Compte-rendu : Projet de conception d'un système numérique

Ascon-AEAD128

Étudiants
Hamza BELARBI

Professeurs
Jean-Baptiste RIGAUD

2024/2025

Table des matières

1 Module `ascon_top`

1.1 Description du module

Le module `ascon_top` regroupe l'ensemble des blocs nécessaires à la réalisation de l'algorithme ASCON-AEAD128.

Il contient les éléments suivants :

— la machine d'état (`FSM`) responsable du séquencement des phases,

— le compteur `compteur_double_init` ,piloté par la FSM, gère les rondes de permutation,

— le bloc de `permutation_simple` qui applique les transformations (pc, ps, pl) sur l'état de 320 bits,

— les blocs `XOR` que l'on a rassemblé sous les appelations de `xor_end` et `xor_begin`. En fonction de s'il agissait sur les 3 derniers registres (les 192 derniers bits) ou sur les 2 premiers registres (les 128 premiers bits).

— les registres de sortie pour le texte chiffré (`cipher_o`) et le tag d'authentification (`tag_o`).

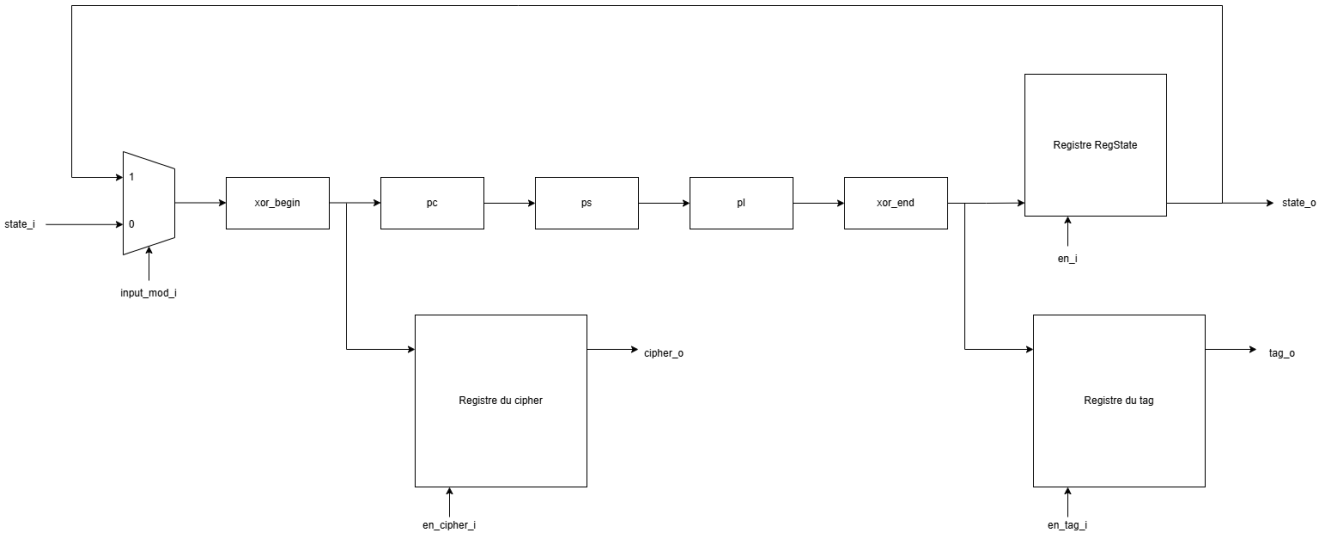

FIGURE 1 – schéma de l'algorithme ASCON-AEAD128

Remarque : on a rassemblé la `permutation_simple`, les blocs `xor_end` et `xor_begin` ainsi qu'un registre de 320 bits (qui permet d'enregistrer les 5 registres en sortie de permutation) et les deux registres de 128 bits du `cipher` et du `tag` dans un seul et même module : `permutation_finale`.

Le module reçoit en entrée le message, la clé, le nonce, ainsi qu'un signal de validation des données. Il produit en sortie un texte chiffré C1, C2, C3 et un tag T de 128 bits. Le texte chiffré couplé aux données associées permet de calculer le tag et s'il correspond au tag en sortie alors cela permet de valider la confirmité de l'information transmise.

L'ensemble est cadencé par un signal d'horloge `clock_i` et contrôlé par un signal de réinitialisation `resetb_i`.

1.2 Chronogramme de simulation

La figure ci-dessous montre le chronogramme résultant de la simulation du module `ascon_top`.

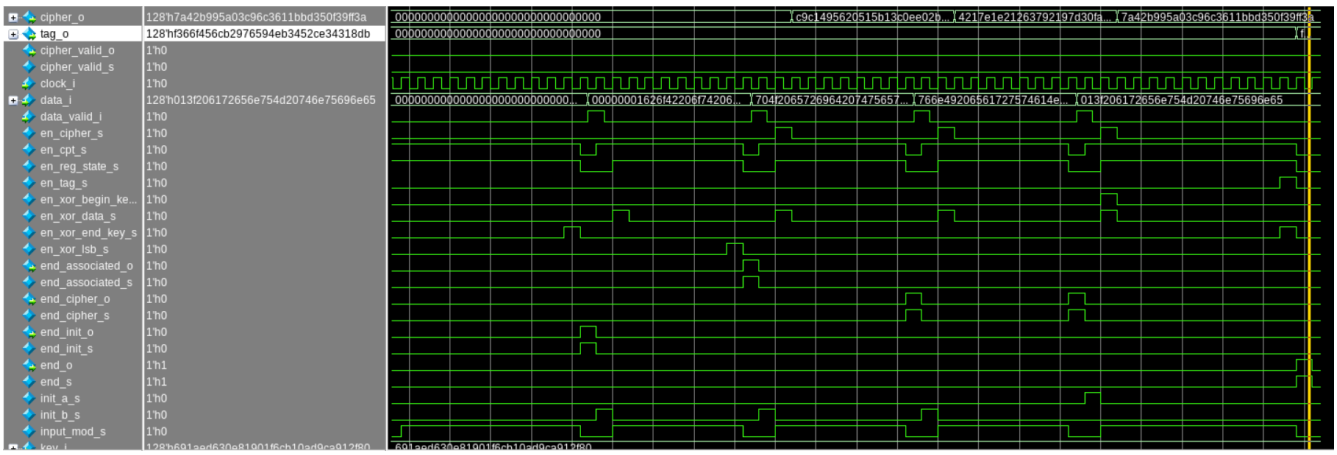

FIGURE 2 – Chronogramme de simulation du module `ascon_top.sv`

On se réfère au sujet et on observe le `texte chiffré` ainsi que le `tag` attendu. Ce test nous permet donc de valider l'intégrité de notre module `ascon_top`. Les différents modules composant ce dernier sont présentés dans la suite.

2 Chronogrammes des modules et interprétation

Les modules constituant l'algorithme Ascon-AEAD128 ont tous été simulés séparément à l'aide de ModelSim.

2.1 Module `pc.sv` – Ajout de constante

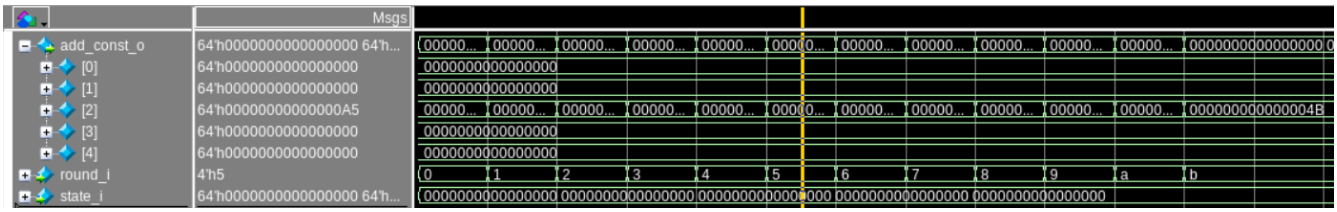

FIGURE 3 – Chronogramme : module `pc.sv`

Interprétation : On a initialisé $\{S_0, S_1, S_2, S_3, S_4\}$ à 0. Cela nous permet de lire la constante effectivement ajoutée à chaque ronde. Le chronogramme montre que seuls les 8 bits de poids faible de S_2 changent, tandis que les autres parties de l'état restent inchangées. De plpus les constantes ajoutées sont conformes au sujet. Le module est validé.

2.2 Module `ps.sv` – Substitution (SBOX)

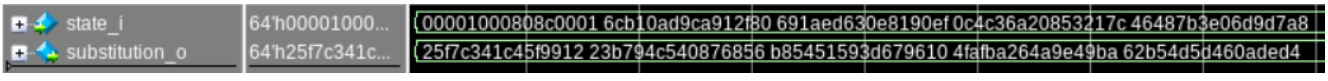

FIGURE 4 – Chronogramme : module `ps.sv`

Interprétation : Chaque colonne de 5 bits de l'état est traitée par une SBOX. Grâce aux données conférées par le sujet, on peut valider le module car le résultat est bien celui attendu.

2.3 Module `pl.sv` – Diffusion linéaire

FIGURE 5 – Chronogramme : module `pl.sv`

Interprétation : Ce module applique des rotations et des XOR internes à chaque mot de l'état. Grâce aux données conférées par le sujet, on peut valider le module car le résultat est bien celui attendu.

2.4 Permutation finale – `permutation_finale.sv`

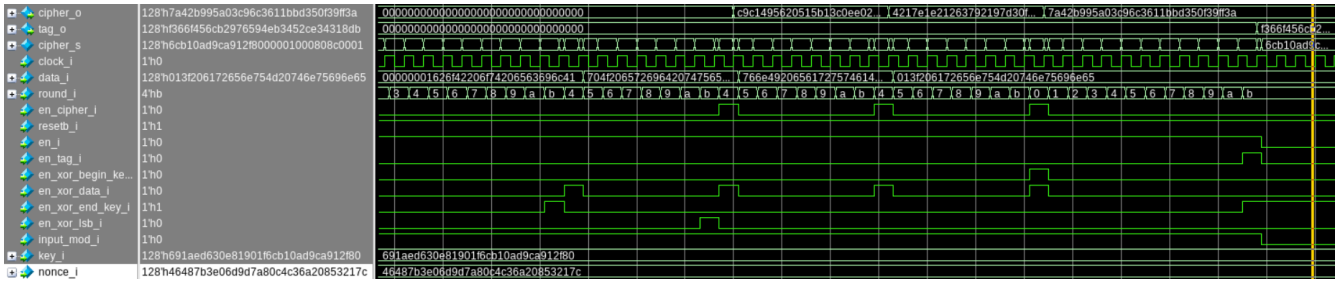

FIGURE 6 – Chronogramme : permutation complète sur plusieurs rounds

Interprétation : On observe le comportement du module sur l'intégralité de l'algorithme ASCON-AEAD128. Ce module est composé des opérations pc,ps, pl, des xor_end et xor_begin ainsi que les registres servant à stocker le cipher, le tag et l'état de 320 bits. L'évolution de l'état est visible ronde après ronde, confirmant que la boucle de permutation fonctionne correctement. Sur la capture d'écran on peut lire le cipher et le tag. Ces derniers sont conformes au sujet on peut donc garantir le fonctionnement du module.

Remarque : Une permutation intérmédiaire constitué de la permutation simple ainsi que des modules xor et du registre d'état a été réalisé au préalable. Mais je n'ai pas jugé important de le mettre étant donné que le testbench de la permutation finale assure son bon fonctionnement. J'avais aussi réalisé un test pour la permutation simple mais je l'ai supprimé par inadvertance et je n'ai pas jugé important de le refaire étant donné que j'avais déjà fait un test satisfaisant sur la permutation intermédiaire au moment de cette suppression.

3 Diagrammes d'architecture des modules

Cette section présente les schémas structurels des principaux modules développés dans le cadre de ce projet. Ils illustrent la manière dont les différents blocs sont connectés pour mettre en œuvre l'algorithme ASCON-AEAD128.

3.1 Architecture de `permutation_simple.sv`

Le module `permutation_simple` regroupe les trois fonctions élémentaires définies précédemment : `pc` (addition de constante), `ps` (substitution non linéaire), et `pl` (diffusion linéaire).

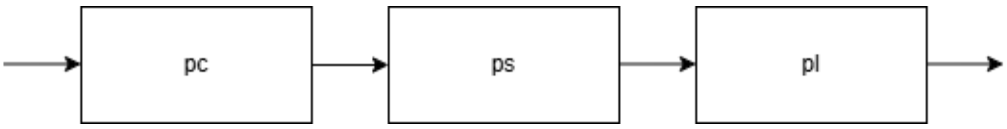

FIGURE 7 – Architecture du module `permutation_simple.sv`

3.2 Architecture des XOR dans la permutation

Pour respecter le protocole ASCON, des opérations XOR sont appliquées sur l'état à des moments précis : avec la clé, les données associées, ou un bit LSB. Ces XOR sont contrôlés par des signaux issus de la FSM.

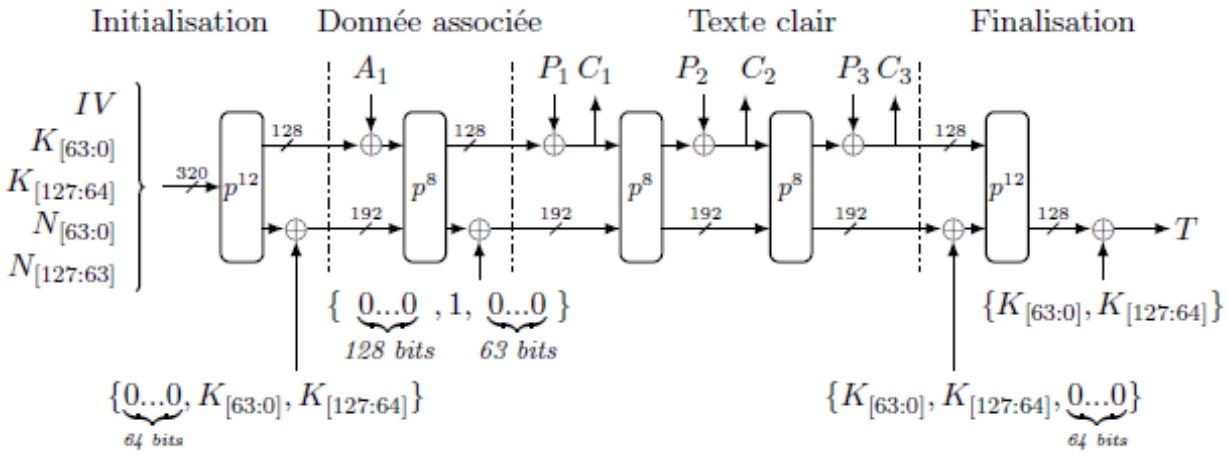

FIGURE 8 – Architecture des opérations XOR autour de la permutation

Des signaux tels que `en_xor_data_o` ou `en_xor_end_key_o` permettent de controler les modules xor.

3.3 Architecture du module `ascon_top.sv`

Le module `ascon_top` regroupe l'ensemble des composants développés : la machine d'état, le compteur de round, la permutation complète, les registres de sortie, et les blocs XOR.

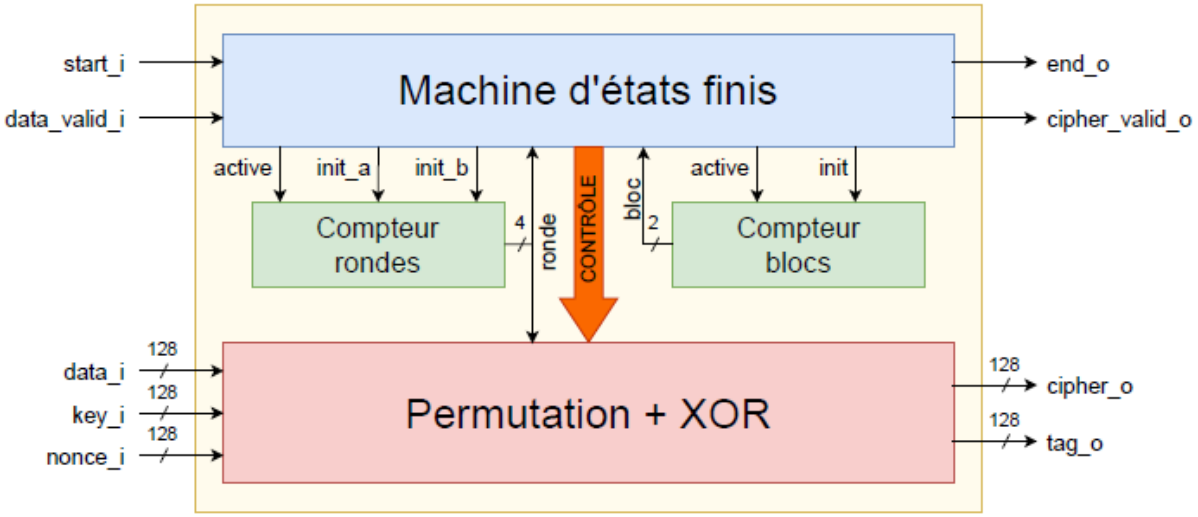

FIGURE 9 – Architecture globale du module `ascon_top`

— La machine d'état pilote le compteur et active les XOR et les permutations au bon moment.
— Le compteur fournit l'index de ronde à la permutation.

Le compteur de blocs n'a pas été instancié mais on a plutôt augmenter le nombre d'état dans la machine d'état pour permettre de traiter convenablement la phase de texte clair.

3.4 FSM – Machine d'état

Aucun testbench n'a été réalisé pour la machine d'état, mais son bon fonctionnement est confirmé par le test réalisé sur le module complet présenté plus tôt.

4 Description de la machine d'états gérant le chiffrement

4.1 Organisation générale de la FSM

La machine d'état pilote le déroulement complet de l'algorithme ASCON-AEAD128. Les différents états sont codés sous forme d'un type énuméré contenant l'ensemble des états.

Le module de la machine d'état repose sur :
— un registre d'état (bloc `always_ff`) mis à jour à chaque cycle d'horloge,
— un bloc combinatoire (transistions entre états),
— un bloc combinatoire (génère les sorties qui sont les signaux de commande du reste de l'algorithme).

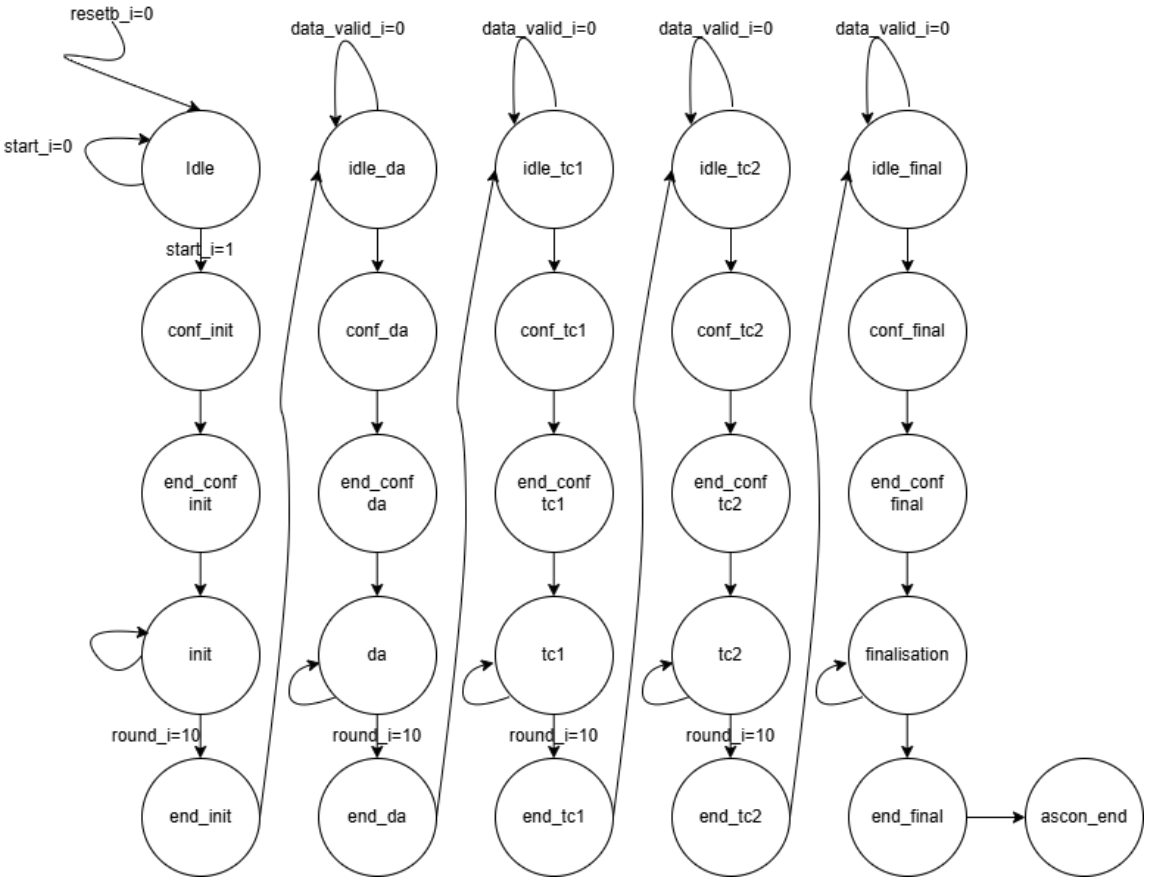

FIGURE 10 – Diagramme d'état de la FSM

4.2 États impliqués dans le chiffrement

Les états dédiés à la phase de chiffrement sont les suivants :

— `idle_tc1` : attente
— `config_tc1` : initialise le compteur
— `end_config_tc1` : configuration du bloc
— `tc1` : traitement de la première phase de chiffrement sur 8 rondes
— `end_tc1` : écriture dans le registre `cipher_o`
— `idle_tc2` → `tc2` : même séquence pour la deuxième phase
— `idle_final` → `finalisation` : dernière bloc de 12 permutations et génération du tag

À chaque passage dans un de ces états, un certain nombre de signaux de commande sont activés pour controler le module de permutation et le compteur .

5 Difficultés rencontrées et solutions envisagées

5.1 Problème 1 – Absence du compteur de blocs

Le principal problème de mon module final est l'absence de compteur de blocs présent dans le sujet. Ce compteur avait pour objectif de déterminer si l'on traite le dernier bloc et n'a pas été implementé à cause d'un léger manque de temps.

Conséquences :

— Le passage de l'état `end_tc1` vers `idle_tc2` ou `idle_final` est géré de manière fixe.

Solution envisagée : Ajouter un compteur de blocs incrémenté à `end_tc`(il n'y aurait plus que 5 états pour la phase de chiffrement), et tester sa valeur pour adapter la suite de la machine d'état. On devrait prévoir un rebouclage sur idle_tc dans le cas où le nombre de bloc est inférieur à 2, on continuerait vers idle_final pour passer à la phase de finalisation.

5.2 Problème 2 – Synchronisation

Un point délicat a été l'activation des différents signaux de commande aux bons moments afin de réaliser les opérations(xor, activation des registres cipher et tag) aux bons fronts d'horloges.

Problème : La lecture du cipher était par exemple faussé par le fait que le registre prenait la valeur trop tard. De même, les xor_end étaient parfois réalisés trop tard et même si le résultat final était le même cela posait problème pour valider l'intégrité des modules.

Solution : Sur les testbench, lorsqu'un xor ou une écriture(cipher) avait lieu en fin de permutation, on initialisait donc quand round_i valait 10 afin de réaliser le xor ou l'écriture au bon front d'horloge

Conclusion

Ce projet nous a permis de modéliser matériellement l'algorithme ASCON-AEAD128, standardisé par le NIST pour le chiffrement authentifié. En partant des spécifications théoriques, nous avons intégré l'ensemble dans un système complet piloté par une machine d'état.

Toutes les fonctionnalités attendues ont été implémentées :

— Gestion des différentes phases du protocole (initialisation, association, chiffrement, finalisation),
— Mise en œuvre de la permutation complète,
— Contrôle par la machine d'état.

Malgré quelques choix simplificateurs (comme l'absence de compteur de blocs), cette implémentation reste fidèle à l'esprit du protocole ASCON.

Ce projet m'a permis de renforcer ma maîtrise du SystemVerilog et m'a servi d'introduction au monde de la cryptographie.

Annexe – Compteur de blocs (conception non implémentée)

Bien que non instancié par manque de temps, un compteur de blocs est prévu par le sujet pour gérer la phase de chiffrement.

Fonctionnement prévu

Le composant aurait :

— reçu un signal d'incrément à chaque `end_tcX`,
— comparé sa valeur à un seuil (par exemple, 2 blocs max),
— envoyé un signal à la FSM pour choisir entre l'état `idle_tc2` ou directement `idle_final`.

Schéma bloc envisagé

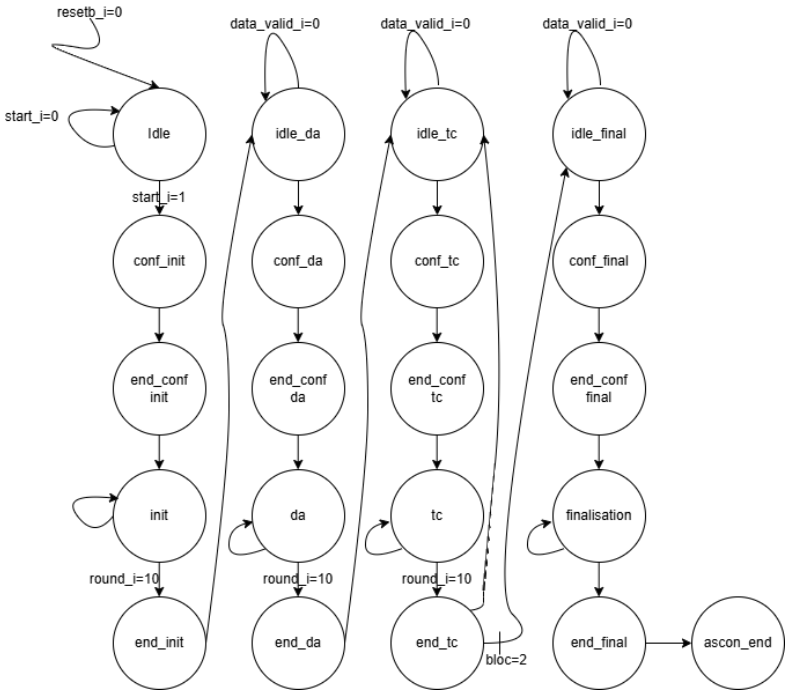

FIGURE 11 – Diagramme d'état si un compteur de blocs avait été implémenté

Références

[1] `ascon_pack.sv` – Définition des types de données et constantes pour ASCON

[2] `register_w_en.sv` – Module SystemVerilog d'un registre

[3] `compteur_double_init.sv` – Compteur 4 bits avec double initialisation (init_a et init_b)

[4] `ascon_top_tb.sv` – Testbench complet de validation du module `ascon_top.sv`