layer

Le résultat attendu et le résultat final sont les mêmes, le module est validé.

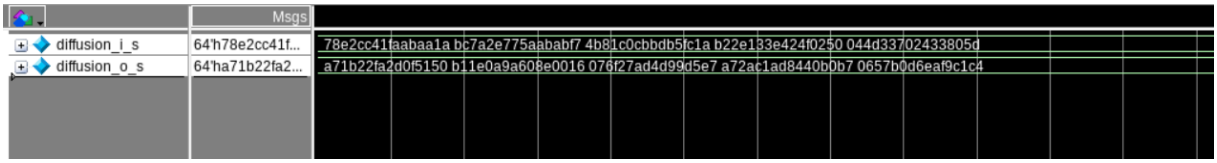

Figure 13 - Simulation du module Diffusion_Layer

Le résultat attendu et le résultat final sont les mêmes, le module est validé.

MODÉLISATION DES PERMUTATIONS P6 ET P12

La permutation est la composante principale de l'algorithme Ascon 128. Cette permutation consiste en l'application consécutive des différentes couches que nous avons implémentées précédemment.

$$p = p_L \, o \, p_s \, o \, p_c$$

Où p_L est la couche de diffusion linéaire, p_c l'addition de constante et p_s la couche de substitution.

Par soucis de simplification, nous allons effectuer la permutation en 2 étapes. Nous allons d'abord créer un module permutation_step_1 qui nous a permis de créer une première permutation simplifiée. Nous avons par la suite implémenté la permutation finale en y ajoutant les opérations XOR et les registres qui renvoient les valeurs de cipher et tag représenté sur la figure ci-dessous.

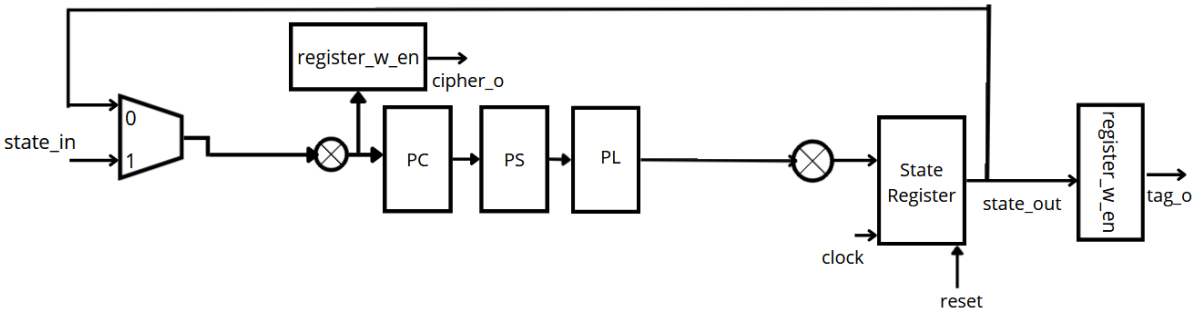

Figure 14 - Modélisation de la permutation finale avec xor

Le fonctionnement de la permutation est assez simple, lors de la 1ère ronde on sélectionne la valeur en entrée du multiplexeur et lui applique successivement différentes opérations comme 2 opérations de type XOR mais également les opérations que nous avons développer plus tôt c'est-à-dire la constante d'addition, la couche de substitution et de diffusion. La valeur finale est ensuite réinjectée dans le même circuit jusqu'à atteindre 6 ou 12 rondes.

L'implémentation en SystemVerilog n'est pas compliquée également. En effet, le module permutation repose sur l'ensemble des modules précédemment crée et ceux fournis. Il suffit alors d'instancier les différents modules et de les faire communiquer à l'aide de variables internes dans lesquels s'échangeront les données entre chaque bloc. Ainsi, par exemple, la variable de sortie du multiplexeur sera également la variable d'entrée du XOR qui le suit pour recréer à l'identique la permutation souhaitée.

Comme pour les modules précédents, il est nécessaire de vérifier la validité de notre module de permutation pour garantir sa fiabilité. Il est donc nécessaire de concevoir également un testbench pour pouvoir le tester.

Figure 15 - Simulation du module Permutation

Le résultat attendu et le résultat final sont les mêmes, le module est validé.

MODÉLISATION DE LA MACHINE D'ÉTAT

La machine de Moore est la pièce centrale de l'algorithme Ascon 128. Cette machine d'état nous permet de contrôler tous les signaux de notre algorithme et est nécessaire à son fonctionnement.

La machine d'état doit être capable de contrôler l'ensemble des signaux d'entrée, de la permutation et du compteur double pour pouvoir mener à bien les 4 phases de l'algorithme.

J'ai fait le choix d'utiliser seulement le compteur double fourni et de ne pas utiliser le compteur de bloc. Cela va certes rallonger l'écriture de mon module FSM qui aura plus d'états mais il me permet néanmoins de simplifier la logique générale de mon code.

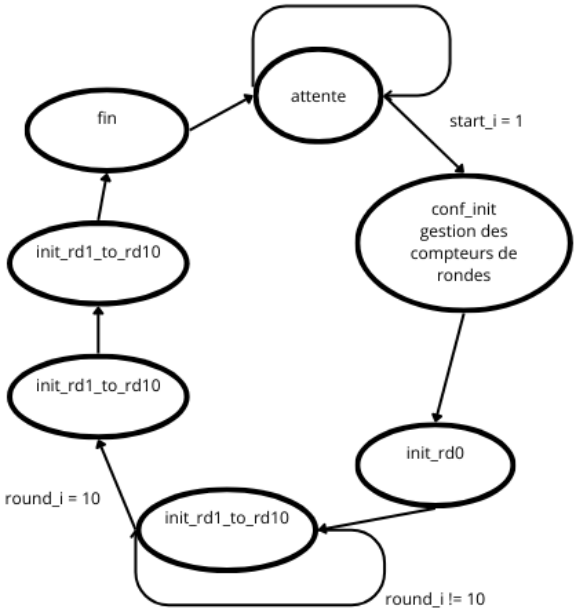

Figure 16 - Diagramme d'état de la phase d'initialisation

Nous pouvons généraliser le diagramme d'état ci-dessus à l'ensemble des phases de l'algorithme ASCON 128. Il est nécessaire de rajouter entre chaque phase des états d'attentes qui nous permettent de valider nos données et de démarrer la nouvelle phase. Il faut simplement faire attention à bien initialiser le compteur à 6 ou 12 rondes quand cela est nécessaire.

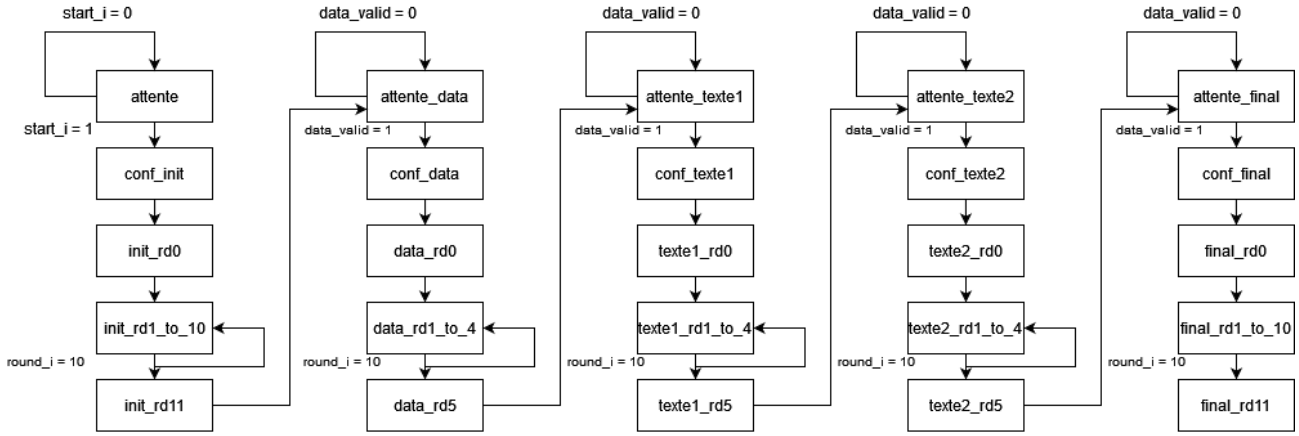

Figure 17 - Diagramme d'état de la machine d'états

Le code de la FSM est décomposé en 2 grandes parties, une première partie traite de la transition des états et la seconde de la transition des valeurs de sortie.

Cette première partie nous permet pour chaque état de définir l'état futur et de mettre une condition au passage de cet état vers l'état futur. Ainsi, dans les états d'attente, on vérifie d'abord que les données sont valides avant de passer à l'état suivant.

La 2ème partie de la FSM permet de gérer les valeurs de sorties des variables à chaque état. On peut alors indiquer à nos autres modules si tel variable doit être égale à 1 à tel état ou non. J'ai modélisé ma FSM de sorte que chaque variable soit réinitialisée à 0 après chaque état. Il ne me suffit donc à chaque état de passer les variables à 1 lorsqu'elles doivent être activées.

Enfin, il nous reste à vérifier que nos signaux s'activent bien quand on le souhaite, ce qui est bien le cas ici.

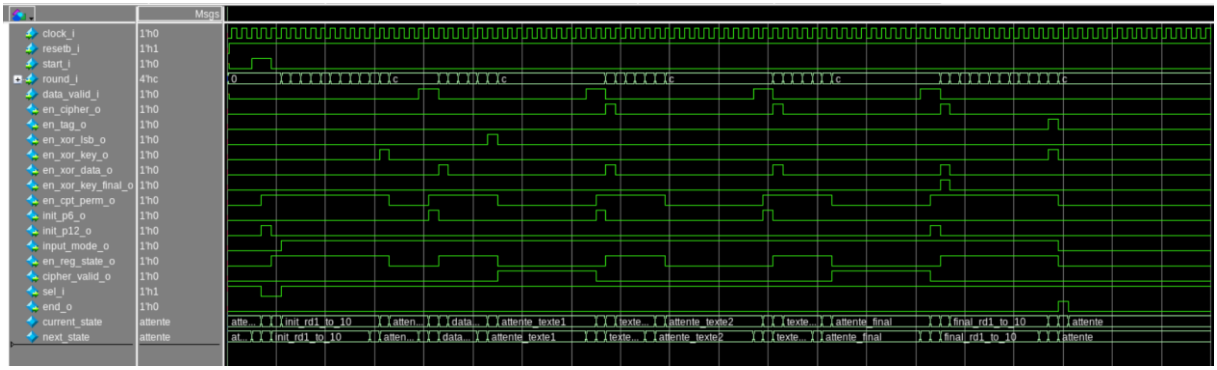

Figure 18 - Simulation de la Machine de Moore

MODÉLISATION FINAL D'ASCON

Le module final que nous obtenons et qui effectue notre algorithme ASCON 128 est le module *ascon_top*. Ce module permet de simuler complètement notre algorithme de chiffrement en faisant appel aux différents modules que nous avons implémenter durant ce projet.

Le module *ascon_top* marche notamment en mettant en relation les modules permutations, FSM et le compteur double ce qui nous permet d'arriver à notre implémentation finale. Ce module coordonne les différents blocs fonctionnels.

RÉSULTAT DE L'IMPLEMENTATION

Il est important de vérifier la finalisation de notre projet pour pouvoir le valider dans son entièreté.

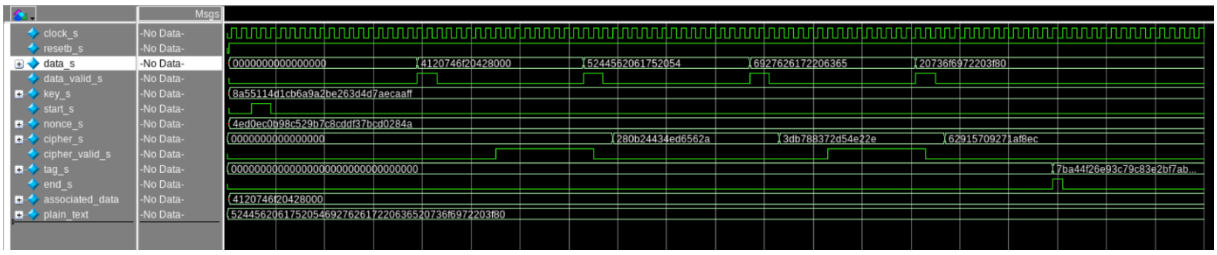

Figure 19 - Simulation d'Ascon_top

Nous obtenons bien les bonnes valeurs pour le cipher et le tag. Les signaux s'activent au bon moment.

Nous pouvons donc valider le résultat final et le projet dans sa globalité.

DIFFICULTÉS RENCONTRÉES

J'ai rencontré plusieurs difficultés durant ce projet. L'une des premières difficultés fut le niveau d'abstraction et de recul nécessaire à la compréhension de l'algorithme dans son ensemble. Il m'a paru assez complexe aux premiers abords de comprendre les différentes parties de l'algorithme et comment elles s'emboîtaient ensemble.

Par la suite, j'ai rencontré beaucoup de problèmes quant au codage des différents modules et leurs testbenchs. J'ai rencontré beaucoup d'erreurs et il n'a pas

toujours été facile d'en détecter la cause et de les régler. Il est notamment assez compliqué de prendre en main pour la première fois un nouveau langage et devoir effectuer tout un projet avec celui-ci.

CONCLUSION

Ce projet fut une occasion de découvrir un nouveau domaine qui est très intéressant. Il est remarquable de voir comment un tel algorithme qui aux premiers abords serait développé de manière logicielle à l'aide de langage de programmation comme le C ou le python peut être développé de manière logicielle.

Le projet fut une bonne expérience qui se termine sur une note positive. Il reste toutefois des points qui peuvent être amélioré dans sa mise en place. Les améliorations majeures qui peuvent être effectuées se situent notamment au niveau de la machine d'états. Nous pouvons déjà réduire drastiquement le nombre d'états en mettant en place le compteur de blocs mais également en remplaçant la machine de Moore par une machine de Mealy. Cela permettrait notamment d'améliorer les performances en temps du système.

En conclusion, ce projet m'a permis d'acquérir des bases solides en SystemVerilog tout en m'introduisant aux cryptographies et à l'algorithme de chiffrement ASCON.