CRYPTOCORE : MODULE DE CHIFFREMENT MATÉRIEL

Programmation des circuits FPGA

Presenté par :

- Zakariae M’ghili
- Hamza Kouissi
- Hatim Bousseta
- Amine Belamine
- Saad El Guelyouy
- Ahmed Aymen Tibtani
- Mouad El Bekkali
- Aya Touirass

Encadré par :

- Pr. Jamal Zbitou

Année universitaire :

- 2025-2026

INTRODUCTION

Le projet CryptoCore vise à répondre aux exigences croissantes de sécurisation des flux de données critiques transmis par les drones, en proposant la conception d'un processeur cryptographique matériel capable d'assurer le chiffrement en temps réel de ces informations à l'aide de l'algorithme AES-128.

Cette présentation aborde l'architecture complète du système, depuis les contraintes définies par le cahier des charges jusqu'à la phase de validation par simulation, en mettant en évidence le rôle essentiel du Data Path et des mémoires FIFO dans la gestion des flux de données, le mécanisme de génération et d'expansion des clés de sécurité, ainsi que l'orchestration globale du système assurée par une unité de contrôle basée sur une machine à états finis (FSM).

SOMMAIRE

CAHIER DES CHARGES

A. Contraintes Techniques :

- **Format des données : Traitement par blocs de 128 bits uniquement.**
- **Mémoire : Profondeur des buffers fixée à 64 mots pour optimiser l'espace FPGA.**
- **VHDL : Implémentation modulaire pour permettre le travail en groupe (séparation Data Path / FSM / AES).**

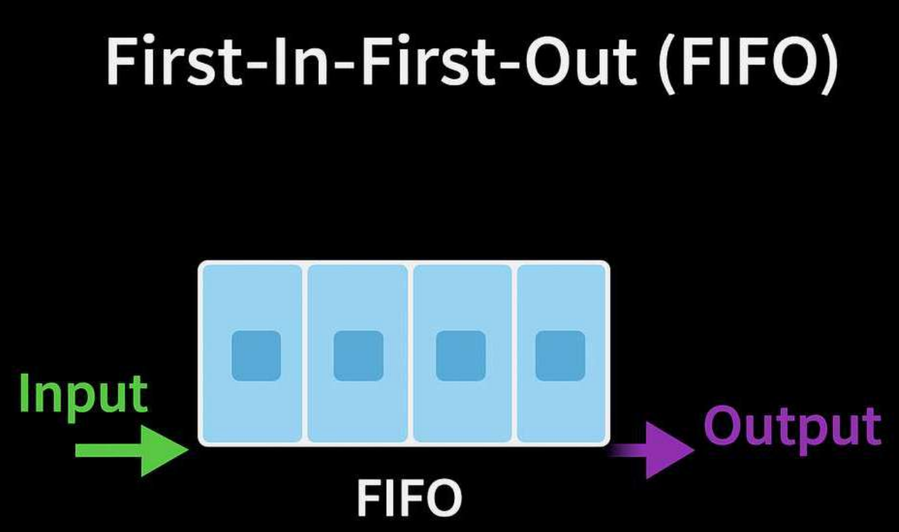
CAHIER DES CHARGES

B. Critères de Sécurité :

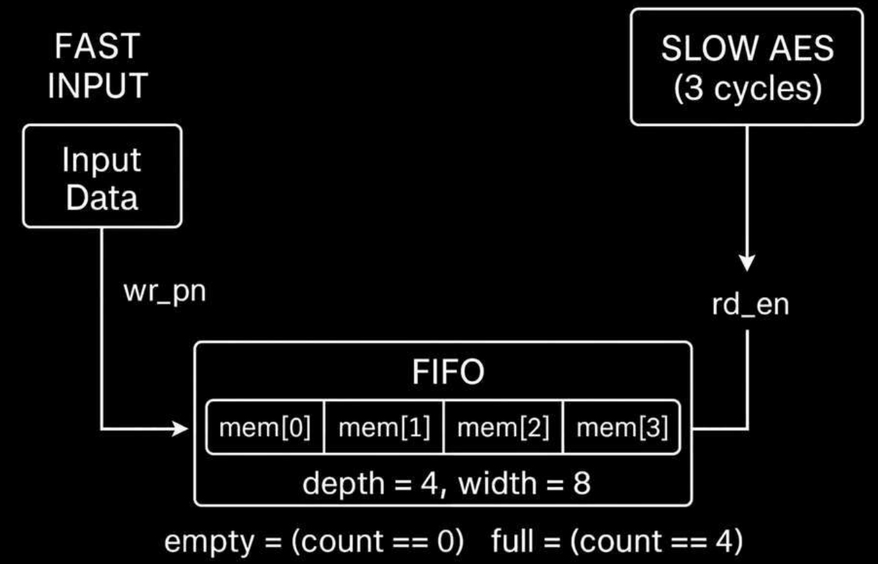
- **Zéro Perte : Interdiction d'écrire si la mémoire est pleine (Anti-Overflow).**
- **Fiabilité : La FSM ne doit lire une donnée que si elle est confirmée comme "prête" (data_ready).**
- **Mise en évidence de la sécurité : Chifrement de données de type AES128**

FIFO (First In First Out)

Le FIFO (First In First Out) est un module matériel de stockage temporaire qui permet de conserver les blocs de données dans l'ordre exact de leur arrivée afin d'assurer un transfert fiable et continu entre les différents modules du système.



Rôle du FIFO dans CryptoCore



- Le moteur AES ne peut traiter un bloc de données qu'après plusieurs cycles d'horloge.
- Les données d'entrée peuvent arriver plus rapidement que l'AES ne peut les traiter.
- Le FIFO agit comme un buffer intermédiaire :
  - Stocke les blocs de données entrants.
  - Les délivre à l'AES dans l'ordre exact où elles sont arrivées.
  - Empêche toute perte de données.

DÉCLARATION DE L'ENTITÉ

Generics :

DATA_WIDTH → taille de chaque bloc de données (8 bits ici).
DEPTH → nombre de blocs stockés dans le FIFO (4 ici).

Ports :

clk → horloge du FIFO
rst → reset, remet le FIFO à zéro
write_en → active l'écriture dans le FIFO
read_en → active la lecture depuis le FIFO
data_in → données entrantes
data_out → données sorties
full → indique que le FIFO est plein
empty → indique que le FIFO est vide

```
entity fifo is
    generic (
        DATA_WIDTH : integer := 8;  -- size of each block
        DEPTH      : integer := 4   -- number of blocks
    );
    port (
        clk       : in  std_logic;
        rst       : in  std_logic;
        write_en  : in  std_logic;
        read_en   : in  std_logic;
        data_in   : in  std_logic_vector(DATA_WIDTH-1 downto 0);
        data_out  : out std_logic_vector(DATA_WIDTH-1 downto 0);
        full      : out std_logic;
        empty     : out std_logic
    );
end fifo;
```

ARCHITECTURE ET MÉMOIRE INTERNE

```
architecture Behavioral of fifo is
    type memory_array is array (0 to DEPTH-1) of std_logic_vector(DATA_WIDTH-1 downto 0);
    signal mem    : memory_array := (others => (others => '0'));
    signal rd_ptr, wr_ptr : integer range 0 to DEPTH-1 := 0;
    signal count : integer range 0 to DEPTH := 0;
begin
```

memory_array → type pour stocker les blocs du FIFO.
mem → tableau qui contient les données. Initialisé à zéro.
rd_ptr → pointeur de lecture, indique quelle case lire.
wr_ptr → pointeur d'écriture, indique quelle case écrire.
count → nombre de blocs actuellement stockés.

PROCESS PRINCIPAL : LECTURE/ÉCRITURE

```
process(clk, rst)
begin
    if rst = '1' then
        rd_ptr <= 0;
        wr_ptr <= 0;
        count  <= 0;
        data_out <= (others => '0');
```

Si rst = 1, le FIFO est réinitialisé : tous les pointeurs et le compteur à zéro, sortie à zéro.

ÉCRITURE

```
elsif rising_edge(clk) then
    -- Write operation
    if write_en = '1' and count < DEPTH then
        mem(wr_ptr) <= data_in;
        wr_ptr <= (wr_ptr + 1) mod DEPTH;
        count <= count + 1;
    end if;
```

- Écrit data_in dans mem si le FIFO n’est pas plein.
- wr_ptr avance au prochain emplacement (modulo DEPTH pour revenir à 0 si fin atteinte).
- count augmente, suivi du nombre de blocs stockés.

LECTURE

```
-- Read operation
if read_en = '1' and count > 0 then
    data_out <= mem(rd_ptr);
    rd_ptr <= (rd_ptr + 1) mod DEPTH;
    count <= count - 1;
end if;
```

- Lit le bloc à rd_ptr si le FIFO n’est pas vide.
- Avance rd_ptr pour le prochain bloc.
- count diminue pour refléter le nombre de blocs restants.

FLAGS FULL / EMPTY

```
full  <= '1' when count = DEPTH else '0';
empty <= '1' when count = 0 else '0';
```

full = 1 → indique que le FIFO est plein, pas d'écriture possible.
empty = 1 → indique que le FIFO est vide, pas de lecture possible.

DATA PATH

Définition :

- C’est l’ensemble des blocs matériels assurant le transfert et le traitement des données. Il est responsable de la circulation des données entre les différents modules du système.
- C’est un concept fondamental en architecture informatique.

Rôle :

- Épine dorsale de l’architecture matérielle

Exemple :

- A data path is a crucial component of a computer's architecture that facilitates the processing and transfer of data within the CPU

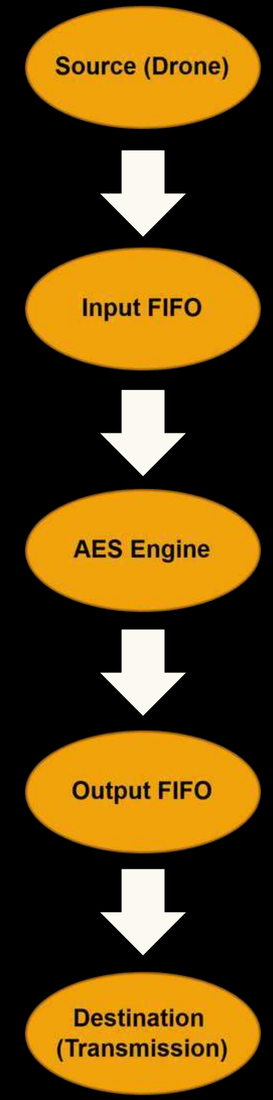
DATA PATH

Implémentation du Data Path :

- Implémenté dans le fichier réalisé en VHDL nommé CryptoCore_Data_Path.vhd
- Gère l'interface de Statut(data_ready, output_fifo_not_full) pour l'Unité de Contrôle (FSM)
- Il assure l’instanciation et l’interconnexion des blocs :

Fonction principale :

- Intégration centralisée des blocs matériels
- Acheminement cohérent des données
- Contrôle précis des échanges internes



DATA PATH

Explication des blocs du code pour bien comprendre le rôle du Data Path :

- **Importation des librairies IEEE pour utiliser des types et opérations VHDL standard, assurant portabilité et fiabilité.**

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.NUMERIC_STD.ALL;
```

- **Définition des paramètres globaux (largeur AES et profondeur des FIFO) dans un package afin de rendre le design modulaire et facilement modifiable.**

```
5   package constants_pkg is
6       constant C_DATA_WIDTH : integer := 128;
7       constant C_FIFO_DEPTH : integer := 64;
8   end package constants_pkg;
9
10  library work;
11  use work.constants_pkg.all;
```

DATA PATH

- **Définition de l'interface externe du Data Path avec clk et reset pour la synchronisation et l'initialisation des composants.**

- **Les signaux de l'Unité de Contrôle permettent à la FSM de piloter le Data Path, qui exécute les actions de lecture, écriture et chiffrement AES.**

- **Les sorties du Data Path informent la FSM de son état interne : disponibilité des données, capacité de la FIFO de sortie et fin du traitement AES.**

- **Les ports du Data Path assurent l'échange des données entre la source (drone, capteurs) et le module de transmission chiffrée, connectant le CryptoCore à l'environnement externe.**

```
13  entity CryptoCore_Data_Path is
14      port (
15          clk                 : in std_logic;
16          reset               : in std_logic;
```

```
18          fsm_read_enable   : in std_logic;
19          fsm_write_enable  : in std_logic;
20          fsm_start_enc     : in std_logic;
```

```
22          data_ready            : out std_logic;
23          output_fifo_not_full  : out std_logic;
24          ciphertext_ready      : out std_logic;
```

```
26          data_in_from_source : in std_logic_vector(C_DATA_WIDTH-1 downto 0);
27          data_out_to_sink    : out std_logic_vector(C_DATA_WIDTH-1 downto 0)
28      );
29  end entity CryptoCore_Data_Path;
30
```

DATA PATH

- Ce bloc déclare le composant FIFO utilisé pour le buffering des données. Il est paramétrable en largeur et en profondeur et fournit les signaux full et empty, essentiels pour le contrôle de flux et la sécurité des données.

```
31  architecture Structure of CryptoCore_Data_Path is
32
33      component fifo is
34          generic (DATA_WIDTH : integer := C_DATA_WIDTH; DEPTH : integer := C_FIFO_DEPTH);
35          port (
36              clk, rst, write_en, read_en : in std_logic;
37              data_in  : in std_logic_vector(DATA_WIDTH-1 downto 0);
38              data_out : out std_logic_vector(DATA_WIDTH-1 downto 0);
39              full, empty : out std_logic
40          );
41      end component fifo;
```

- Ce composant représente le moteur AES. Le Data Path le considère comme une boîte noire chargée du chiffrement. Il fournit les données en entrée, la clé, et récupère les données chiffrées ainsi que le signal indiquant la fin du chiffrement.

```
43      component AES_Engine is
44          port (
45              clk, reset, start_enc : in std_logic;
46              data_in       : in std_logic_vector(C_DATA_WIDTH-1 downto 0);
47              data_out      : out std_logic_vector(C_DATA_WIDTH-1 downto 0);
48              key_in        : in std_logic_vector(C_DATA_WIDTH-1 downto 0);
49              ciphertext_ready : out std_logic
50          );
51      end component AES_Engine;
```

- Les signaux internes relient les composants, transmettent les données entre FIFO et AES, et stockent l'état des FIFOs, tandis que la clé est simplifiée pour se concentrer sur le Data Path.

```
53      signal input_fifo_data_out : std_logic_vector(C_DATA_WIDTH-1 downto 0);
54      signal aes_data_out        : std_logic_vector(C_DATA_WIDTH-1 downto 0);
55
56      signal input_empty_flag    : std_logic;
57      signal output_full_flag    : std_logic;
58
59      signal internal_key_bus    : std_logic_vector(C_DATA_WIDTH-1 downto 0) := (others => '0');
```

DATA PATH

- **Cette FIFO stocke temporairement les données provenant de la source. La lecture est contrôlée par la FSM, tandis que le signal empty permet de détecter la disponibilité des données. Elle joue un rôle clé dans le découplage temporel entre la source et le moteur AES.**

- **Le moteur AES reçoit les données issues de la FIFO d'entrée et effectue le chiffrement. Le signal ciphertext_ready indique à la FSM que le bloc chiffré est prêt, assurant une synchronisation correcte avec le reste du système.**

```
61  begin
62
63      Input_UUT : component fifo
64          generic map (DATA_WIDTH => C_DATA_WIDTH, DEPTH => C_FIFO_DEPTH)
65          port map (
66              clk => clk,
67              rst => reset,
68              write_en => '1',
69              read_en => fsm_read_enable,
70              data_in => data_in_from_source,
71              data_out => input_fifo_data_out,
72              full => open,
73              empty => input_empty_flag
74          );
```

```
76      AES_UUT : component AES_Engine
77          port map (
78              clk => clk,
79              reset => reset,
80              start_enc => fsm_start_enc,
81              data_in => input_fifo_data_out,
82              data_out => aes_data_out,
83              key_in => internal_key_bus,
84              ciphertext_ready => ciphertext_ready
85          );
```

DATA PATH

- **Cette FIFO stocke les données chiffrées avant leur transmission. Le signal full empêche toute écriture supplémentaire lorsque la FIFO est saturée, garantissant une protection contre l'overflow et la perte de données.**

```
87      Output_UUT : component fifo
88          generic map (DATA_WIDTH => C_DATA_WIDTH, DEPTH => C_FIFO_DEPTH)
89          port map (
90              clk => clk,
91              rst => reset,
92              write_en => fsm_write_enable,
93              read_en => '1',
94              data_in => aes_data_out,
95              data_out => data_out_to_sink,
96              full => output_full_flag,
97              empty => open
98          );
```

- **Cette logique convertit les signaux internes des FIFOs en signaux simples pour la FSM. Elle indique clairement quand une donnée est disponible et quand l'écriture est autorisée, simplifiant ainsi le contrôle global du système.**

```
100         data_ready <= NOT input_empty_flag;
101
102         output_fifo_not_full <= NOT output_full_flag;
103
104     end Structure;
```

DATA PATH

Récapitulatif :

Pour conclure, on peut dire que le FIFO et le Data Path sont deux parties essentielles de notre projet parce qu'ils :

- **Assurent une transmission de données fiable**
- **Garantissent un découplage temporel efficace**
- **Offrent une interface claire et robuste avec l'Unité de Contrôle (FSM)**
- **Constituent une base solide et sécurisée pour le fonctionnement global du CryptoCore**

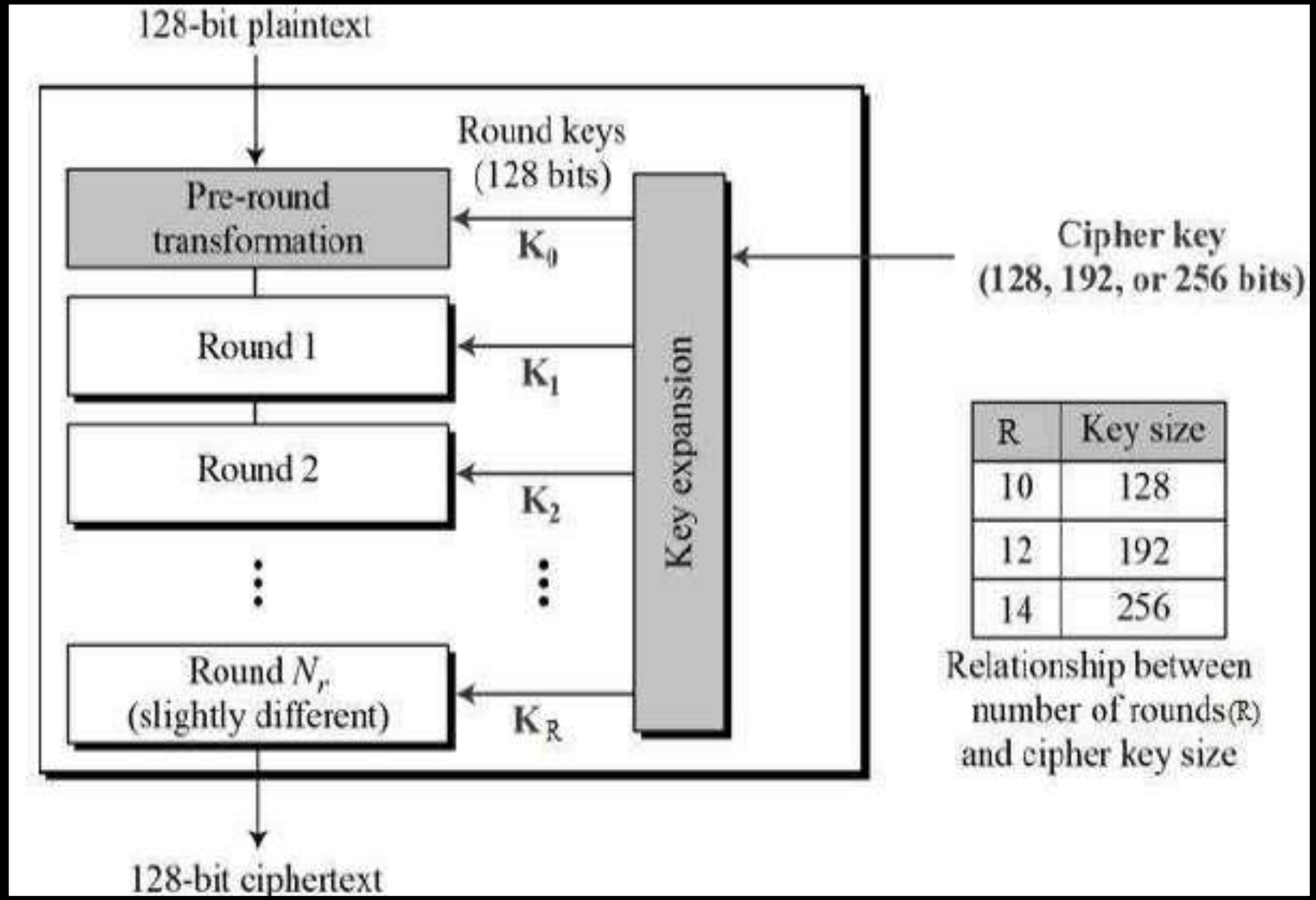
KEY GENERATION

En Cryptographie : La clé (Key) C'est le secret qui contrôle le chiffrement et le déchiffrement des données.

Méthodes de génération de clé :

- PRNG
- TRNG
- CSPRNG

La clé doit être aléatoire pour garantir la sécurité (imprévisibilité) et l'unicité du chiffrement.



R	Key size
10	128
12	192
14	256

Relationship between number of rounds(R) and cipher key size

LA NATURE PROFONDE DE L'ALÉATOIRE

L'Aléa Véritable (True Randomness).

-L'aléa véritable est le seul type de hasard qui offre une imprévisibilité totale.

- Il provient de phénomènes physiques qui sont intrinsèquement et fondamentalement imprédictibles, même si l'on connaît toutes les lois de la physique. On parle de sources d'entropie.

- Exemples : Le bruit thermique dans un transistor, le désordre quantique, ou la gigue (jitter) dans la fréquence d'un oscillateur d'horloge.

-Il est impossible de recréer exactement la même séquence de bits aléatoires, car les conditions initiales du phénomène physique ne peuvent jamais être reproduites à l'identique.

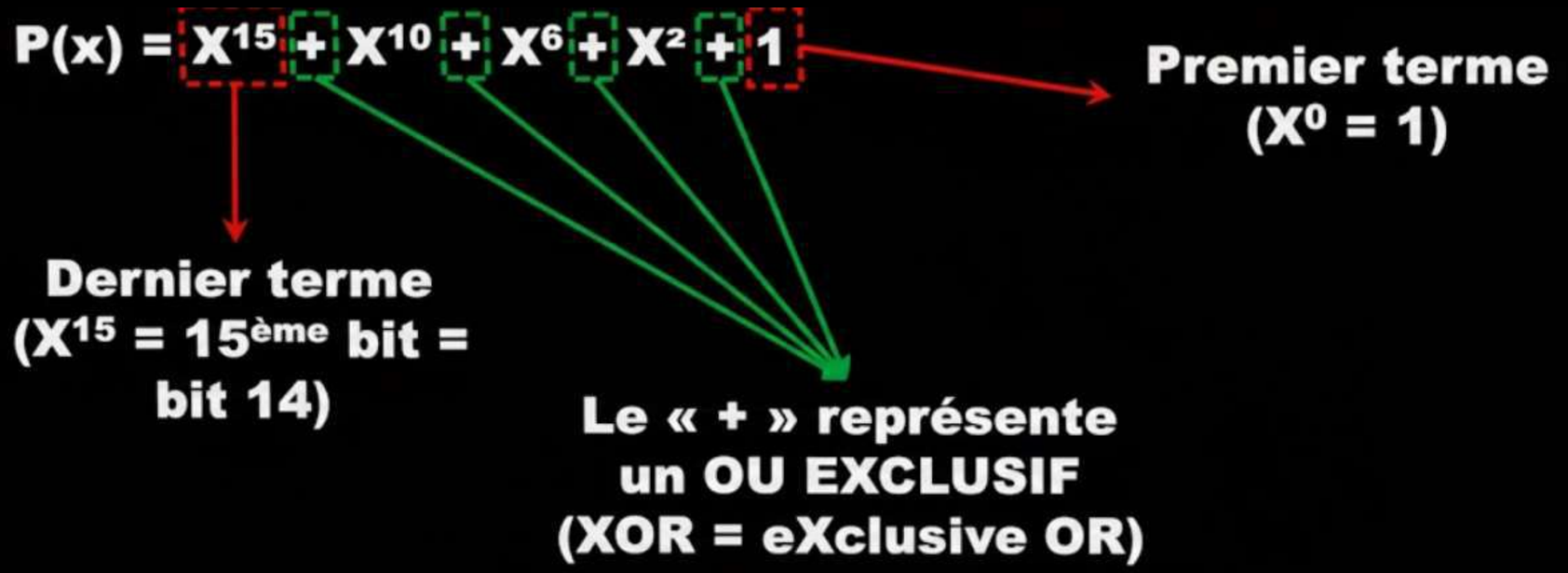
PRNG
PSEUDO-RANDOM NUMBER GENERATOR

Le PRNG commence avec une valeur initiale
appelée graine (seed)

Technique de Base : LFSR (Linear Feedback Shift Register)
Fonctionnement : Un LFSR est une chaîne de registres à décalage où le bit d'entrée est le résultat d'un XOR (feedback) de certains bits de la chaîne .

Le choix des bits qui sont XORés (les taps du LFSR) est défini par un polynôme primitif (ou polynôme générateur).
Un bon polynôme assure la période maximale de la séquence 2^n - 1.

La séquence semble aléatoire, mais elle est en réalité déterministe



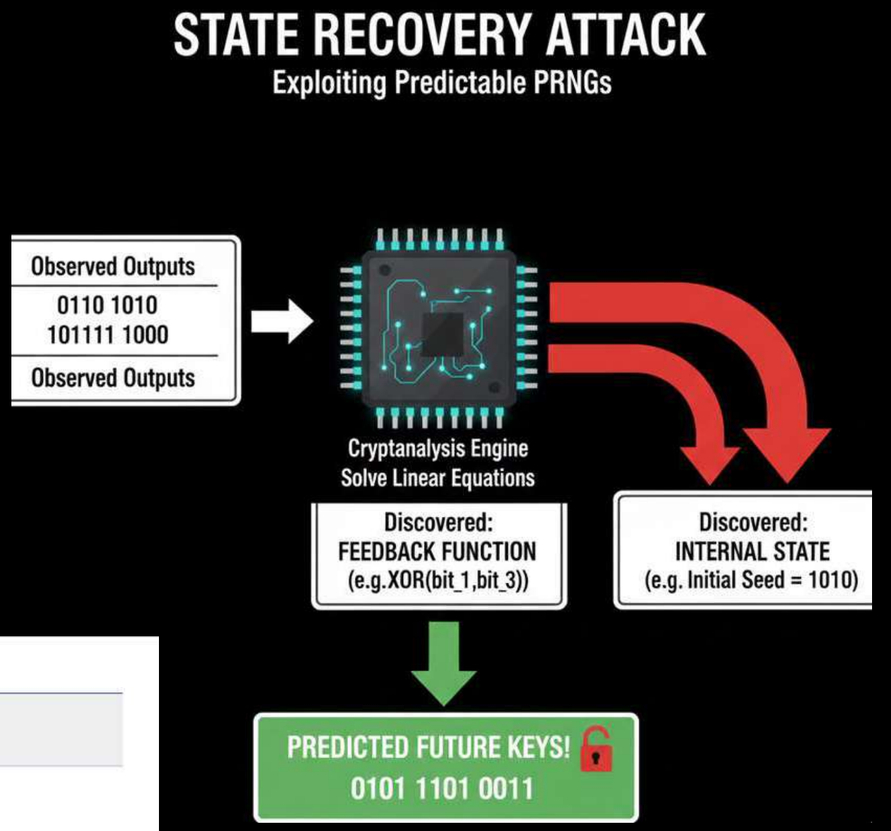
$$P(x) = X^{15} + X^{10} + X^{6} + X^{2} + 1$$


Wave - Default
Msgs
/tb_prng_128/s_clk
/tb_prng_128/s_rst
/tb_prng_128/s_ke...
0
0
266AAEF3377BB...
004488CD115599DE2266AAEF3377BBFF
0089119A22AB33BC44CD55DE66EF77FF
01122334455667788 99AABBCCDDEEFFF
02244...

PROBLÈMES DE SÉCURITÉ DES PRNG DANS LE CHIFFREMENT

- Problème n°1 – La Graine par Défaut (Predictable Seed Attack)

- Problème n°2 – Les Relations de Linéarité (State Recovery Attack)



CWE-337: Predictable Seed in Pseudo-Random Number Generator (PRNG)

Weakness ID: 337
Vulnerability Mapping: ALLOWED
Abstraction: Variant

View customized information: Conceptual | Operational | Mapping Friendly | Complete | Custom

Description

A Pseudo-Random Number Generator (PRNG) is initialized from a predictable seed, such as the process ID or system time.

Extended Description

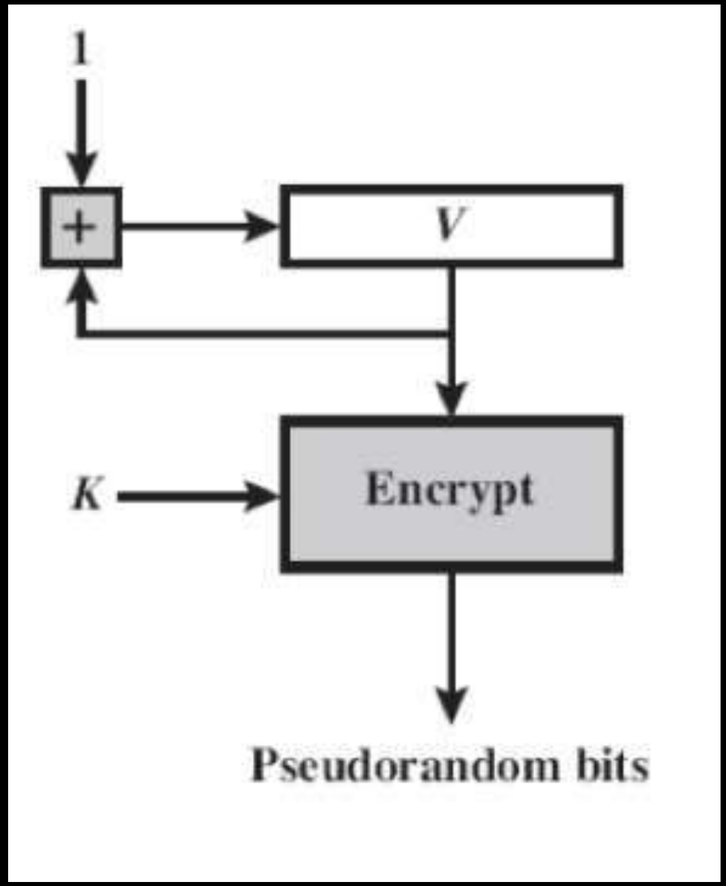
The use of predictable seeds significantly reduces the number of possible seeds that an attacker would need to test in order to predict which random numbers will be generated by the PRNG.

L'APPROCHE IDÉALE ?

TRNG + CSPRNG

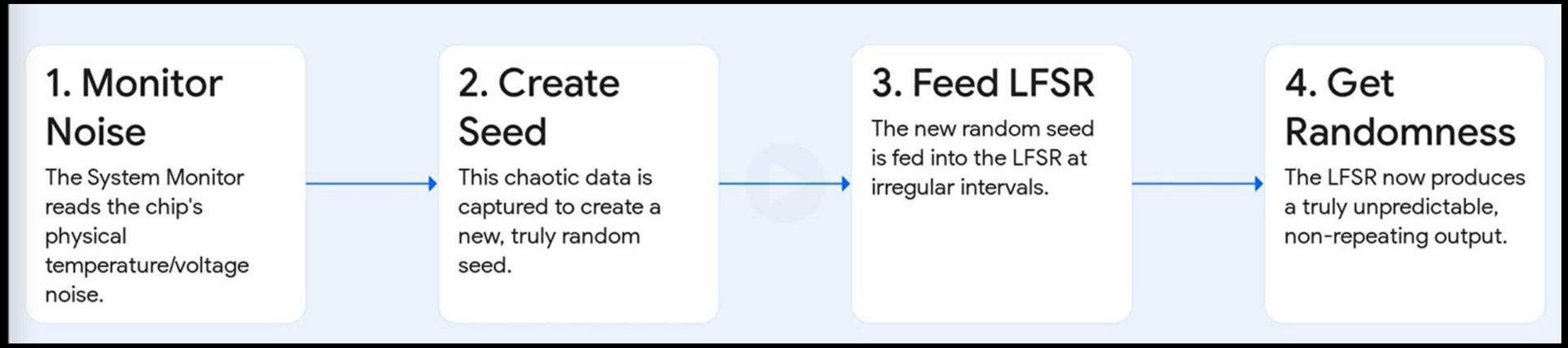
- CSPRNG : Cryptographically Secure Pseudo-Random Number Generator (Générateur de Nombres Pseudo- Aléatoires Cryptographiquement Sécurisé).

- Le CSPRNG utilise l'algorithme AES (hautement non-linéaire) pour transformer l'état interne, le rendant impossible à prédire par des analyses mathématiques. De plus, il garantit que cet état interne est initialisé et régulièrement mis à jour avec de l'entropie forte (TRNG), empêchant toute attaque par graine par défaut

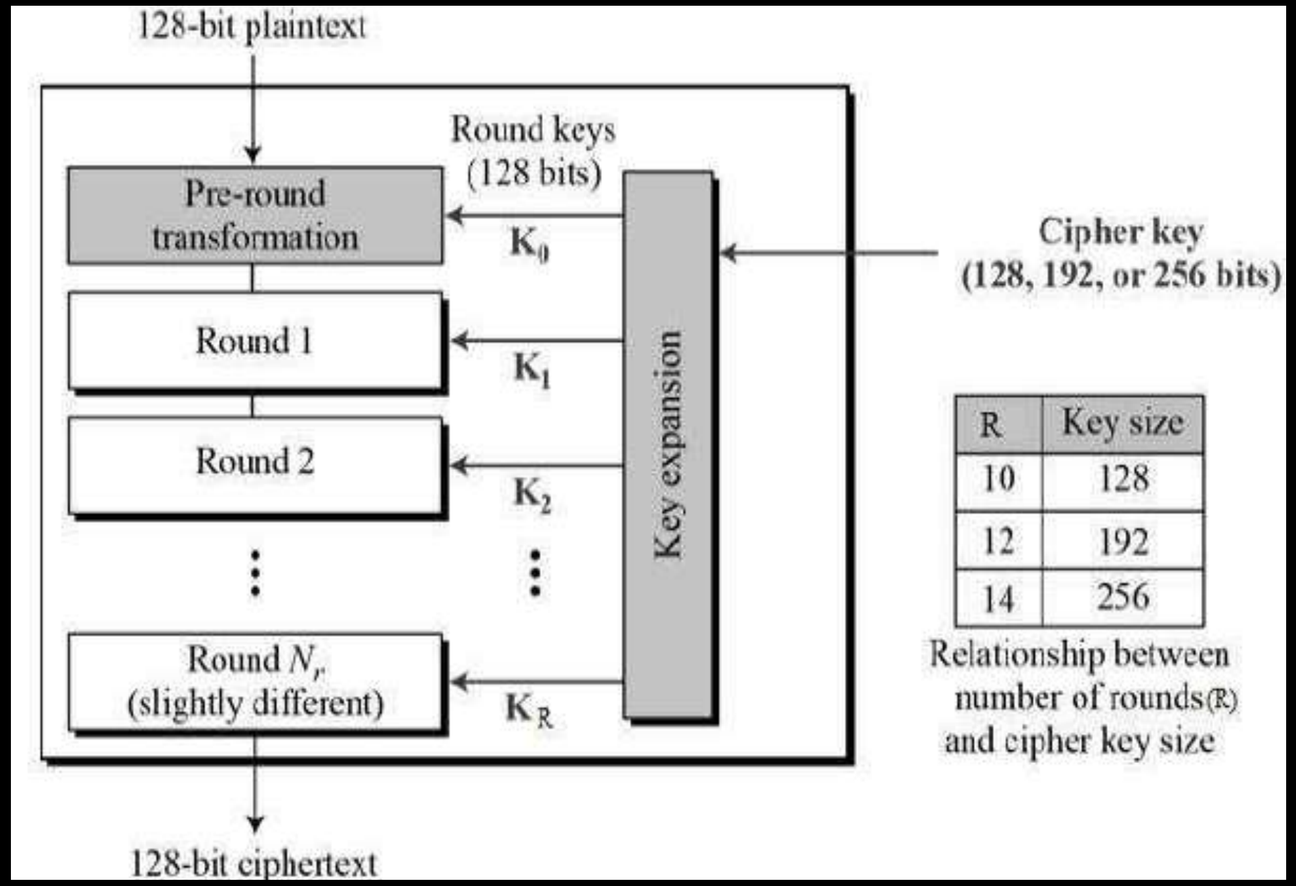


TRNG
TRUE RANDOM NUMBER GENERATOR

Le TRNG (True Random Number Generator) est un générateur de nombres véritablement aléatoires qui utilise un phénomène physique imprévisible (appelé source d'entropie), comme le bruit thermique ou la désintégration radioactive, pour produire des nombres. Contrairement aux PRNG, ses sorties sont non reproductibles et ne peuvent pas être prédites, garantissant ainsi le plus haut niveau de sécurité pour la génération de clés cryptographiques.



LE PROCESSUS DE GÉNÉRATION DE LA CLÉ



R	Key size
10	128
12	192
14	256

Relationship between number of rounds(R) and cipher key size

LA DIFFÉRENCE FONDAMENTALE D'UTILISATION DE L'AES :

AES NORMAL (CHIFFREMENT DE DONNÉES) :

PLAINTEXT + CIPHERKEY = CIPHERTEXT

AES EN MODE CSPRNG :

MASTER KEY + COMPTEUR = KEY

KEY = AES_ENCRYPT(C , K)

Implementation de CSPRNG & AES_Core VHDL

Vue General :

Qu'est-ce qu'un CSPRNG ?

Un CSPRNG est un générateur de nombres pseudo-aléatoires conçu pour la cryptographie. Il produit des séquences qui paraissent aléatoires, mais qui sont impossibles à prédire, même pour un attaquant puissant.

Pourquoi c’est important ?

Parce qu'en crypto, on a besoin de nombres aléatoires sécurisés pour générer des clés, des nonces, des vecteurs d'initialisation...

Si l'aléatoire est prévisible, tout le système de sécurité s'effondre !

Relation entre CSPRNG et AES

AES est un algorithme de chiffrement par bloc très sécurisé et largement utilisé

on utilise souvent AES pour construire des CSPRNG !

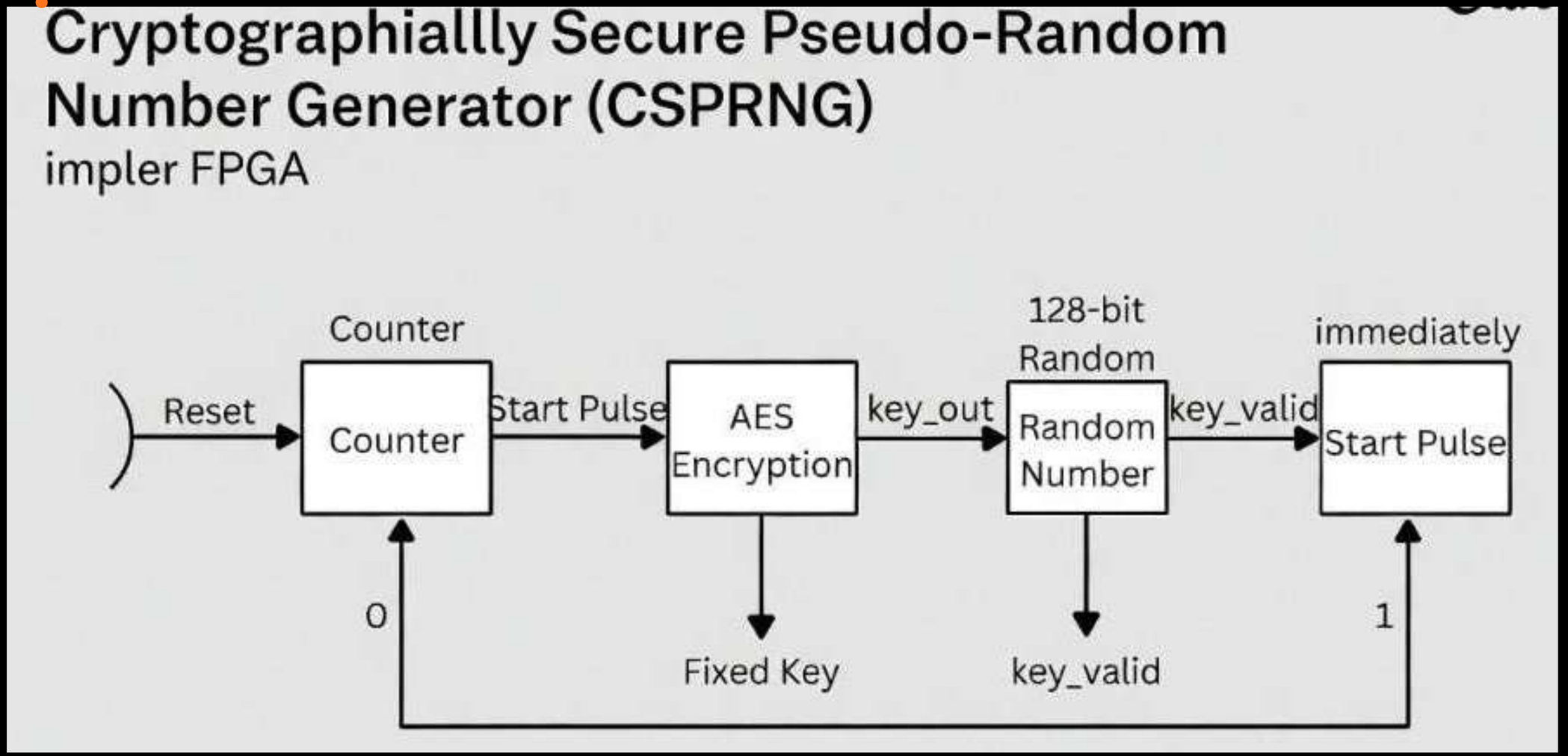
Comment ça fonctionne ?

l’ AES utilise le mode CTR.

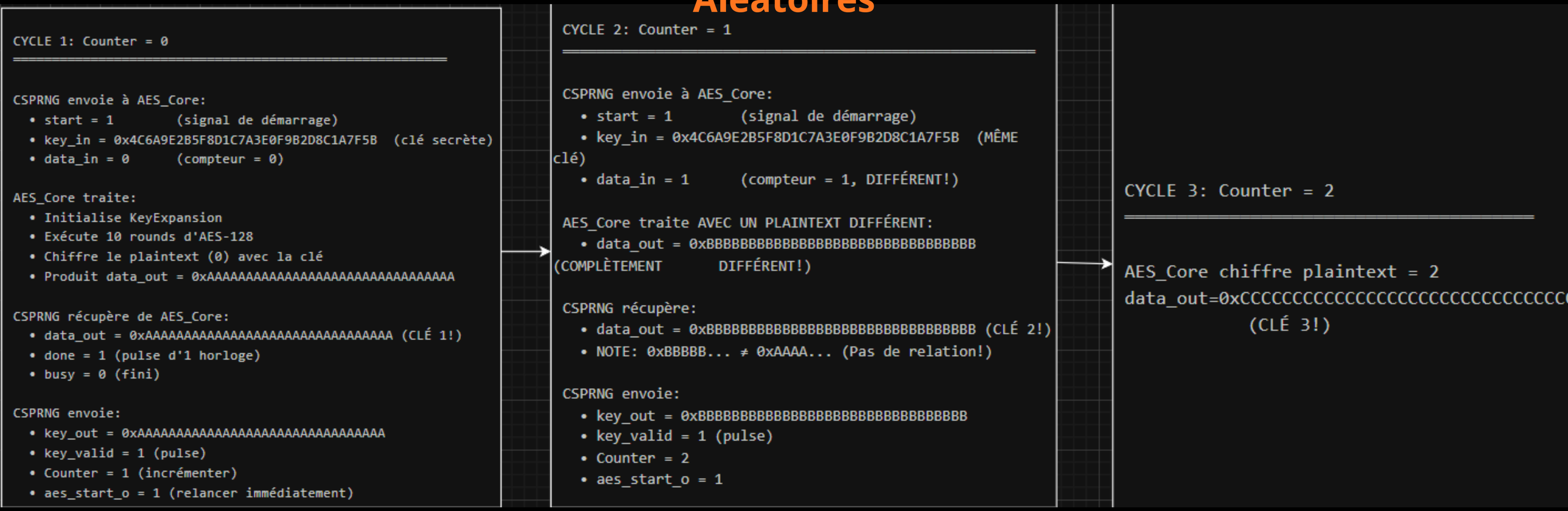
Au lieu de chiffrer directement le texte clair (plaintext), AES-CTR chiffre un compteur. Le résultat est XORé avec le texte clair pour produire le texte chiffré (ciphertext).

Workflow

:


Cryptographiallly Secure Pseudo-Random Number Generator (CSPRNG)
impler FPGA
Reset
Counter
Counter
Start Pulse
AES Encryption
key_out
128-bit Random
Random Number
key_valid
immediately
Start Pulse
0
Fixed Key
key_valid
1

Flux de Données: CSPRNG → AES_Core → Clés Aléatoires



CODE 1: CSPRNG

Key Components

Ports :

- clk : Horloge (input).
- reset : Reset synchrone (input, actif à '1').
- key_out : Sortie de la clé aléatoire générée (128 bits).
- key_valid : Pulse d'un cycle indiquant qu'une nouvelle clé valide est prête sur key_out.

```
entity CSPRNG is
  port (
    clk       : in  std_logic;
    reset     : in  std_logic;
    -- The output random key (Ciphertext from AES)
    key_out   : out std_logic_vector(127 downto 0);
    -- One-cycle pulse indicating a NEW, valid 128-bit key is ready
    key_valid : out std_logic
  );
end entity CSPRNG;
```

Un composant AES_Core est instancié

Ports :

- clk,
- reset,
- start ⟶ (pulse pour démarrer),
- key_in ⟶ (clé de chiffrement),
- data_in ⟶ (données à chiffrer),
- data_out ⟶ (ciphertext),
- busy ⟶ (en cours),
- done ⟶ pulse quand fini).

```
component AES_Core
  port (
    clk      : in  std_logic;
    reset    : in  std_logic;
    start    : in  std_logic;
    key_in   : in  std_logic_vector(127 downto 0);
    data_in  : in  std_logic_vector(127 downto 0);
    data_out : out std_logic_vector(127 downto 0);
    busy     : out std_logic;
    done     : out std_logic
  );
end component AES_Core;
```

Constantes Internes (La Graine Secrète)

K_CSPRNG :

Qu'est-ce que c'est?

- La clé secrète fixe (128 bits en hexadécimal)
- Elle ne change JAMAIS pendant toute l'exécution
- C'est la "graine" qui produit l'aléa

Dans le drone: c'est déjà programmée au démarrage

Pourquoi?

- Si tu connais K_CSPRNG, tu peux déchiffrer les communications
- Si tu ne la connais pas, les clés te paraissent aléatoires

Impossible de prédire la clé suivante sans connaître K_CSPRNG

```
constant K_CSPRNG : std_logic_vector(127 downto 0) := x"4C6A9E2B5F8D1C7A3E0F9B2D8C1A7F5B";
```

Signaux Internes (À l'Intérieur du Module):

A. Compteur :

Compteur de 128 bits, initialisé à 0.

```
signal Counter          : unsigned(127 downto 0) := (others => '0');
```

B.Signaux pour contrôler l'AES

```
signal aes_start_o      : std_logic := '0';
signal aes_busy_i       : std_logic;
signal aes_done_i       : std_logic;
signal aes_output_data  : std_logic_vector(127 downto 0);
```

aes_start_o : AES, commence le chiffrement!
aes_busy_i : AES est en train de travailler
aes_done_i : AES a terminé" (pulse d'1 horloge)
aes_output_data : Résultat chiffré

Instantiation d'AES_Core (Connexion avec le Chiffreur)

CSPRNG envoie à AES_Core:

- start: "Commence à chiffrer!"
- key_in: La clé secrète K_CSPRNG
- data_in: La valeur du compteur

CSPRNG reçoit de AES_Core:

- data_out: Le résultat chiffré (ta clé aléatoire!)
- busy: "Je suis en train de travailler"
- done: "J'ai terminé!"

```
AES_CSPRNG_Inst : AES_Core
  port map (
    clk      => clk,
    reset    => reset,
    start    => aes_start_o,
    key_in   => K_CSPRNG,                    -- The Fixed Secret Seed
    data_in  => std_logic_vector(Counter),   -- The Incrementing Counter
    data_out => aes_output_data,             -- The Generated Key
    busy     => aes_busy_i,
    done     => aes_done_i
  );
```

Mapping des Sorties

- key_out = toujours la dernière valeur chiffrée par AES
- key_valid = pulse d'1 horloge exactement quand AES finit
- Externe au module: quand tu vois key_valid = 1, tu sais qu'il y a une nouvelle clé à key_out

```
key_out   <= aes_output_data;
key_valid <= aes_done_i;
```

Logique de Contrôle :

```
process(clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
      Counter   <= (others => '0');
      aes_start_o <= '0';
    else
      aes_start_o <= '0';
      if aes_done_i = '1' then
        Counter <= Counter + 1;
        aes_start_o <= '1';
      elsif Counter = to_unsigned(0, 128) and aes_busy_i = '0' then
        aes_start_o <= '1';
      end if;
    end if;
```

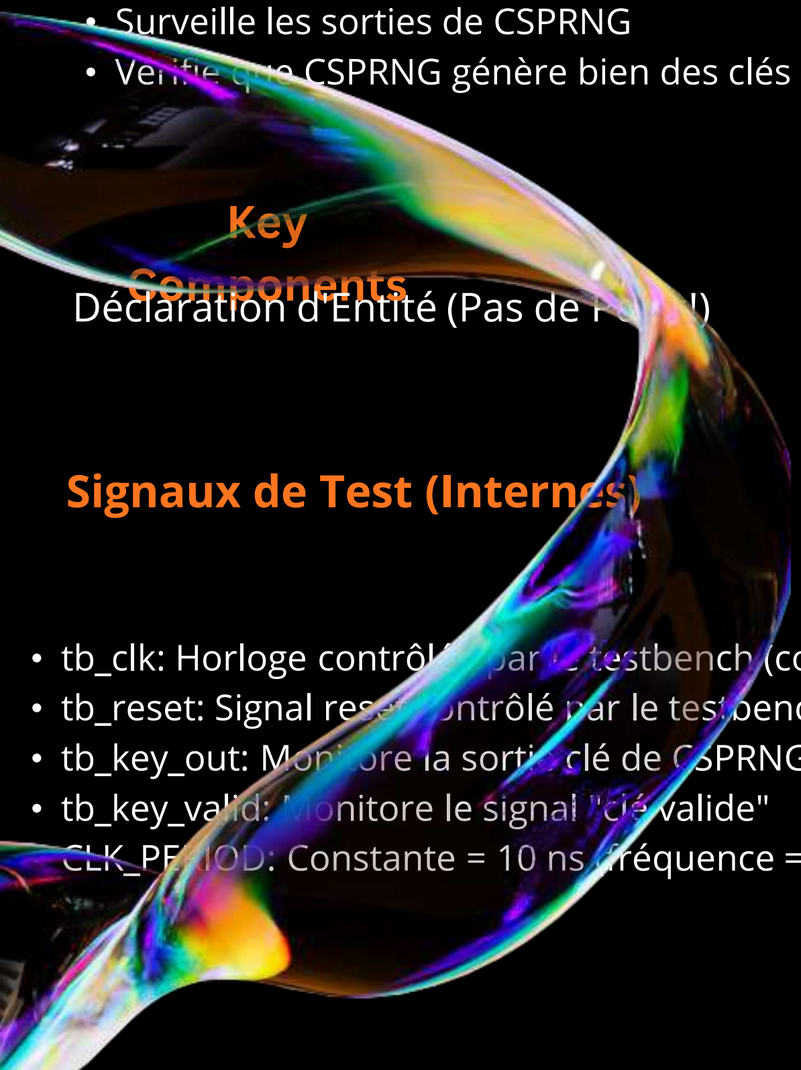
Chronologie Temporelle

Temps (ns)	
	Réinitialisation tenue
0	Reset libéré
50	aes_start_0 pulse (démarrage)
60	AES chiffre (22 horloges)
70	AES finit (done=1 pulse)
89-220	key_valid=1, key_out=CLÉ1
	Counter++ (devient 1)
220	Counter++ (devient 1)
	aes_start_0 pulse (relance)
240	AES commence pour CLÉ2
330	AES finit (done=1 pulse)
	key_valid=1, key_out=CLÉ2
440	Counter++ (devient 2)
	AES finit (done=1 pulse)

Code 2 : CSPRNG_tb (Le Banc de Test)

CSPRNG_tb est un MODULE DE TEST qui:

- Crée un environnement de simulation
- Génère l'horloge et le signal reset
- Surveille les sorties de CSPRNG
- Vérifie que CSPRNG génère bien des clés différentes



Key Components

Déclaration d'Entité (Pas de Port)

```
entity CSPRNG_tb is
end entity CSPRNG_tb;
```

Signaux de Test (Internes)

- tb_clk: Horloge contrôlée par le testbench (commence à '0')
- tb_reset: Signal reset contrôlé par le testbench (commence à '1')
- tb_key_out: Monitore la sortie clé de CSPRNG
- tb_key_valid: Monitore le signal "clé valide"
- CLK_PERIOD: Constante = 10 ns (fréquence = 100 MHz)

```
signal tb_clk       : std_logic := '0';
signal tb_reset     : std_logic := '1';
signal tb_key_out   : std_logic_vector(127 downto 0);
signal tb_key_valid : std_logic;
```

Instantiation du DUT (Device Under Test = CSPRNG)

- Crée une instance de CSPRNG
- Relie les signaux de test aux ports de CSPRNG
- Maintenant on peut contrôler CSPRNG depuis le testbench!

```
DUT : CSPRNG
  port map (
    clk       => tb_clk,
    reset     => tb_reset,
    key_out   => tb_key_out,
    key_valid => tb_key_valid
  );
```

```
constant CLK_PERIOD : time := 10 ns;
```

Génération d'Horloge

Génération d'horloge : Boucle infinie qui toggle tb_clk toutes les 5 ns.

```
CLK_GEN : process
begin
  loop
    wait for CLK_PERIOD / 2;
    tb_clk <= not tb_clk;
  end loop;
end process CLK_GEN;
```

Génération de Stimulus (Scénario de Test)

-- Step 1: Initialization and Reset du Pulse

```
report "--- Starting CSPRNG Test Bench Simulation ---" severity NOTE;
tb_reset <= '1';
wait for CLK_PERIOD * 5;

tb_reset <= '0';
report "Reset deasserted. CSPRNG should start first key generation." severity NOTE;
```

-- Step 2: Monitor la création des clés

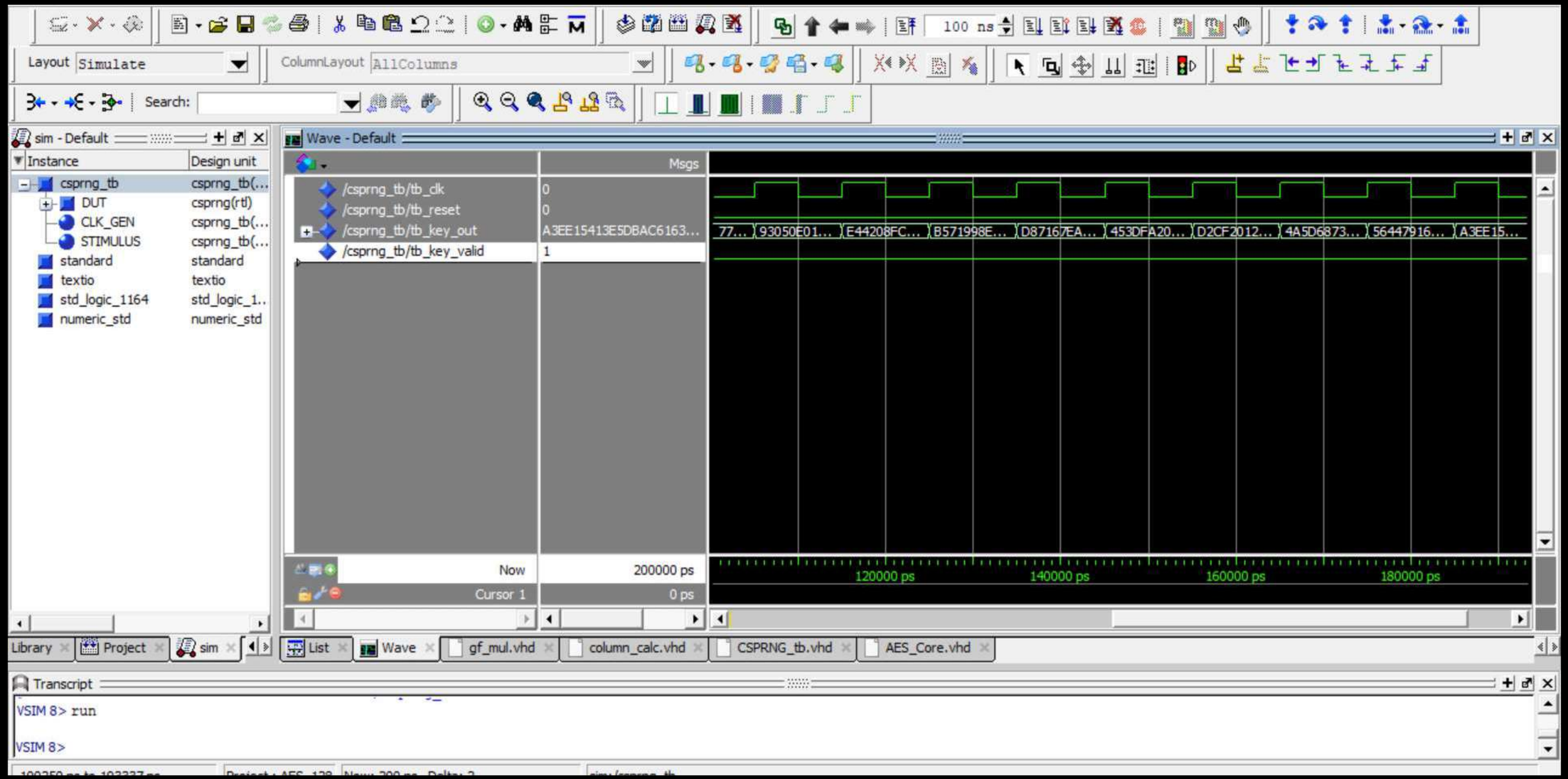
- Attend le premier key_valid → affiche "KEY 1 generated" avec le temps.
- Même pour KEY 2 et KEY 3

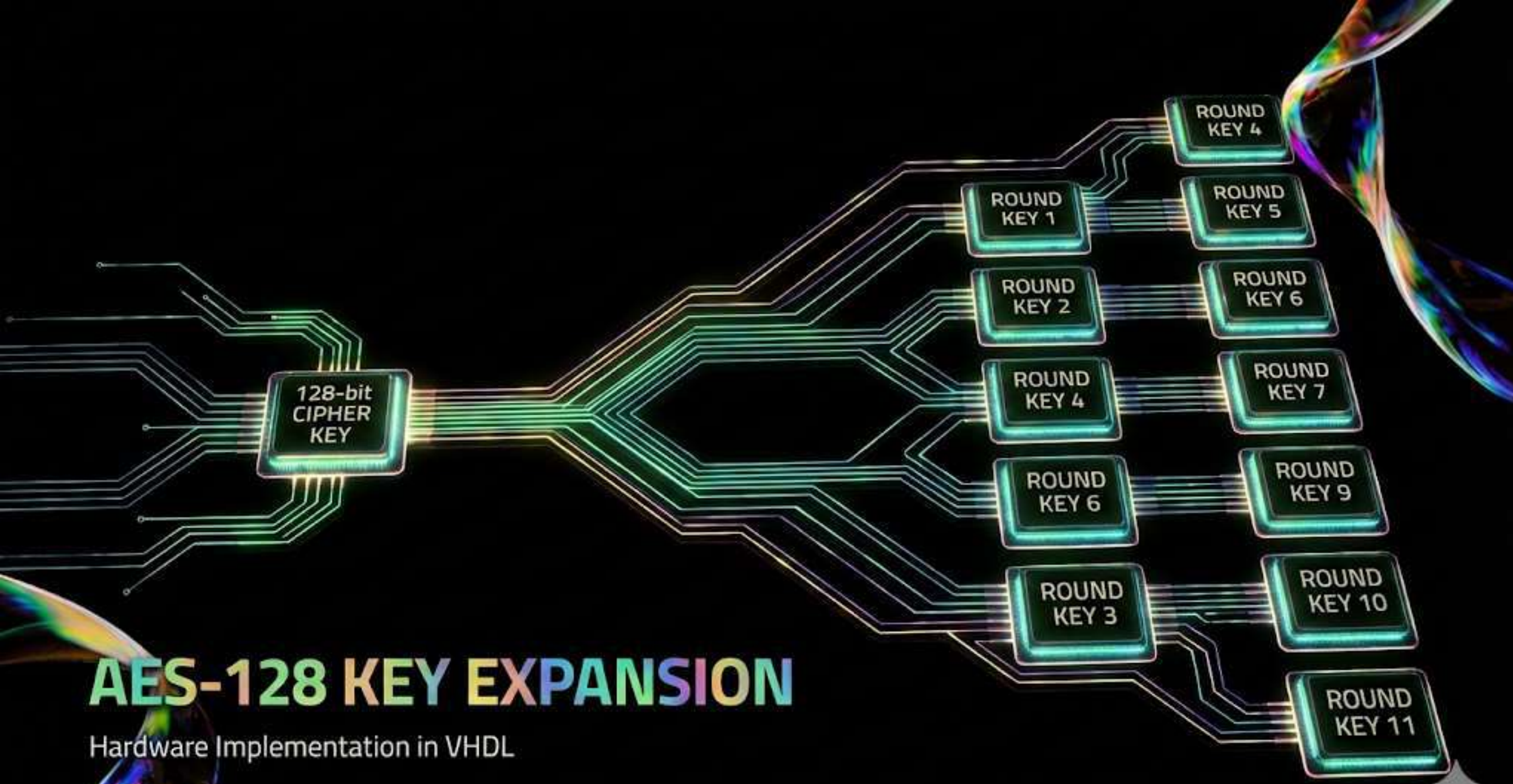
```
-- Wait for the first key to become valid
wait until rising_edge(tb_clk) and tb_key_valid = '1';
report "KEY 1 generated at T=" & integer'image(now / CLK_PERIOD) & ". Check wave viewer (tb_key_out) for the 128-bit hex value." severity NOTE;

-- Wait for the second key to become valid
wait until rising_edge(tb_clk) and tb_key_valid = '1';
report "KEY 2 generated at T=" & integer'image(now / CLK_PERIOD) & ". Check wave viewer (tb_key_out) for the 128-bit hex value." severity NOTE;

-- Wait for the third key to become valid
wait until rising_edge(tb_clk) and tb_key_valid = '1';
report "KEY 3 generated at T=" & integer'image(now / CLK_PERIOD) & ". Check wave viewer (tb_key_out) for the 128-bit hex value." severity NOTE;
```

-- Step 3: End Simulation

```
report "--- Test Complete: 3 keys generated and verified in waves ---" severity NOTE;
wait for CLK_PERIOD * 10;
assert false report "Simulation finished." severity failure;
```

Simulation du testbench





AES-128 KEY EXPANSION

Hardware Implementation in VHDL

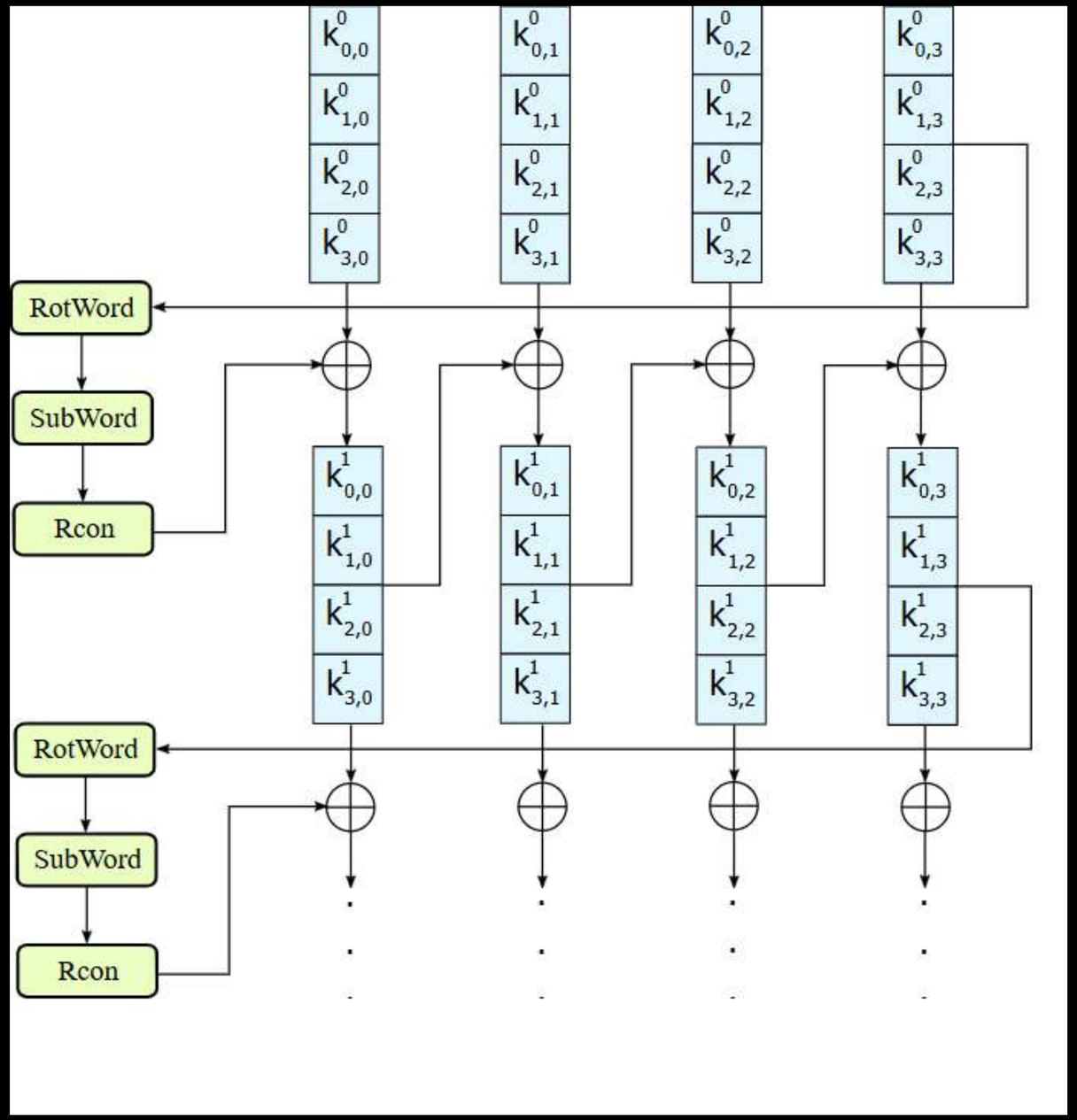
POURQUOI L'EXPANSION DE CLÉ EST IMPORTANT

- Génère une clé différente pour chaque tour AES
- Évite l'utilisation de la même clé à chaque tour
- Répartit l'effet de la clé sur tous les tours
- Rend chaque tour plus sûr et différent

RÔLE D'EXPANSION DE CLÉ DANS AES

- Prend la clé secrète de 128 bits en entrée
- Produit les clés de tour (round keys)
- Fournit les clés à l'opération AddRoundKey

POSITION DANS L'ARCHITECTURE GLOBALE AES



Roundkey 0 (la même clé générée par le CSPRNG)

Roundkey 1

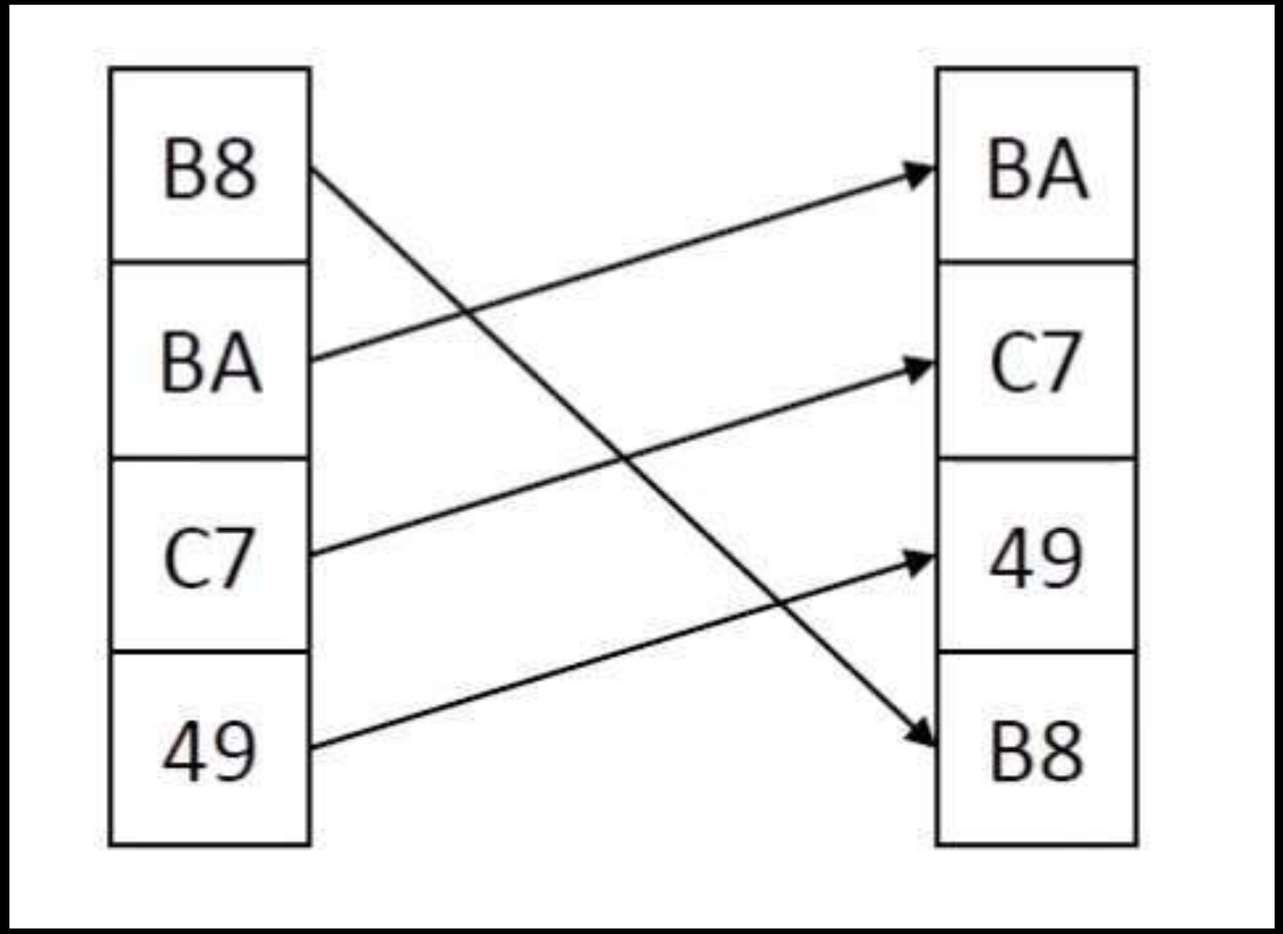
OPÉRATIONS IMPORTANTES D'EXPANSION DE CLÉ

Rotation Word

Substitution Word

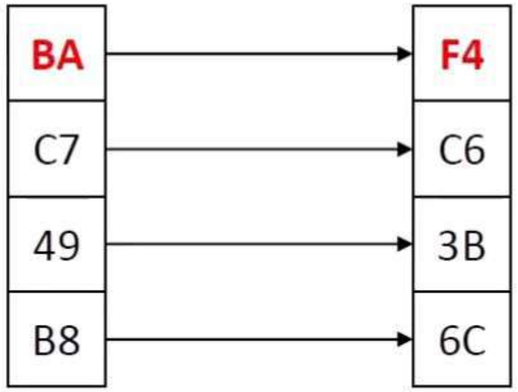
Round Constant

ROTATION WORD



SUBSTITUTION WORD (SUBWORD)

Substitutes a 32 bit word using the AES S-Box



w[0] = S-Box(w[0])
w[1] = S-Box(w[1])
w[2] = S-Box(w[2])
w[3] = S-Box(w[3])

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1_	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2_	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3_	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4_	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5_	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6_	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7_	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8_	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9_	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A_	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B_	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C_	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D_	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E_	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F_	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

AES S-Box

ROUND CONSTANT (RCON)

Round constants are generated using a recursive function

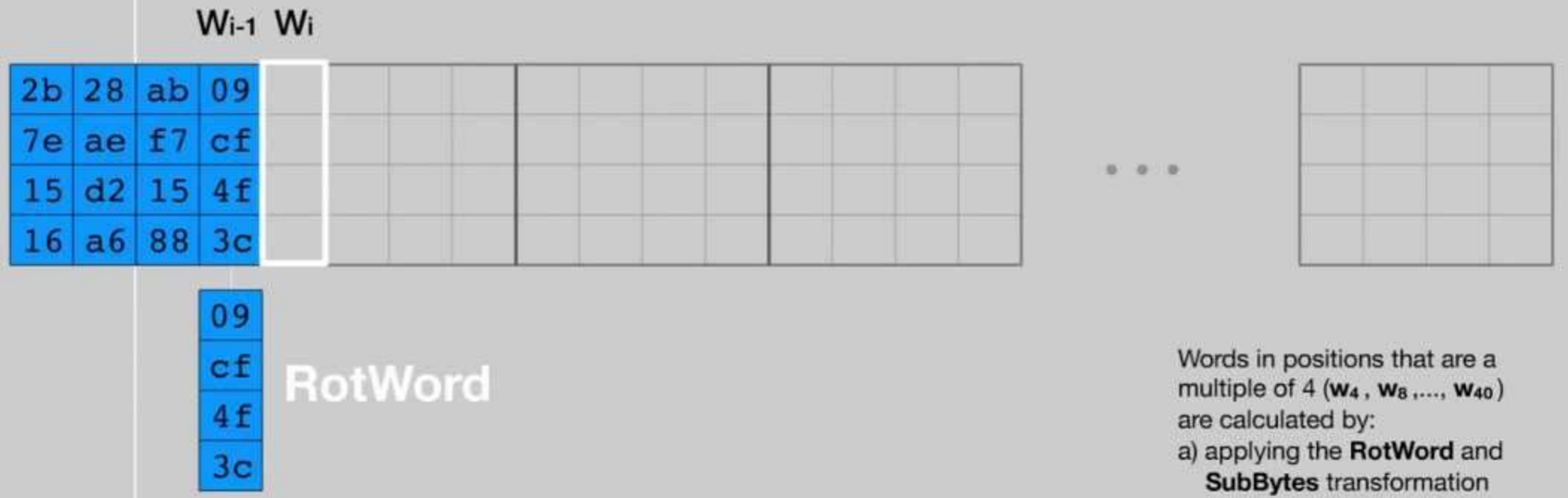
01	02	04	08	10	20	40	80	1B	36	...
00	00	00	00	00	00	00	00	00	00	
00	00	00	00	00	00	00	00	00	00	
00	00	00	00	00	00	00	00	00	00	

$$rc(1) = 1$$

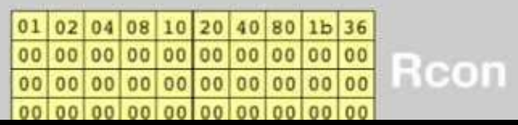
$$rc(i) = 2 \cdot rc(i-1) \quad if\ rc(i-1) < 0x80$$

$$rc(i) = (2 \cdot rc(i-1)) \oplus 0x11B \quad if\ rc(i-1) \geq 0x80$$

Key Schedule



Words in positions that are a multiple of 4 (w_4, w_8,..., w_{40}) are calculated by:
a) applying the **RotWord** and **SubBytes** transformation to the previous word w_{i-1}.



```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
      zakaria-mghili, 4 days ago • last version
zakaria-mghili, 4 days ago | 1 author (zakaria-mghili)
entity KeyExpansion is
    port (
        clk       : in  std_logic;
        reset     : in  std_logic;
        start     : in  std_logic;
        key_in    : in  std_logic_vector(127 downto 0);
        round_key : out std_logic_vector(127 downto 0);
        round     : out integer range 0 to 10;
        valid     : out std_logic;
        done      : out std_logic
    );
end entity KeyExpansion;

zakaria-mghili, 4 days ago | 1 author (zakaria-mghili)
architecture Behavioral of KeyExpansion is

    zakaria-mghili, 4 days ago | 1 author (zakaria-mghili)
    component Sbox is
        port (
            byte_in  : in  std_logic_vector(7 downto 0);
            byte_out : out std_logic_vector(7 downto 0)
        );
    end component;

    type rcon_array is array (1 to 10) of std_logic_vector(31 downto 0);
    constant RCON : rcon_array := (
        x"01000000", x"02000000", x"04000000", x"08000000",
        x"10000000", x"20000000", x"40000000", x"80000000",
        x"1B000000", x"36000000"
    );

    signal w0, w1, w2, w3 : std_logic_vector(31 downto 0);
    signal temp    : std_logic_vector(31 downto 0);
    signal rotated : std_logic_vector(31 downto 0);

    signal sub_byte0, sub_byte1, sub_byte2, sub_byte3 : std_logic_vector(7 downto 0);

    -- ?? Always between 1 and 10 now (safe for RCON index)
    signal current_round : integer range 1 to 10 := 1;
    signal expanding     : std_logic := '0';
```

On importe les bibliothèques IEEE. std_logic_1164 fournit les types std_logic et std_logic_vector (signaux logiques en VHDL). numeric_std sert pour les opérations numériques propres (compteurs, conversions, calculs).

déclaration du composant S-Box chargé d'effectuer l'opération SUBWORD

déclaration du tableau RCON qui contiendra toutes les constantes nécessaires à l'opération Rcon

Déclaration des signaux internes w0 à w3, qui servent à découper key_in afin de faciliter le traitement et l'avancement des différentes étapes.Le signal temp joue le rôle de variable temporaire, tandis que rotated contient le résultat de l'opération RotWord.
Les signaux currentRound et expanding permettent, quant à eux, d'assurer la communication entre notre code et la machine à états (FSM).

```
begin

    -- RotWord
    rotated <= w3(23 downto 0) & w3(31 downto 24);

    -- S-boxes
    sbox0: Sbox port map (byte_in => rotated(31 downto 24), byte_out => sub_byte0);
    sbox1: Sbox port map (byte_in => rotated(23 downto 16), byte_out => sub_byte1);
    sbox2: Sbox port map (byte_in => rotated(15 downto 8),  byte_out => sub_byte2);
    sbox3: Sbox port map (byte_in => rotated(7 downto 0),   byte_out => sub_byte3);

    -- SubWord(RotWord(w3)) XOR Rcon[current_round]
    temp <= (sub_byte0 & sub_byte1 & sub_byte2 & sub_byte3) xor RCON(current_round);
```

```
    process(clk)
    begin
        zakaria-mghili, 4 days ago | 1 author (zakaria-mghili)
        if rising_edge(clk) then
            zakaria-mghili, 4 days ago | 1 author (zakaria-mghili)
            if reset = '1' then
                w0            <= (others => '0');
                w1            <= (others => '0');
                w2            <= (others => '0');
                w3            <= (others => '0');
                current_round <= 1;      -- ?? safe index for RCON
                expanding     <= '0';
                valid         <= '0';
                done          <= '0';
                round_key     <= (others => '0');
                round         <= 0;

            else
                -- Default every cycle
                valid <= '0';
                done  <= '0';
```

L'operation du Rotation Word de le premier mot du key initial (pour tous les clés en generale)

Ici, quatre instances de la S-Box sont utilisées, une pour chaque octet du mot après RotWord. Chaque octet du signal rotated est envoyé séparément vers une S-Box, ce qui permet d'appliquer l'opération SubWord octet par octet.

Ce bloc est un processus synchrone contrôlé par l'horloge. À chaque front montant du signal clk, le key expansion avance d'une étape et met à jour ses registres internes. Cette organisation permet de générer les clés de tour de manière séquentielle et contrôlée dans le temps.

```
if start = '1' and expanding = '0' then
    -- Load initial key
    w0            <= key_in(127 downto 96);
    w1            <= key_in(95  downto 64);
    w2            <= key_in(63  downto 32);
    w3            <= key_in(31  downto 0);

    current_round <= 1;   -- next round will use RCON(1)
    expanding     <= '1';

    -- Output round 0 key
    round_key <= key_in;
    round     <= 0;
    valid     <= '1';
elsif expanding = '1' then
    -- Compute new round key (round 1..10)
    w0 <= w0 xor temp;
    w1 <= w1 xor (w0 xor temp);
    w2 <= w2 xor (w1 xor (w0 xor temp));
    w3 <= w3 xor (w2 xor (w1 xor (w0 xor temp)));

    round_key <= (w0 xor temp) &
                 (w1 xor (w0 xor temp)) &
                 (w2 xor (w1 xor (w0 xor temp))) &
                 (w3 xor (w2 xor (w1 xor (w0 xor temp))));
    round     <= current_round;
    valid     <= '1';

    zakaria-mghili, 4 days ago | 1 author (zakaria-mghili)
```

```
if current_round < 10 then
    current_round <= current_round + 1;
else
    signal expanding : std_logic := '0';
    expanding <= '0';
    done      <= '1';
end if;
```

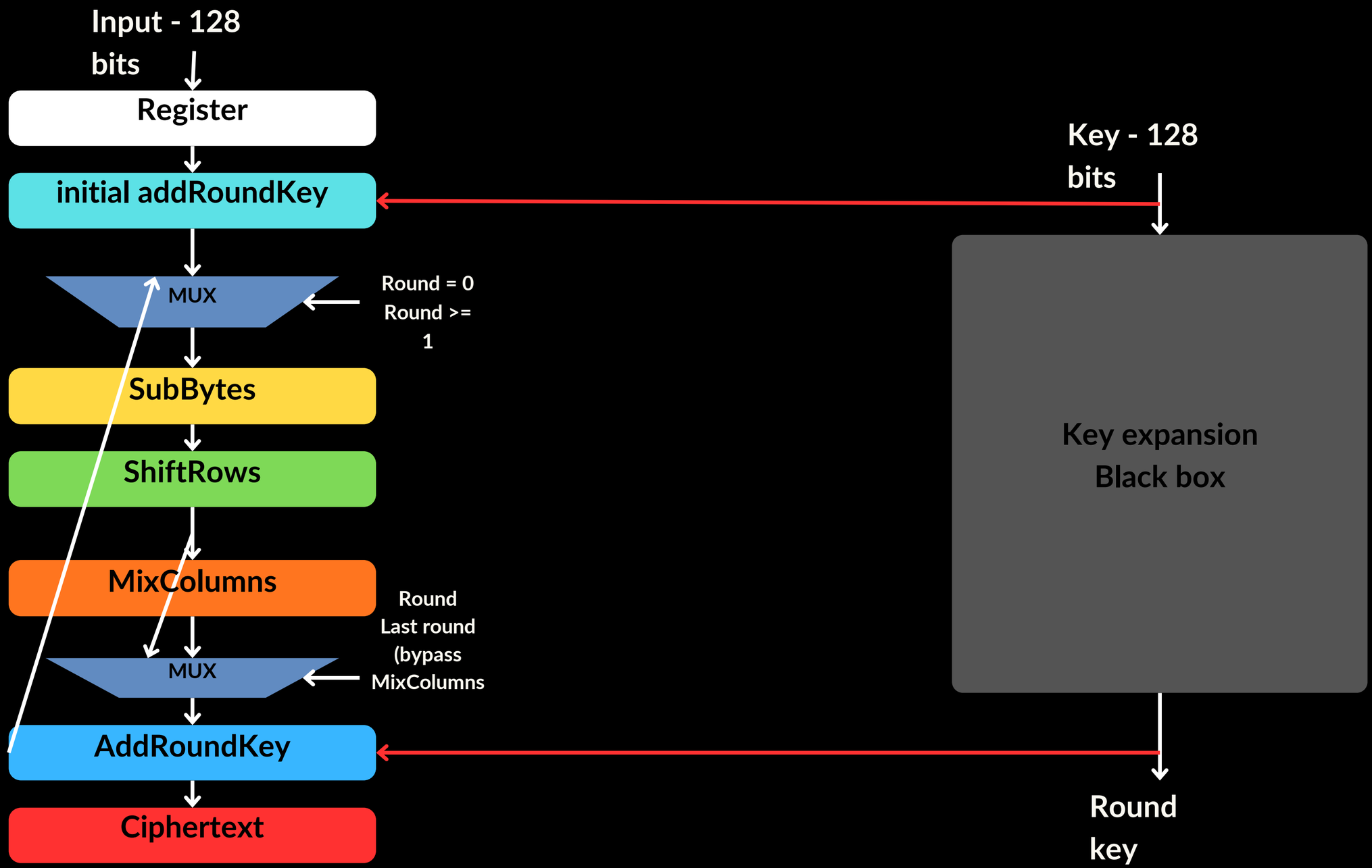
Diviser key_in en quatre mots afin de pouvoir commencer les opérations

appliquer les opérations de XOR entre ces quatre mots, puis commencer l'envoi des round keys vers le datapath

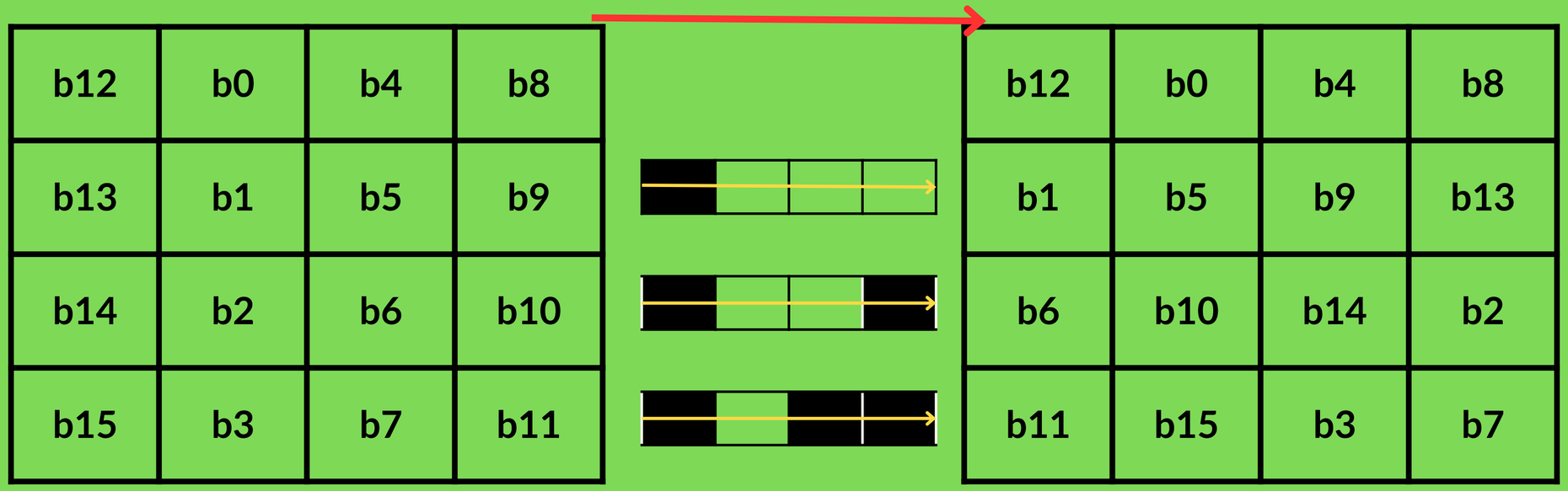
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

zakaria-mghili, 6 days ago | 1 author (zakaria-mghili)
entity Sbox is
    port (
        byte_in  : in  std_logic_vector(7 downto 0);   -- Input byte to substitute
        byte_out : out std_logic_vector(7 downto 0)    -- Substituted output byte
    );
end entity Sbox;

zakaria-mghili, 6 days ago | 1 author (zakaria-mghili)
architecture Combinational of Sbox is

    -- AES S-box constant lookup table (exact standard values, no computation).
    -- This is a ROM in hardware ? super efficient on FPGAs (uses LUTs or Block RAM if big).
    -- Indexed by to_integer(unsigned(byte_in)).
    type sbox_array is array (0 to 255) of std_logic_vector(7 downto 0);
    constant SBOX : sbox_array := (
        x"63", x"7c", x"77", x"7b", x"f2", x"6b", x"6f", x"c5", x"30", x"01", x"67", x"2b", x"fe", x"d7", x"ab", x"76",
        x"ca", x"82", x"c9", x"7d", x"fa", x"59", x"47", x"f0", x"ad", x"d4", x"a2", x"af", x"9c", x"a4", x"72", x"c0",
        x"b7", x"fd", x"93", x"26", x"36", x"3f", x"f7", x"cc", x"34", x"a5", x"e5", x"f1", x"71", x"d8", x"31", x"15",
        x"04", x"c7", x"23", x"c3", x"18", x"96", x"05", x"9a", x"07", x"12", x"80", x"e2", x"eb", x"27", x"b2", x"75",
        x"09", x"83", x"2c", x"1a", x"1b", x"6e", x"5a", x"a0", x"52", x"3b", x"d6", x"b3", x"29", x"e3", x"2f", x"84",
        x"53", x"d1", x"00", x"ed", x"20", x"fc", x"b1", x"5b", x"6a", x"cb", x"be", x"39", x"4a", x"4c", x"58", x"cf",
        x"d0", x"ef", x"aa", x"fb", x"43", x"4d", x"33", x"85", x"45", x"f9", x"02", x"7f", x"50", x"3c", x"9f", x"a8",
        x"51", x"a3", x"40", x"8f", x"92", x"9d", x"38", x"f5", x"bc", x"b6", x"da", x"21", x"10", x"ff", x"f3", x"d2",
        x"cd", x"0c", x"13", x"ec", x"5f", x"97", x"44", x"17", x"c4", x"a7", x"7e", x"3d", x"64", x"5d", x"19", x"73",
        x"60", x"81", x"4f", x"dc", x"22", x"2a", x"90", x"88", x"46", x"ee", x"b8", x"14", x"de", x"5e", x"0b", x"db",
        x"e0", x"32", x"3a", x"0a", x"49", x"06", x"24", x"5c", x"c2", x"d3", x"ac", x"62", x"91", x"95", x"e4", x"79",
        x"e7", x"c8", x"37", x"6d", x"8d", x"d5", x"4e", x"a9", x"6c", x"56", x"f4", x"ea", x"65", x"7a", x"ae", x"08",
        x"ba", x"78", x"25", x"2e", x"1c", x"a6", x"b4", x"c6", x"e8", x"dd", x"74", x"1f", x"4b", x"bd", x"8b", x"8a",
        x"70", x"3e", x"b5", x"66", x"48", x"03", x"f6", x"0e", x"61", x"35", x"57", x"b9", x"86", x"c1", x"1d", x"9e",
        x"e1", x"f8", x"98", x"11", x"69", x"d9", x"8e", x"94", x"9b", x"1e", x"87", x"e9", x"ce", x"55", x"28", x"df",
        x"8c", x"a1", x"89", x"0d", x"bf", x"e6", x"42", x"68", x"41", x"99", x"2d", x"0f", x"b0", x"54", x"bb", x"16"
    );

begin

    -- Pure combinational lookup ? no delay in simulation, but in hardware it's LUT delay (~1-2 ns).
    -- Use unsigned conversion for index ? safe and standard.
    byte_out <= SBOX(to_integer(unsigned(byte_in)));

end architecture Combinational;
```

Déclaration du S-Box complet sous forme de tableau. Ensuite, nous combinons plusieurs commandes : unsigned pour garantir que la valeur reste positive, et to_integer pour convertir cette valeur en entier afin d'effectuer un accès (lookup) à l'indice correspondant.

AES-128 ENCRYPTION — DATAPATH ARCHITECTURE & HARDWARE IMPLEMENTATION


Input - 128 bits
Register
initial addRoundKey
MUX
Round = 0
Round >= 1
SubBytes
ShiftRows
MixColumns
MUX
Round
Last round
(bypass
MixColumns
AddRoundKey
Ciphertext
Key - 128 bits
Key expansion
Black box
Round key

ShiftRows



L'opération ShiftRows consiste à effectuer une permutation fixe des octets ligne par ligne :

- la première ligne n'est pas décalée,
- la deuxième ligne est décalée circulairement d'un octet vers la gauche,
- la troisième ligne est décalée de deux octets,
- et la quatrième ligne est décalée de trois octets.

ShiftRows

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity ShiftRows is
    port ( input: in std_logic_vector(127 downto 0); output: out std_logic_vector(127 downto 0) );
end ShiftRows;

architecture Beha of ShiftRows is
begin
    -- to find index we do , 127 - 8xi | i is number of element in martix
    -- | b0  b4  b8   b12 |
    -- | b1  b5  b9   b13 |
    -- | b2  b6  b10  b14 |
    -- | b3  b7  b11  b15 |

    -- row 1
    output(127 downto 120) <= input(127 downto 120);
    output(95 downto 88) <= input(95 downto 88);
    output(63 downto 56) <= input(63 downto 56);
    output(31 downto 24) <= input(31 downto 24);
    -- row 2
    output(119 downto 112) <= input(87 downto 80);
    output(87 downto 80) <= input(55 downto 48);
    output(55 downto 48) <= input(23 downto 16);
    output(23 downto 16) <= input(119 downto 112);
    -- row 3
    output(111 downto 104) <= input(47 downto 40);
    output(79 downto 72) <= input(15 downto 8);
    output(47 downto 40) <= input(111 downto 104);
    output(15 downto 8) <= input(79 downto 72);
    --row 4
    output(103 downto 96) <= input(7 downto 0);
    output(71 downto 64) <= input(103 downto 96);
    output(39 downto 32) <= input(71 downto 64);
    output(7 downto 0) <= input(39 downto 32);
end Beha;
```

L'entrée est l'état AES de 128 bits, organisé logiquement comme une matrice de 4 lignes et 4 colonnes

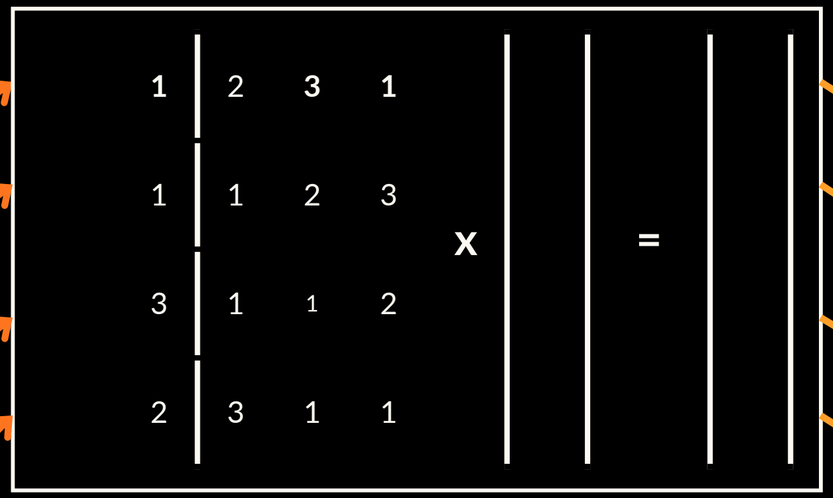
Cette partie correspond à la première ligne de la matrice AES.
Selon l'algorithme AES, la première ligne n'est pas décalée.
Les octets sont donc recopiés directement de l'entrée vers la sortie

Cette partie implémente le décalage cyclique à gauche d'un octet pour la deuxième ligne.
Chaque octet est déplacé d'une position vers la gauche, avec un retour circulaire du premier élément en fin de ligne

Cette partie correspond à la troisième ligne de la matrice AES

Cette partie implémente le décalage cyclique à gauche de trois octets pour la quatrième ligne

MixColumns



b12	b0	b4	b8
b13	b1	b5	b9
b14	b2	b6	b10
b15	b3	b7	b11

Arithmetic is done in $GF(2^8)$

b'12	b'0	b'4	b'8
b'13	b'1	b'5	b'9
b'14	b'2	b'6	b'10
b'15	b'3	b'7	b'11

MixColumns

Multiplication dans $GF(2^8)$

En AES, toutes les opérations sur les octets sont effectuées dans le corps fini $GF(2^8)$

Dans l'opération MixColumns, la multiplication générale n'est pas nécessaire. Seules les multiplications par 01, 02 et 03 sont requises

La multiplication par 01 laisse l'octet inchangé

La multiplication par 02 est réalisée par un décalage à gauche d'un bit. Si le bit de poids fort de l'octet initial est égal à 1, on fait un XOR avec la constante 0x1B

La multiplication par 03 est réalisée par le XOR entre le résultat de la multiplication par 02 et l'octet original

- 3 * a = (2*a) XOR a

MixColumns

Multiplication dans $GF(2^8)$

```vhdl
library ieee;
use ieee.STD_LOGIC_1164.all;

entity gf_mul is
    port (
        byte_int : in std_logic_vector(7 downto 0);
        mul2_out: out std_logic_vector(7 downto 0);
        mul3_out: out std_logic_vector(7 downto 0)
    );
end gf_mul;

architecture Behavioral of gf_mul is

    function mul2 (x : std_logic_vector(7 downto 0)) return std_logic_vector is
    begin
        if (x(7) = '0') then
            return x(6 downto 0) & '0';
        else
            return ((x(6 downto 0) & '0') xor x"1B" );
        end if;
    end mul2;

    function mul3 (x : std_logic_vector(7 downto 0)) return std_logic_vector is
    begin
        return mul2(x) xor x;
    end mul3;
begin
    mul2_out <= mul2(byte_int);
    mul3_out <= mul3(byte_int);

end Behavioral;
```

la bibliothèque IEEE et le package STD_LOGIC_1164, qui permet d'utiliser les types std_logic et std_logic_vector. Ces types sont nécessaires pour représenter les données binaires dans le circuit

entité définit un module de multiplication dans le corps fini $GF(2^8)$.
Le module reçoit en entrée un octet de 8 bits, appelé byte_int.
En sortie, il fournit deux résultats :
– mul2_out, qui correspond à la multiplication par 02,
– et mul3_out, qui correspond à la multiplication par 03.

La fonction mul2 implémente la multiplication d'un octet par 02 dans $GF(2^8)$

La fonction mul3 implémente la multiplication d'un octet par 03 dans $GF(2^8)$

MixColumns

Code MixColumns pour une colonne AES

Ces signaux stockent les résultats intermédiaires des multiplications par 02 et par 03 pour chacun des quatre octets de la colonne

Déclaration du composant gf_mul est utilisé pour effectuer les multiplications dans GF(2^8)

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.ALL;
3
4   ENTITY Column_Calc IS
5       PORT (
6           input : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
7           output : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
8       );
9   END Column_Calc;
10
11  ARCHITECTURE Beha OF Column_Calc IS
12      SIGNAL temp0_2 : STD_LOGIC_VECTOR(7 DOWNTO 0);
13      SIGNAL temp0_3 : STD_LOGIC_VECTOR(7 DOWNTO 0);
14
15      SIGNAL temp1_2 : STD_LOGIC_VECTOR(7 DOWNTO 0);
16      SIGNAL temp1_3 : STD_LOGIC_VECTOR(7 DOWNTO 0);
17
18      SIGNAL temp2_2 : STD_LOGIC_VECTOR(7 DOWNTO 0);
19      SIGNAL temp2_3 : STD_LOGIC_VECTOR(7 DOWNTO 0);
20
21      SIGNAL temp3_2 : STD_LOGIC_VECTOR(7 DOWNTO 0);
22      SIGNAL temp3_3 : STD_LOGIC_VECTOR(7 DOWNTO 0);
23
24      COMPONENT gf_mul
25          PORT (
26              byte_int : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
27              mul2_out : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
28              mul3_out : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
29          );
30      END COMPONENT;
31
32  BEGIN
```

MixColumns

Code MixColumns pour une colonne AES

```
32  BEGIN
33
34      gf_mul_b0 : gf_mul
35      PORT MAP(
36          byte_int => input(31 DOWNTO 24),
37          mul2_out => temp0_2,
38          mul3_out => temp0_3
39      );
40
41      gf_mul_b1 : gf_mul
42      PORT MAP(
43          byte_int => input(31 - 8 DOWNTO 24 - 8),
44          mul2_out => temp1_2,
45          mul3_out => temp1_3
46      );
47
48      gf_mul_b2 : gf_mul
49      PORT MAP(
50          byte_int => input(31 - 2 * 8 DOWNTO 24 - 8 * 2),
51          mul2_out => temp2_2,
52          mul3_out => temp2_3
53      );
54
55      gf_mul_b3 : gf_mul
56      PORT MAP(
57          byte_int => input(31 - 3 * 8 DOWNTO 24 - 3 * 8),
58          mul2_out => temp3_2,
59          mul3_out => temp3_3
60      );
61
62      output(31 DOWNTO 24) <= temp0_2 XOR temp1_3 XOR input(31 - 2 * 8 DOWNTO 24 - 8 * 2) XOR input(31 - 8 * 3 DOWNTO 24 - 8 * 3);
63      output(31 - 8 DOWNTO 24 - 8) <= input(31 DOWNTO 24) XOR temp1_2 XOR temp2_3 XOR input(31 - 8 * 3 DOWNTO 24 - 8 * 3);
64      output(31 - 2 * 8 DOWNTO 24 - 2 * 8) <= input(31 DOWNTO 24) XOR input(31 - 8 DOWNTO 24 - 8) XOR temp2_2 XOR temp3_3;
65      output(31 - 3 * 8 DOWNTO 24 - 3 * 8) <= temp0_3 XOR input(31 - 8 DOWNTO 24 - 8) XOR input(31 - 2 * 8 DOWNTO 24 - 8 * 2) XOR temp3_2;
66  END Beha;
```

Ces blocs instancient le module gf_mul pour chacun des quatre octets de la colonne AES
L'octet d'entrée correspond aux bits 31−8i à 24−8i du vecteur input, où i représente l'indice de l'octet dans la colonne

Ces lignes de code réalisent le calcul de la sortie de l'opération MixColumns pour une colonne AES.
Chaque octet de sortie est obtenu par une combinaison linéaire des quatre octets d'entrée de la colonne, conformément à la matrice MixColumns définie par la norme AES

MixColumns

Code MixColumns

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY MixColumns IS
    PORT (
        input : IN STD_LOGIC_VECTOR(127 DOWNTO 0);
        output : OUT STD_LOGIC_VECTOR(127 DOWNTO 0)
    );
END MixColumns;

ARCHITECTURE Beha OF MixColumns IS
    COMPONENT Column_Calc
        PORT (
            input : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
            output : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
        );
    END COMPONENT;

BEGIN
    calcul_column_1 : Column_Calc
        PORT MAP(
            input => input(127  DOWNTO 120 - 3*8),
            output => output(127 DOWNTO 120 - 3*8)
        );
    calcul_column_2 : Column_Calc
        PORT MAP(
            input => input(127 - 4*8 DOWNTO 120 - 7*8),
            output => output(127 - 4*8 DOWNTO 120 - 7*8)
        );

    calcul_column_3 : Column_Calc
        PORT MAP(
            input => input(127 - 8*8 DOWNTO 120 - 11*8),
            output => output(127 - 8*8 DOWNTO 120 - 11*8)
        );

    calcul_column_4 : Column_Calc
        PORT MAP(
            input => input(127 - 12*8 DOWNTO 120 - 15*8),
            output => output(127 - 12*8 DOWNTO 120 - 15*8)
        );

END Beha;
```

Le module MixColumns implémente l'opération MixColumns sur l'état AES complet, L'entrée correspond à l'état AES avant MixColumns, et la sortie correspond à l'état après l'application de cette transformation

Le composant Column_Calc réalise l'opération MixColumns pour une seule colonne AES de 32 bits.
Le module MixColumns va donc instancier ce composant quatre fois, une fois par colonne

Ces instanciations permettent d'appliquer l'opération MixColumns à l'état AES complet. L'état d'entrée de 128 bits est découpé en quatre colonnes indépendantes de 32 bits.
Chaque colonne est ensuite traitée par une instance du module Column_Calc, qui réalise la transformation MixColumns pour une seule colonn

AES Round Datapath — Intégration des Composants

Intégration des Composants

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY aes_round IS
    PORT (
        clk : IN STD_LOGIC;
        rst_n : IN STD_LOGIC; -- add this! (active low)
        ena : IN STD_LOGIC; -- pulse or high every round
        round_count : IN UNSIGNED(3 DOWNTO 0); -- 0 to 10
        input_data : IN STD_LOGIC_VECTOR(127 DOWNTO 0); -- plaintext or previous state
        round_key : IN STD_LOGIC_VECTOR(127 DOWNTO 0);
        output_data : OUT STD_LOGIC_VECTOR(127 DOWNTO 0);
        round_done : OUT STD_LOGIC -- one-cycle pulse
    );
END ENTITY;
```

- clk : **horloge du système. Tout ce qui est dans le PROCESS change sur front montant**
- rst_n : **reset actif à 0**
- ena : **signal d'activation. Quand ena='1', on "valide" un round et on enregistre la sortie**

- round_count : **compteur de round (0→10).**
  - **0 : AddRoundKey initial**
  - **1..9 : rounds normaux**
  - **10 : round final (pas de MixColumns)**
- input_data : **l'état AES 128 bits en entrée (plaintext au début ou état précédent).**
- round_key : **clé de round 128 bits (key schedule)**

- output_data : **état de sortie enregistré (utilisé ensuite comme feedback state ou ciphertext final)**
- round_done **est un signal de synchronisation qui indique qu'un round AES vient d'être complété. Il est activé pendant un seul cycle d'horloge lorsque la sortie du round est validée**

Intégration des Composants

```
18  ARCHITECTURE rtl OF aes_round IS
19
20      SIGNAL sb_out : STD_LOGIC_VECTOR(127 DOWNTO 0);
21      SIGNAL sr_out : STD_LOGIC_VECTOR(127 DOWNTO 0);
22      SIGNAL mc_out : STD_LOGIC_VECTOR(127 DOWNTO 0);
23      SIGNAL round_out : STD_LOGIC_VECTOR(127 DOWNTO 0);
24
25  BEGIN
26
27      -- Transformations
28      u1 : ENTITY work.subByte PORT MAP (input_data => input_data, output_data => sb_out);
29      u2 : ENTITY work.shiftRows PORT MAP (input => sb_out, output => sr_out);
30      u3 : ENTITY work.mixColumns PORT MAP (input => sr_out, output => mc_out);
31
32      -- Correct round logic
33      round_out <=
34          input_data XOR round_key WHEN round_count = 0 ELSE -- initial AddRoundKey
35          sr_out XOR round_key WHEN round_count = 10 ELSE -- final round → no MixColumns
36          mc_out XOR round_key; -- rounds 1-9
37
38      -- Register output every time we have enable
39      PROCESS (clk)
40      BEGIN
41          IF rising_edge(clk) THEN
42              IF ena = '1' THEN
43                  output_data <= round_out;
44                  round_done <= '1';
45              ELSE
46                  round_done <= '0';
47              END IF;
48          END IF;
49      END PROCESS;
50
51  END rtl;
```

- sb_out **: sortie de SubBytes (128 bits)**
- sr_out **: sortie de ShiftRows**
- mc_out **: sortie de MixColumns**
- round_out **: résultat final du round, avant enregistrement.**

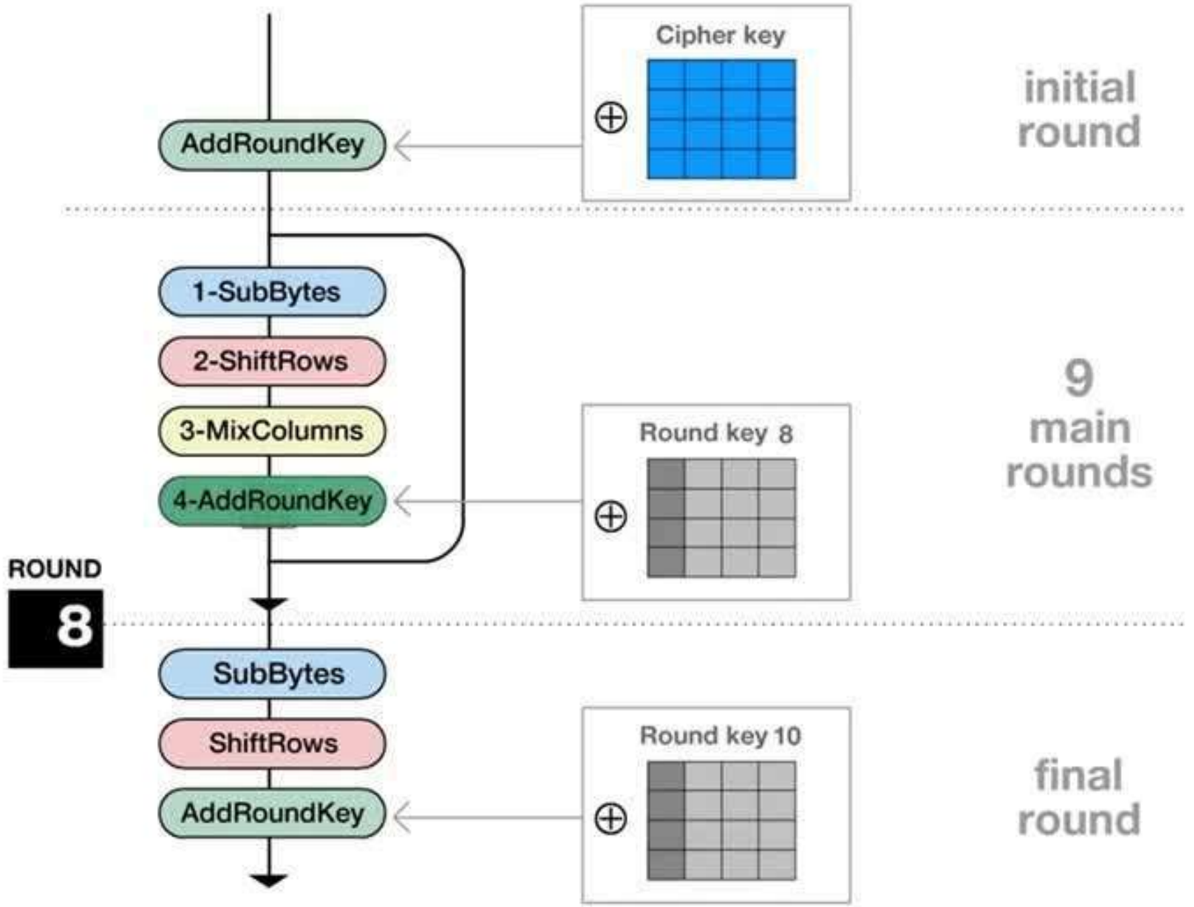
- ENTITY work.xxx **: instanciation directe d'une entité dans la librairie work**
- u1 (SubBytes) **: prend input_data et produit sb_out.**
- u2 (ShiftRows) **: prend sb_out et produit sr_out.**
- u3 (MixColumns) **: prend sr_out et produit mc_out.**

- **Si round_count = 0 : on fait AddRoundKey initial** input_data XOR round_key **(dans AES, avant d'entrer dans le round 1, on XORe avec la clé 0)**
- **Si round_count = 10 : round final , pas de MixColumns, donc on utilise** sr_out XOR round_ke
- **Sinon (1..9) : round normal, on utilise mc_out XOR round_key**

À chaque front montant :

- **si** ena='1' **: on enregistre round_out dans output_data (sortie stable jusqu'au prochain enable), on met** round_done='1' **(pulse 1 cycle)**
- **sinon :** round_done='0'

Encryption Process



FSM (Finite State Machine) – Contrôle et Synchronisation du chiffrement AES

C’est quoi un FSM ?

- C’est un mécanisme de contrôle qui permet d’organiser le fonctionnement d’un système complexe en plusieurs états, et de définir quand et comment on passe d’un état à un autre
- Chaque état correspond à une phase précise du fonctionnement, et les transitions dépendent de conditions ou de signaux

Rôle du FSM dans AES_Core

- Dans notre implémentation AES, on a plusieurs blocs importants :
  - le bloc de Key Expansion
  - le bloc AES Roun
  - et les registres de

données

- Ces blocs doivent être synchronisés correctement.
- La FSM sert donc à coordonner ces blocs, à lancer le chiffrement, à enchaîner les rounds, et à indiquer quand le résultat final est prêt

États principaux du FSM

- IDLE : attente d’une nouvelle demande
- RUN : chiffrement en cours (La Key Expansion génère les clés de round, et à chaque fois qu’une clé est valide, le bloc aes_round est activé pour calculer un round)
- DONE_STATE: Lorsque le dernier round est terminé, la FSM passe à l’état DONE_STATE

FSM (Finite State Machine) –AES_Core.vhd

```vhdl
entity AES_Core is
    port (
        clk      : in  std_logic;
        reset    : in  std_logic;                      -- active high
        start    : in  std_logic;                      -- from global FSM
        key_in   : in  std_logic_vector(127 downto 0); -- master key
        data_in  : in  std_logic_vector(127 downto 0); -- plaintext

        data_out : out std_logic_vector(127 downto 0); -- ciphertext
        busy     : out std_logic;
        done     : out std_logic
    );
end entity AES_Core;

zakaria-mghili, 6 days ago | 1 author (zakaria-mghili)
architecture rtl of AES_Core is

    ------------------------------------------------------------------
    --  Component declarations (your existing blocks)
    ------------------------------------------------------------------
    zakaria-mghili, 6 days ago | 1 author (zakaria-mghili)
    component KeyExpansion is
        port (
            clk       : in  std_logic;
            reset     : in  std_logic;
            start     : in  std_logic;
            key_in    : in  std_logic_vector(127 downto 0);
            round_key : out std_logic_vector(127 downto 0);
            round     : out integer range 0 to 10;
            valid     : out std_logic;
            done      : out std_logic
        );
    end component;
       zakaria-mghili, 6 days ago • last version
    zakaria-mghili, 6 days ago | 1 author (zakaria-mghili)
    component aes_round is
        port (
            clk         : in  std_logic;
            rst_n       : in  std_logic;                -- active low
            ena         : in  std_logic;                -- enable one round
            round_count : in  unsigned(3 downto 0);     -- 0..10
            input_data  : in  std_logic_vector(127 downto 0);
            round_key   : in  std_logic_vector(127 downto 0);
            output_data : out std_logic_vector(127 downto 0);
            round_done  : out std_logic                 -- (we don't use it)
        );
    end component;
```

Déclaration des entrées et sorties de la FSM, qui est constituée de quatre signaux importants : data_in et data_out pour le transfert des données, ainsi que busy et done pour assurer la communication avec la FSM générale.

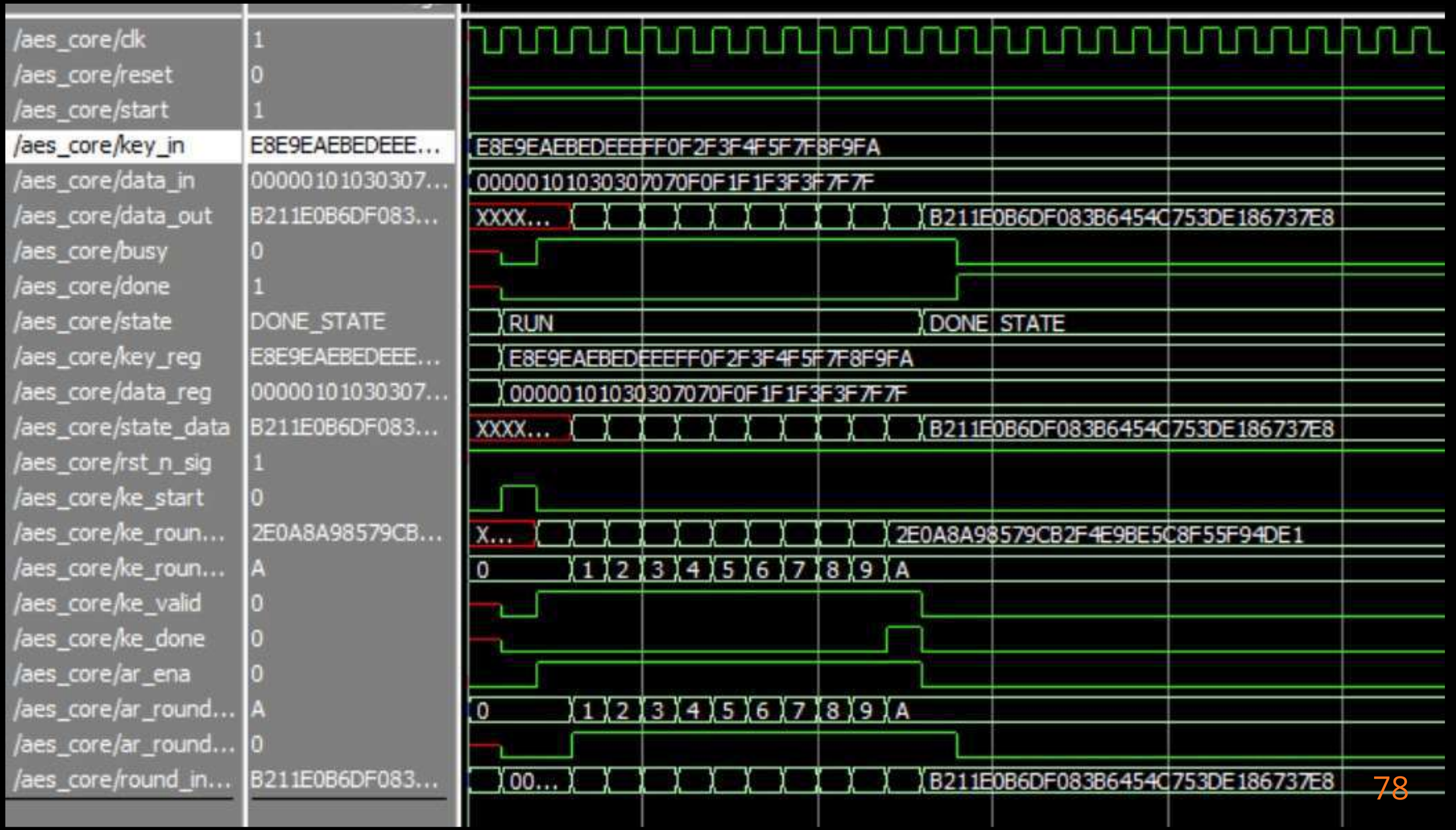
Déclaration des composants key_expansion et aes_round, qui constituent le cœur de l'ensemble des opérations.

```
-- ------------------------------------------------------------------
--  Local FSM state
-- ------------------------------------------------------------------
type state_t is (IDLE, RUN, DONE_STATE);
signal state : state_t := IDLE;

-- ------------------------------------------------------------------
--  Internal registers
-- ------------------------------------------------------------------
signal key_reg   : std_logic_vector(127 downto 0) := (others => '0');
signal data_reg  : std_logic_vector(127 downto 0) := (others => '0');

-- AES state (output of aes_round, fed back as next input)
signal state_data : std_logic_vector(127 downto 0) := (others => '0');

-- Active-low reset for aes_round
signal rst_n_sig : std_logic;

-- ------------------------------------------------------------------
--  KeyExpansion <-> controller signals
-- ------------------------------------------------------------------
signal ke_start     : std_logic := '0';
signal ke_round_key : std_logic_vector(127 downto 0);
signal ke_round_int : integer range 0 to 10;
signal ke_valid     : std_logic;
signal ke_done      : std_logic;

-- ------------------------------------------------------------------
--  aes_round <-> controller signals
-- ------------------------------------------------------------------
signal ar_ena        : std_logic;
signal ar_round_cnt  : unsigned(3 downto 0);
signal ar_round_done : std_logic;

-- Mux for aes_round input data
signal round_input_data : std_logic_vector(127 downto 0);
```

```
begin
    -- ------------------------------------------------------------------
    --  Reset for aes_round (active low)
    -- ------------------------------------------------------------------
    rst_n_sig <= not reset;
    -- ------------------------------------------------------------------
    --  KeyExpansion instance
    -- ------------------------------------------------------------------
    zakaria-mghili, 6 days ago | 1 author (zakaria-mghili)
    u_keyexp : KeyExpansion
        port map (
            clk       => clk,
            reset     => reset,
            start     => ke_start,
            key_in    => key_reg,
            round_key => ke_round_key,
            round     => ke_round_int,
            valid     => ke_valid,
            done      => ke_done
        );

    -- ------------------------------------------------------------------
    --  Mux for input_data of aes_round:
    --    round 0  -> use plaintext (data_reg)
    --    rounds 1..10 -> use previous state (state_data)
    -- ------------------------------------------------------------------
    round_input_data <= data_reg  when ke_round_int = 0 else
                        state_data;

    -- ------------------------------------------------------------------
    --  Integer (0..10) -> UNSIGNED(3 downto 0) for aes_round.round_count
    -- ------------------------------------------------------------------
    ar_round_cnt <= to_unsigned(ke_round_int, 4);

    -- ------------------------------------------------------------------
    --  aes_round instance
    -- ------------------------------------------------------------------
    zakaria-mghili, 6 days ago | 1 author (zakaria-mghili)
    u_round : aes_round
        port map (
            clk         => clk,
            rst_n       => rst_n_sig,
            ena         => ar_ena,
            round_count => ar_round_cnt,
            input_data  => round_input_data,
            round_key   => ke_round_key,
            output_data => state_data,      -- feedback state
            round_done  => ar_round_done    -- not used by controller
        );

    -- ------------------------------------------------------------------
    --  Ciphertext output = current AES state
    -- ------------------------------------------------------------------
    data_out <= state_data;
```

```
-- ------------------------------------------------------------------
ar_ena <= '1' when (state = RUN and ke_valid = '1') else '0';      zakaria-mghili, 6 days ago • last version
-- ------------------------------------------------------------------
--  Main controller FSM
```

```vhdl
process(clk)
begin
    zakaria-mghili, 6 days ago | 1 author (zakaria-mghili)
    if rising_edge(clk) then
        zakaria-mghili, 6 days ago | 1 author (zakaria-mghili)
        if reset = '1' then
            state    <= IDLE;
            key_reg  <= (others => '0');
            data_reg <= (others => '0');
            ke_start <= '0';
            busy     <= '0';
            done     <= '0';

        else
            -- defaults every clock
            ke_start <= '0';
            done     <= '0';

            zakaria-mghili, 6 days ago | 1 author (zakaria-mghili)
            case state is

                ------------------------------------------------
                when IDLE =>
                    busy <= '0';

                    zakaria-mghili, 6 days ago | 1 author (zakaria-mghili)
                    if start = '1' then
                        -- Latch inputs from global FSM
                        key_reg  <= key_in;
                        data_reg <= data_in;

                        -- Start key expansion
                        ke_start <= '1';

                        state <= RUN;
                    end if;

                ------------------------------------------------
                when RUN =>
                    busy <= '1';

                    -- KeyExpansion + aes_round are running.
                    -- ke_done becomes '1' when round 10 key is issued.
                    zakaria-mghili, 6 days ago | 1 author (zakaria-mghili)
                    if ke_done = '1' then
                        state <= DONE_STATE;
                    end if;

                when DONE_STATE =>
                    busy <= '0';
                    done <= '1';  -- one-cycle pulse to global FSM

                    -- Wait for global 'start' to go low before next job
                    zakaria-mghili, 6 days ago | 1 author (zakaria-mghili)
                    if start = '0' then
                        state <= IDLE;
                    end if;

            end case;
        end if;
    end if;
end process;

end architecture rtl;
```


/aes_core/clk
/aes_core/reset
/aes_core/start
/aes_core/key_in
/aes_core/data_in
/aes_core/data_out
/aes_core/busy
/aes_core/done
/aes_core/state
/aes_core/key_reg
/aes_core/data_reg
/aes_core/state_data
/aes_core/rst_n_sig
/aes_core/ke_start
/aes_core/ke_roun...
/aes_core/ke_roun...
/aes_core/ke_valid
/aes_core/ke_done
/aes_core/ar_ena
/aes_core/ar_round...
/aes_core/ar_round...
/aes_core/round_in...
1
0
1
E8E9EAEBEDEEE...
000001010303307...
B211E0B6DF083...
0
1
DONE_STATE
E8E9EAEBEDEEE...
000001010303307...
B211E0B6DF083...
1
0
2E0A8A98579CB...
A
0
0
0
A
0
B211E0B6DF083...
E8E9EAEBEDEEEFF0F2F3F4F5F7F8F9FA
000001010303070F0F1F1F3F3F7F7F
XXXX...
B211E0B6DF083B6454C753DE186737E8
RUN
DONE STATE
E8E9EAEBEDEEEFF0F2F3F4F5F7F8F9FA
000001010303070F0F1F1F3F3F7F7F
XXXX...
B211E0B6DF083B6454C753DE186737E8
X...
2E0A8A98579CB2F4E9BE5C8F55F94DE1
0 1 2 3 4 5 6 7 8 9 A
0 1 2 3 4 5 6 7 8 9 A
00...
B211E0B6DF083B6454C753DE186737E8

L'Unité de Contrôle Centrale : La FSM globale

Garantir le Chiffrement des Données du Drone en Temps Réel

La FSM : Le Chef d'Orchestre

Rôle de la FSM :

La FSM est vitale pour garantir le bon déroulement du processus :

1. Respect de l'Ordre : Attendre Donnée → Clé → Chiffrement → Envoi.
2. Gestion de l'Attente : Attendre les signaux done avant de passer aux étapes suivantes.
3. Vérification des Conditions : Vérifier les signaux ready ou valid des autres modules.

Les Modules :

* CSPRNG : génération des clés.
* AES : Le chiffrement des données.
* FIFOs : Les buffers d'entrée qui gèrent le flux de données.

Le Dialogue des Signaux : Comment la FSM Communique

ENTRÉES STANDARD

```
CLK                 : in  std_logic;
RESET               : in  std_logic;
```

- Horloge (synchronisation).
- Réinitialisation asynchrone de la FSM.

ENTRÉES (Statuts)

```
System_START        : in  std_logic;
Key_Refresh_Request : in std_logic;

data_ready          : in  std_logic;
key_valid           : in  std_logic;
ciphertext_ready    : in  std_logic;
output_fifo_not_full : in std_logic;
```

- Déclenchement initial du processus.
- Demande de priorité pour renouveler la clé (sécurité).
- Indique qu'un bloc de données est prêt à être traité.
- Confirme que la nouvelle clé est générée et valide.
- Confirme que le chiffrement du bloc est terminé.
- Indique qu'il y a de la place pour écrire le résultat chiffré.

Le Dialogue des Signaux : Comment la FSM Communique

SORTIES
(Commandes)

```
fsm_keygen_start_o      : out std_logic;
fsm_start_enc_o         : out std_logic;
fsm_read_enable_o       : out std_logic;
fsm_write_enable_o      : out std_logic;
reset_counter           : out std_logic;

key_valid_out           : out std_logic
```

- Pulse de commande pour démarrer la génération de clé.
- Pulse de commande pour lancer le cycle de chiffrement AES.
- Autorisation de consommer le bloc de données d'entrée.
- Autorisation d'écrire le texte chiffré dans le buffer de sortie.
- Pulse de réinitialisation du compteur du mode d'opération, obligatoire lors d'un changement de clé.

- État reflétant si une clé est actuellement chargée et active.

Les 4 États Fondamentaux et Leurs Logiques

1. État IDLE (Repos)

Rôle : Attente de l'instruction System_START et que data_ready soit actif.
Transition : Vers KEY_GENERATION (si clé invalide) ou ENCRYPT (si clé valide).

2. État KEY_GENERATION

Rôle : Envoie fsm_keygen_start_o pour générer la clé et active reset_counter.
Transition : Vers ENCRYPT dès réception de key_valid = 1.

3. État ENCRYPT (Chiffrement)

Rôle : Envoie fsm_start_enc_o et fsm_read_enable_o pour traiter le bloc.
Transition : Vers OUTPUT_READY dès réception de ciphertext_ready = 1.

4. État OUTPUT_READY (Transmission)

Rôle : Envoie fsm_write_enable_o pour stocker le résultat chiffré dans le FIFO de sortie.
Décision : Retourne vers ENCRYPT (bloc suivant), KEY_GENERATION (renouvellement de clé), ou IDLE (fin de flux).

Implémentation : La Logique VHDL du Contrôleur FSM

```
architecture Behavioral of CryptoCore_FSM is


    type FSM_STATES is (S_IDLE, S_KEY_GENERATION, S_ENCRYPT, S_OUTPUT_READY);
    signal current_state, next_state : FSM_STATES;

    signal internal_key_valid : std_logic := '0';

begin
```

Définition des états de la FSM et des signaux de contrôle.
La FSM pilote les phases : repos, génération de clé, chiffrement AES et transmission. Le signal internal_key_valid indique si une clé de chiffrement valide est déjà disponible.

Implémentation : La Logique VHDL du Contrôleur FSM

PROCESSUS 1 : Registre d'État
(Synchronisé par l'horloge)

```
STATE_REGISTER : process(CLK, RESET)
begin
    if RESET = '1' then
        current_state <= S_IDLE;
        internal_key_valid <= '0';
    elsif rising_edge(CLK) then
        current_state <= next_state;

        if (current_state = S_KEY_GENERATION) and (key_valid = '1') then
            internal_key_valid <= '1';
        elsif (current_state = S_OUTPUT_READY) and (Key_Refresh_Request = '1') then
            internal_key_valid <= '0';
        end if;

    end if;
end process STATE_REGISTER;
```

Ce processus représente le registre d’état de la FSM.
Lorsqu’un reset est activé, le système revient à l’état IDLE et aucune clé n’est considérée comme valide.
À chaque front montant de l’horloge, la FSM passe à l’état suivant calculé.
Si la génération de clé est terminée avec succès, la clé est marquée comme valide.
En revanche, lorsqu’une demande de renouvellement de clé est reçue après l’envoi des données, la clé est invalidée afin d’en générer une nouvelle.

Implémentation : La Logique VHDL du Contrôleur FSM

PROCESSUS 2 : Logique de Transition (Détermination de next_state)

```
NEXT_STATE_LOGIC : process(current_state, System_START, data_ready, key_valid, ciphertext_ready, Key_Refresh_Request, internal_key_valid)
begin
    next_state <= current_state;

    case current_state is

        when S_IDLE =>
            if (System_START = '1' or data_ready = '1') then
                if (internal_key_valid = '0') then
                    next_state <= S_KEY_GENERATION;
                else
                    next_state <= S_ENCRYPT;
                end if;
            end if;
        when S_KEY_GENERATION =>

            if key_valid = '1' then
                next_state <= S_ENCRYPT;
            end if;
```

```
        when S_ENCRYPT =>
            if ciphertext_ready = '1' then
                next_state <= S_OUTPUT_READY;
            end if;

        when S_OUTPUT_READY =>
            if Key_Refresh_Request = '1' then
                next_state <= S_KEY_GENERATION;
            elsif data_ready = '1' then
                next_state <= S_ENCRYPT;
            else
                next_state <= S_IDLE;
            end if;

    end case;
end process NEXT_STATE_LOGIC;
```

Ce processus définit la logique de transition de la FSM. Il calcule l'état suivant du système à partir de l'état courant et des signaux de contrôle. En mode repos, la FSM vérifie le démarrage, la disponibilité des données et la validité de la clé. Si nécessaire, elle lance la génération de clé avant le chiffrement. Une fois la clé prête, le système passe au chiffrement AES, puis à la transmission des données chiffrées. Enfin, la FSM gère soit le chiffrement des blocs suivants, soit le renouvellement de la clé, soit le retour à l'état IDLE lorsque le flux est terminé.

Implémentation : La Logique VHDL du Contrôleur FSM

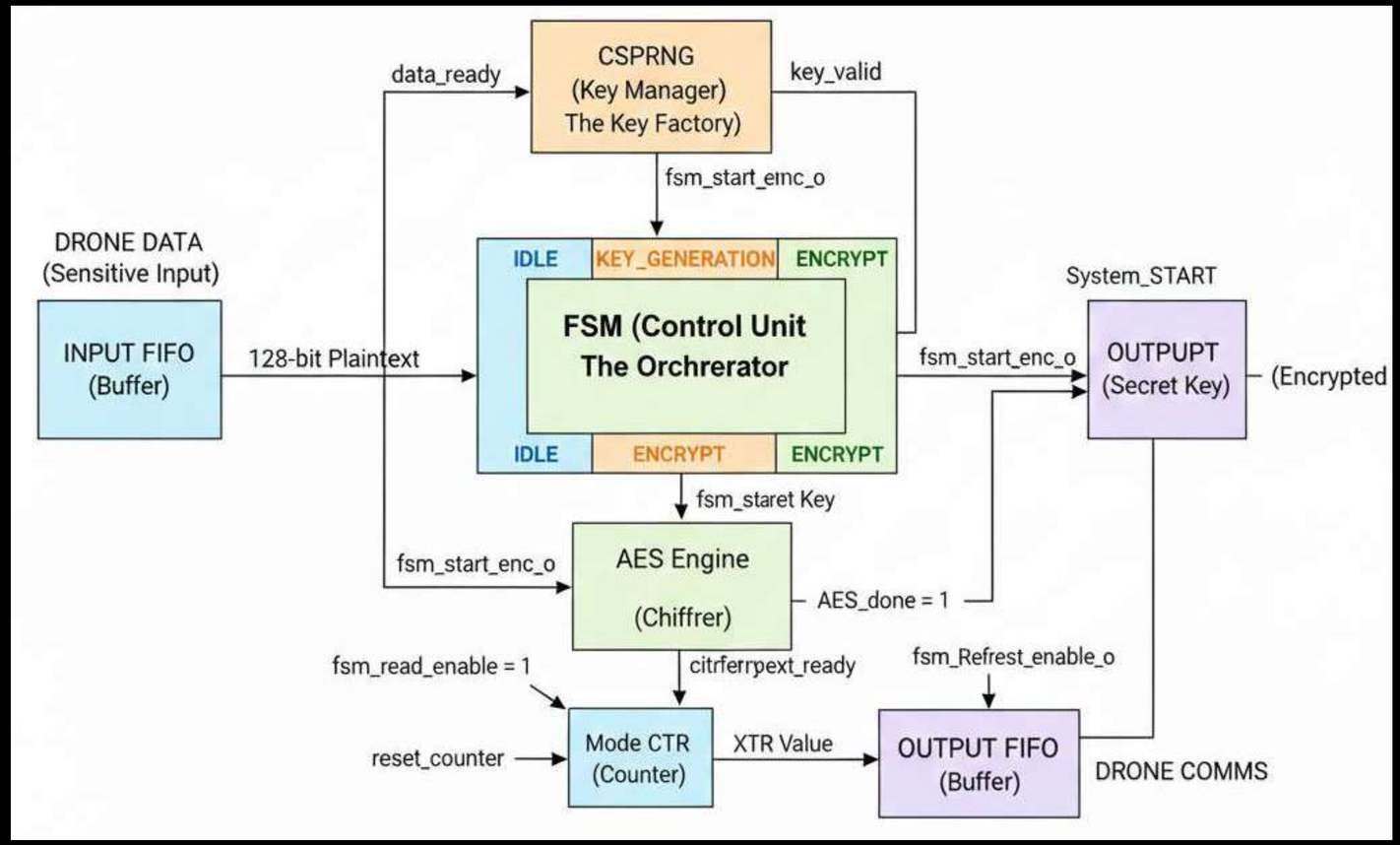
```
    OUTPUT_ACTIONS : process(current_state, ciphertext_ready, output_fifo_not_full)
    begin
        fsm_keygen_start_o      <= '0';
        fsm_start_enc_o         <= '0';
        fsm_read_enable_o       <= '0';
        fsm_write_enable_o      <= '0';
        reset_counter           <= '0';

        case current_state is
            when S_KEY_GENERATION =>
                fsm_keygen_start_o <= '1';
                reset_counter      <= '1';
            when S_ENCRYPT =>
                fsm_start_enc_o <= '1';
                fsm_read_enable_o <= '1';

            when S_OUTPUT_READY =>
                if (ciphertext_ready = '1' and output_fifo_not_full = '1') then
                    fsm_write_enable_o <= '1';
                end if;

            when others =>
                null;
        end case;
    end process OUTPUT_ACTIONS;

end architecture Behavioral;
```

PROCESSUS 3 : Génération des Signaux de Sortie (Actions)

Ce processus génère les signaux de sortie de la FSM en fonction de l'état courant du système.
Par défaut, tous les signaux sont désactivés afin d'éviter toute action non désirée.
En état de génération de clé, la FSM active le signal de démarrage du générateur de clé et réinitialise le compteur.
En état de chiffrement, elle lance le moteur AES et autorise la lecture des données depuis le FIFO d'entrée.
En état de transmission, la FSM permet l'écriture des données chiffrées dans le FIFO de sortie, uniquement si le chiffrement est terminé et que le FIFO n'est pas plein.
Ainsi, ce processus assure que chaque module est activé uniquement au bon moment.

Diagramme d'États Finis (FSM) Globale

Synchronisation du Flux de Données et de la Gestion des Clés



Ce FSM assure une synchronisation complète entre la génération des clés, le chiffrement des données et leur transmission sécurisée.

INTÉGRATION SYSTÈME ET PERFORMANCE

L'Unité de Contrôle : Clé de la Fiabilité et du Haut Débit du CryptoCore

Objectifs Validés :

- Garantir l'Intégrité du flux de données.
- Optimiser la Vitesse (Atteindre le Fmax).
- Valider l'Architecture du système Top-Level.

Ces objectifs sont validés. Je vais maintenant vous présenter les preuves concrètes de cette validation, en commençant par la sécurité du flux.

SÉCURITÉ DU FLUX (CONTRÔLE DES FIFOS)

SÉCURISATION DU FLUX (Contrôle des Buffers)

Rôle : Éviter les erreurs et les pertes de données critiques du drone.

A. CONTRÔLE EN LECTURE (Input FIFO)

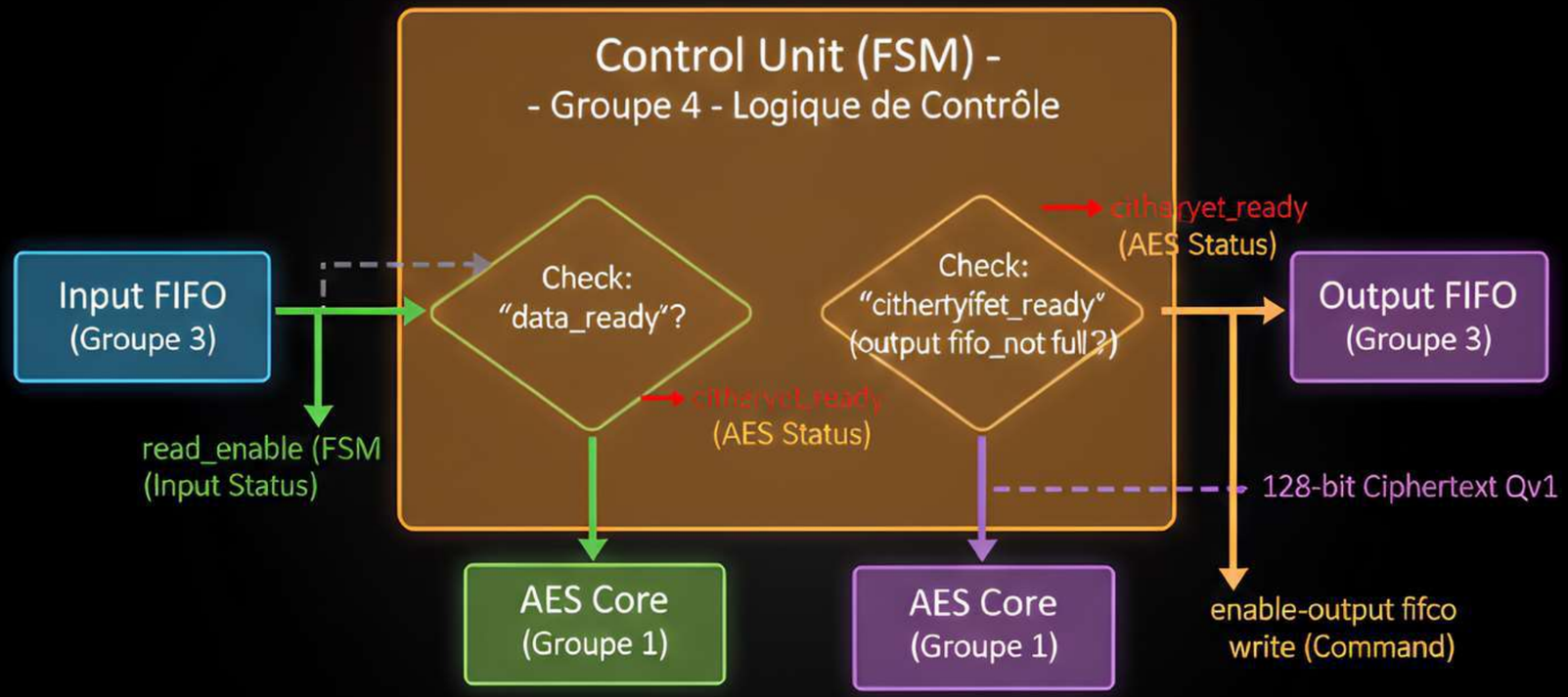
- Vérification : Le signal data_ready est actif (FIFO non vide).
- Action : La FSM active read_enable.
- But : Prévenir une lecture à vide de l'AES (Erreur système).

B. CONTRÔLE EN ÉCRITURE (Output FIFO)

- Vérification : Le statut output_fifo_not_full est actif.
- Action : La FSM envoie enable_output_fifo_write.
- But : Éviter l'Overflow (Perte de données chiffrées) si le module de communication est saturé.

SÉCURITÉ DU FLUX (CONTRÔLE DES FIFOS)

FSM CryptoCore: Contrôl des FIFOS (Sécurité du Flux)



OPTIMISATION DU DÉBIT (PIPELINING)

HAUT DÉBIT GARANTI : LE MODE STREAMING

Principe : Exploiter l'architecture pipelinée de l'AES pour le temps réel.

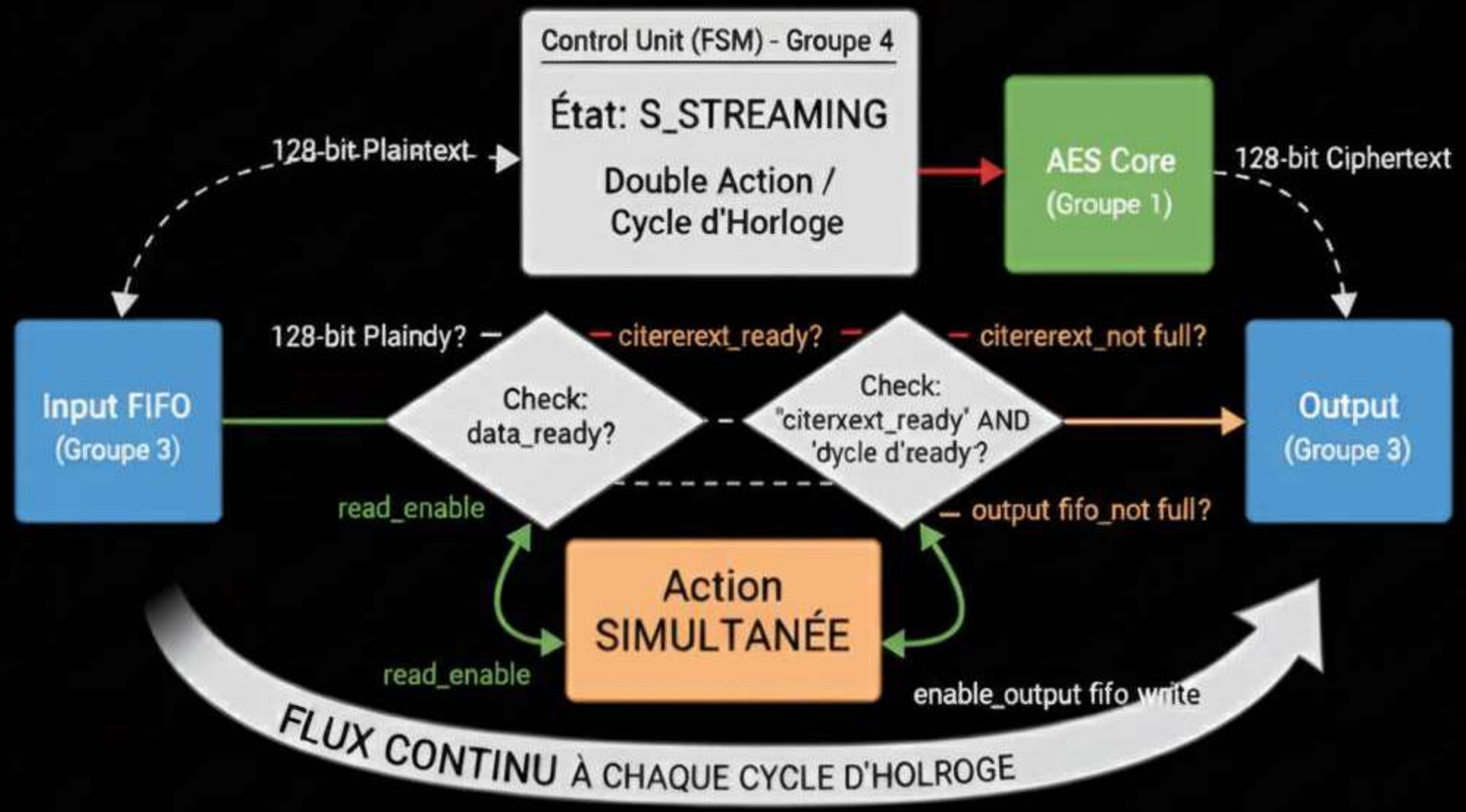
A. Transition Stratégique

- La FSM remplace l'état d'attente par l'état S_STREAMING.
- Ceci permet de maintenir l'AES actif en permanence.

B. Logique de l'Action Simultanée

- La FSM exécute DEUX commandes PAR CYCLE D'HORLOGE :
  a. Injection : Lecture d'un nouveau bloc de texte clair (read_enable).
  b. Récupération : Écriture du chiffré précédent (enable_output_fifo_write).
- Résultat : Le CryptoCore maintient un flux continu et atteint le Débit Maximal (Fmax).

OPTIMISATION DU DÉBIT (PIPELINING)



RÉSUMÉ: Gestion en Parallele & Continu ⇒ Débit Maximal (Fmax)

VALIDATION DU SYSTÈME (LIVRABLES)

PREUVES DE VALIDATION ET INTÉGRATION

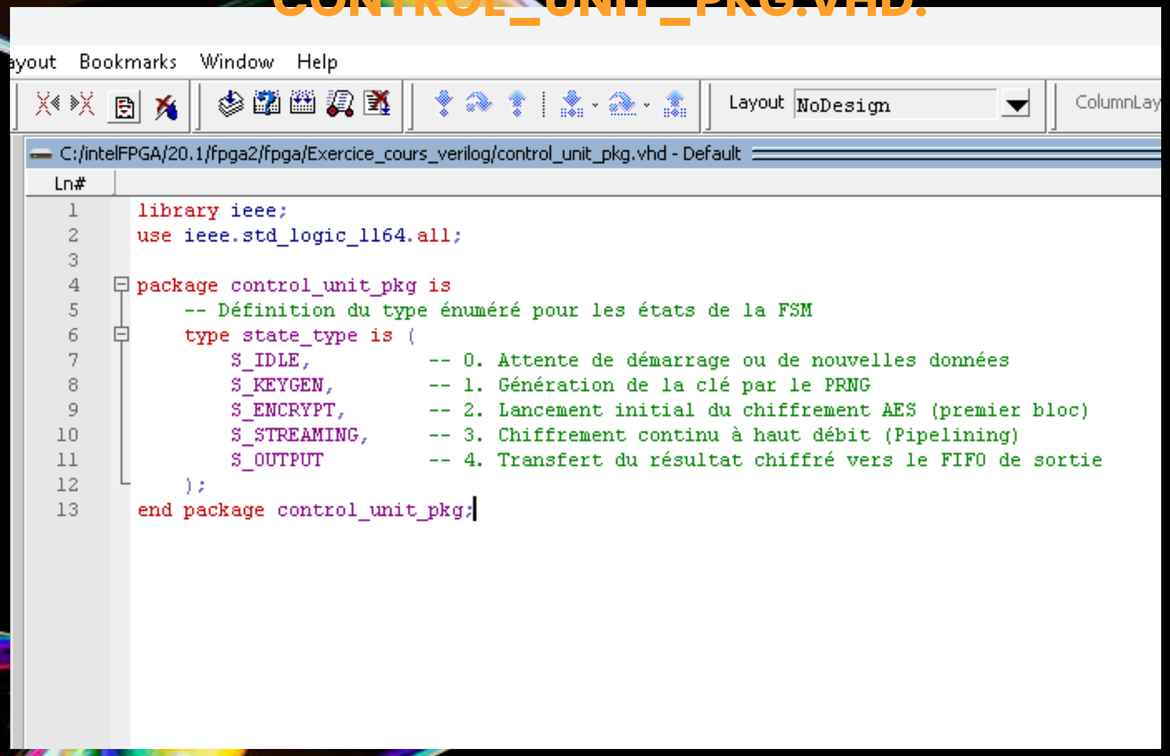
Notre Rôle : Valider la conception logique et la synchronisation.

- **Design Logique Complet : Nous présentons l'analyse du Diagramme FSM pour la séquence sécurisée, et le Code VHDL en entier pour la preuve de l'implémentation synthétisable.**
- **Intégration Top-Level : Le schéma montre notre Control Unit au centre, ce qui valide notre rôle de pilote central et d'Architecte du Contrôle du système.**
- **Simulation Chronométrée : Cette preuve formelle confirme le respect de la séquence critique (ex : key_done autorise start_enc). La validation de la synchronisation est prouvée à un cycle d'horloge près.**

VALIDATION DU SYSTÈME (LIVRABLES)

POUR RENDRE NOTRE CODE LISIBLE ET FACILEMENT MAINTENABLE, IL EST CRUCIAL DE DÉFINIR LE TYPE DE VOS ÉTATS DANS UN PACKAGE SÉPARÉ.

LE CODE : FICHIER 1 : CONTROL_UNIT_PKG.VHD.



"LE PREMIER LIVRABLE, ET LA BASE DE NOTRE SOLUTION, EST LE CODE VHDL. NOUS N'AVONS PAS SEULEMENT ÉCRIT LE CODE, NOUS L'AVONS STRUCTURÉ POUR GARANTIR LA MAINTENABILITÉ ET LA SYNTHÉTISABILITÉ."

1. ARCHITECTURE MODULAIRE ET CLARTÉ (LE PACKAGE)

"POUR UN PROJET PROFESSIONNEL, NOUS AVONS D'ABORD SÉPARÉ NOS DÉFINITIONS. NOUS AVONS CRÉÉ UN PACKAGE SÉPARÉ POUR DÉFINIR LE TYPE DE NOS ÉTATS : S_IDLE, S_KEYGEN, S_STREAMING, ETC."

"CECI REND NOTRE CODE PRINCIPAL BEAUCOUP PLUS CLAIR ET NOUS PERMET DE CENTRALISER LA LISTE DE NOS ÉTATS POUR FACILITER LE DÉBOGAGE ET LA MISE À JOUR."

VALIDATION DU SYSTÈME (LIVRABLES)

C'EST LA PARTIE LA PLUS IMPORTANTE POUR L'INTÉGRATION TOP-LEVEL, CAR ELLE DÉFINIT L'INTERFACE DE NOTRE FSM AVEC LES AUTRES GROUPES.

FICHIER 2 : CONTROL_UNIT.VHD

```
library ieee;
use ieee.std_logic_1164.all;
use work.control_unit_pkg.all; -- Utilisation du package pour les états

entity Control_Unit is
    port (
        -- Signaux de Base
        clk                     : in  std_logic;
        reset                   : in  std_logic;
        System_START            : in  std_logic; -- Signal de démarrage du système

        -- Signaux d'Entrée (Statuts des autres groupes)
        data_ready              : in  std_logic; -- (Groupe 3) Le FIFO d'entrée a un bloc prêt
        key_done                : in  std_logic; -- (Groupe 2) La clé est générée et étendue
        ciphertext_ready        : in  std_logic; -- (Groupe 1) L'AES a un bloc chiffré prêt
        output_fifo_not_full    : in  std_logic; -- (Groupe 3) Le FIFO de sortie n'est pas plein
        Key_Refresh_Request     : in  std_logic; -- Demande de renouvellement de clé

        -- Signaux de Sortie (Commandes vers les autres groupes)
        start_keygen            : out std_logic; -- (Groupe 2) Commande de génération de clé
        start_enc               : out std_logic; -- (Groupe 1) Commande de lancement du chiffrement
        read_enable             : out std_logic; -- (Groupe 3) Commande de lecture du FIFO d'entrée
        enable_output_fifo_write: out std_logic; -- (Groupe 3) Commande d'écriture dans le FIFO de sortie

        -- Signal de sortie pour la surveillance/débogage
        FSM_State_Out           : out state_type -- État actuel (pour le testbench)
    );
end entity Control_Unit;
```

1. L'INTERFACE (LES PORTS D'ENTRÉE/SORTIE)

"D'ABORD, REGARDONS L'INTERFACE (LES PORTS). CETTE PARTIE EST ESSENTIELLE CAR ELLE PROUVE NOTRE RÔLE DE PILOTE CENTRAL."

"LES ENTRÉES SONT LES STATUTS QUE NOUS LISONS DES AUTRES GROUPES : DATA_READY (FIFO), KEY_DONE (PRNG), CIPHERTEXT_READY (AES). LES SORTIES SONT NOS COMMANDES : START_ENC, READ_ENABLE, ET ENABLE_OUTPUT_FIFO_WRITE.

- "CETTE INTERFACE FAIT DE NOTRE FSM LE SEUL POINT DE CONNEXION ENTRE TOUS LES BLOCS."

VALIDATION DU SYSTÈME (LIVRABLES)

C'EST LA PARTIE LA PLUS IMPORTANTE POUR L'INTÉGRATION TOP-LEVEL, CAR ELLE DÉFINIT L'INTERFACE DE NOTRE FSM AVEC LES AUTRES GROUPES.

2. L'ARCHITECTURE (LOGIQUE DES TRANSITIONS ET DES COMMANDES)

"À L'INTÉRIEUR DE L'ARCHITECTURE, NOUS AVONS IMPLÉMENTÉ LA LOGIQUE CRITIQUE QUE NOUS AVONS PRÉSENTÉE :"

* IMPLÉMENTATION DE LA SÉCURITÉ DES FLUX : "NOUS UTILISONS DES CONDITIONS MULTIPLES DANS LA LOGIQUE DES ÉTATS. PAR EXEMPLE, L'ACTIVATION DE READ_ENABLE EST TOUJOURS CONDITIONNÉE PAR DATA_READY, ET L'ÉCRITURE EST CONDITIONNÉE PAR OUTPUT_FIFO_NOT_FULL. CECI EST L'IMPLÉMENTATION DIRECTE DU CONTRÔLE DES FIFOS."

* IMPLÉMENTATION DU MODE STREAMING : "L'ÉTAT S_STREAMING EST LE CŒUR DE LA PERFORMANCE. C'EST DANS CET ÉTAT QUE NOUS IMPLÉMENTONS L'ACTION SIMULTANÉE : SI LES CONDITIONS DE SÉCURITÉ SONT REMPLIES, NOUS ACTIVONS READ_ENABLE ET ENABLE_OUTPUT_FIFO_WRITE AU MÊME CYCLE D'HORLOGE.

* RÉSULTAT : "LE CODE VALIDE QUE NOTRE FSM PEUT LANCER LE CHIFFREMENT, GÉRER LES DONNÉES D'ENTRÉE/SORTIE, ET GARANTIR LA SÉCURITÉ CONTRE L'OVERFLOW TOUT EN MAINTENANT UN DÉBIT CONTINU."

VALIDATION DU SYSTÈME (LIVRABLES)

FICHIER 2 : CONTROL_UNIT.VHD

C'EST LA PARTIE LA PLUS IMPORTANTE POUR L'INTÉGRATION TOP-LEVEL, CAR ELLE DÉFINIT L'INTERFACE DE NOTRE FSM AVEC LES AUTRES GROUPES.

C:/intelFPGA/20.1/fpga2/fpga/Exercice_cours_verilog/Control_Unit.vhd - Default *

```
Ln#
45  architecture Behavioral of Control_Unit is
46
47      -- Signaux internes de la FSM
48      signal current_state : state_type := S_IDLE;
49      signal next_state    : state_type;
50
51      -- Signal pour indiquer si une clé valide est déjà chargée
52      signal key_valid     : std_logic := '0';
53
54  begin
55
56      -- Mappage de l'état de sortie pour le débogage
57      FSM_State_Out <= current_state;
58
59      -----------------------------------------------------------
60      -- PROCESSUS 1: Registre d'État (SÉQUENTIEL)
61      -- Gère la transition d'état à chaque front d'horloge.
62      -----------------------------------------------------------
63      State_Register: process(clk, reset)
64      begin
65          if reset = '1' then
66              current_state <= S_IDLE; -- Reset place l'état à IDLE
67              key_valid     <= '0';    -- Clé invalidée
68          elsif rising_edge(clk) then
69              current_state <= next_state;
70              -- Mise à jour de la validité de la clé
71              if current_state = S_KEYGEN and key_done = '1' then
72                  key_valid <= '1';
73              end if;
74          end if;
75      end process State_Register;
76
77      -----------------------------------------------------------
78      -- PROCESSUS 2: Logique d'État Suivant et Sorties (COMBINATOIRE)
79      -- Détermine l'état suivant et les signaux de commande.
80      -----------------------------------------------------------
81      Next_State_Logic_and_Outputs: process(current_state, System_START, data_ready,
82      begin
83          -- Valeurs par défaut des sorties (Inactifs par défaut)
84          next_state             <= current_state;
85          start_keygen           <= '0';
86          start_enc              <= '0';
87          read_enable            <= '0';
88          enable_output_fifo_write <= '0';
89
90          -- La logique de l'état (Définition de l'état suivant)
91          case current_state is
92
93              -- ÉTAT S_IDLE : Attente du lancement et/ou de données
94              when S_IDLE =>
95                  -- Condition de démarrage du système et données prêtes
96                  if System_START = '1' and data_ready = '1' then
97                      if key_valid = '0' or Key_Refresh_Request = '1' then
98                          next_state <= S_KEYGEN; -- Besoin de générer la clé
99                      else
100                         next_state <= S_ENCRYPT; -- Clé déjà prête, on peut chi
101                     end if;
102                 end if;
103
104             -- ÉTAT S_KEYGEN : Génération de Clé
105             when S_KEYGEN =>
106                 start_keygen <= '1'; -- COMMANDE : Active la génération de clé
107                 if key_done = '1' then
108                     next_state <= S_ENCRYPT; -- ÉVÉNEMENT : Clé prête, passer a
109                 end if;
110
```

Partie sécurité des flux (CTRL FIFO)

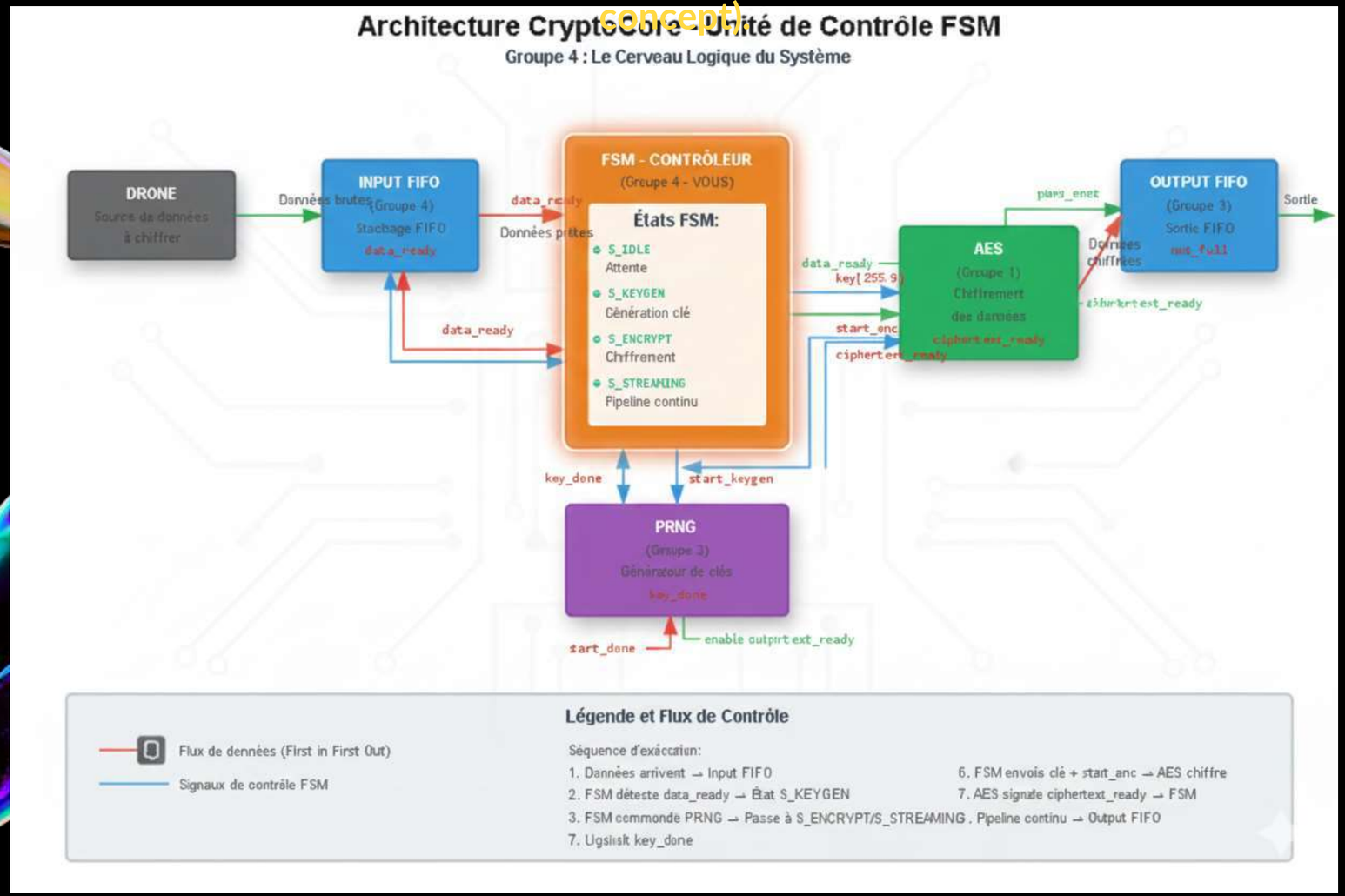
```
111             -- ////////////////////////////////////////////////////////////////     ÉTAT S_ENCRYPT : Lancement du pre
112             when S_ENCRYPT =>
113                 start_enc   <= '1'; -- COMMANDE : Lance l'opération AES
114                 read_enable <= '1'; -- COMMANDE : Injection du premier bloc de données (lecture du FIFO)
115
116                 if ciphertext_ready = '1' then
117                     -- Dès que le premier chiffré est prêt, on entre en mode streaming
118                     next_state <= S_STREAMING;
119                 end if;
120
121             -- ÉTAT S_STREAMING : Chiffrement continu à haut débit (PIPELINING)
122             when S_STREAMING =>
123                 -- Le moteur AES est actif en permanence dans cet état
124                 start_enc <= '1';
125
126                 -- LOGIQUE D'ACTION SIMULTANÉE (Gestion des FIFOs)
127                 -- Le cœur AES termine un bloc et est prêt à en commencer un autre à chaque cycle.
128                 if data_ready = '1' and output_fifo_not_full = '1' then
129                     -- ACTION SIMULTANÉE: LECTURE (injection) ET ÉCRITURE (récupération)
130                     read_enable              <= '1';
131                     enable_output_fifo_write <= '1';
132                     next_state               <= S_STREAMING; -- Reste en mode continu
133                 elsif data_ready = '0' then
134                     -- Fin des données d'entrée
135                     next_state <= S_OUTPUT;
136                 elsif output_fifo_not_full = '0' then
137                     -- FIFO de sortie est plein, arrêt temporaire (retour au transfert)
138                     next_state <= S_OUTPUT;
139                 end if;
140
141                 -- Vérification de rafraîchissement (laisse le bloc en cours se terminer)
142                 if Key_Refresh_Request = '1' and ciphertext_ready = '1' then
143                     next_state <= S_KEYGEN;
144                 end if;
145
146             -- ÉTAT S_OUTPUT : Transfert de la donnée chiffrée
147             when S_OUTPUT =>
148                 enable_output_fifo_write <= '1'; -- COMMANDE : Écrit le bloc chiffré (peut prendre un cycle)
149
150                 -- Décision de l'état suivant
151                 -- Après l'écriture (assumé 1 cycle pour une FSM simple)
152                 if data_ready = '1' and output_fifo_not_full = '1' then
153                     -- Données prêtes et place disponible : Retour au streaming
154                     next_state <= S_STREAMING;
155                 elsif Key_Refresh_Request = '1' then
156                     -- Demande de clé : Retour à la génération
157                     next_state <= S_KEYGEN;
158                 else
159                     -- Pas de données/pas de place : Retour à l'attente
160                     next_state <= S_IDLE;
161                 end if;
162
163             -- État par défaut (sécurité)
164             when others =>
165                 next_state <= S_IDLE;
166
167         end case;
168     end process Next_State_Logic_and_Outputs;
169
170 end architecture Behavioral;
```

Control_Unit.vhd * | control_unit_pkg.vhd

Partie optimisation du débit (pipeline)

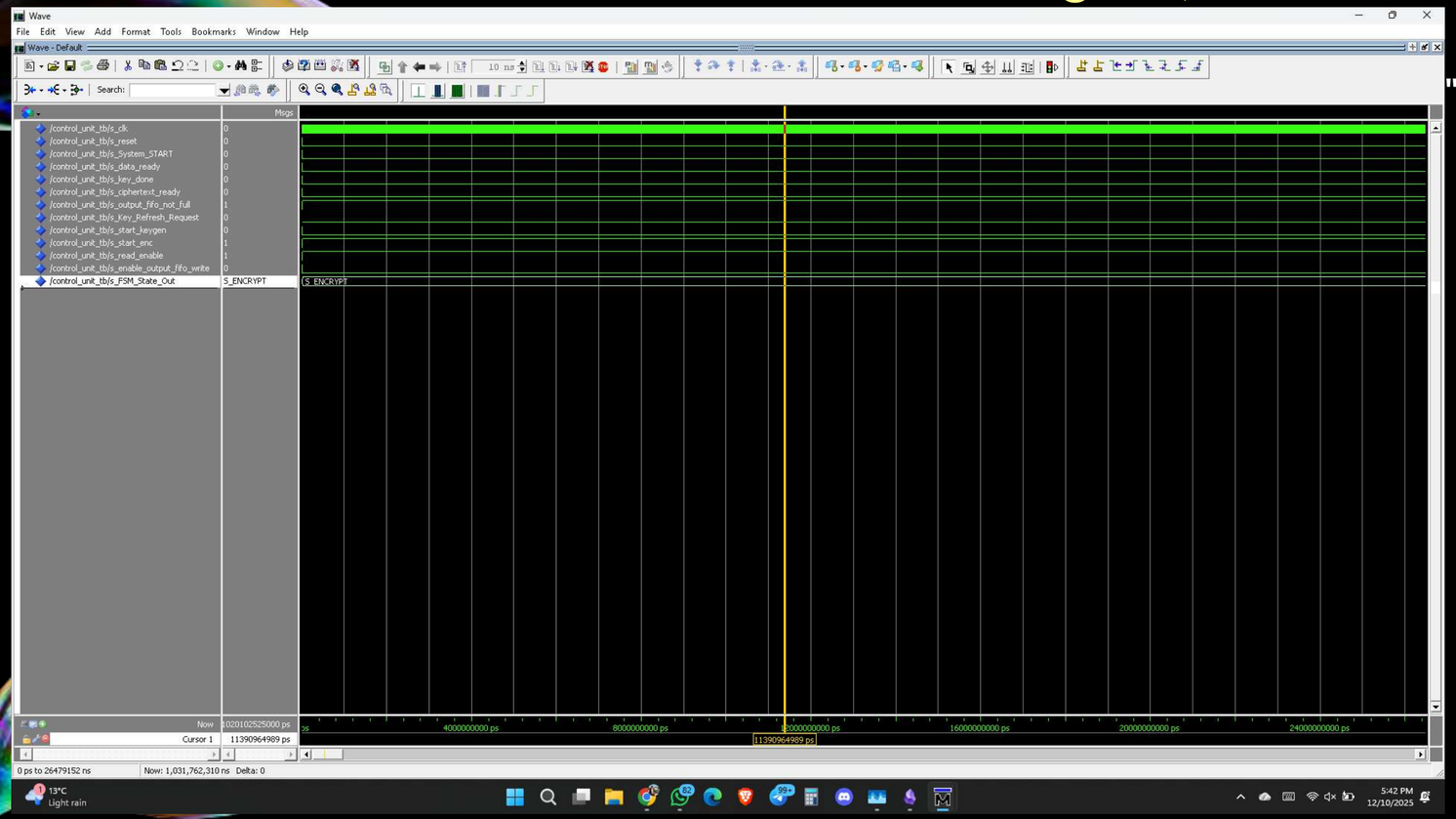
VALIDATION DU SYSTÈME (LIVRABLES)

La Logique : Diagramme FSM (Le concept)



VALIDATION DU SYSTÈME (LIVRABLES)

La Preuve : Simulation Chronométrée (Timing Diagram).



La Preuve de Synchronisation (Le Cycle Près)

"Le point d'honneur est la validation temporelle :

- **"Nous avons simulé des transitions clés, comme le passage de key_done à start_enc, ou le démarrage du S_STREAMING."**
- **"Ici, lorsque les conditions de démarrage sont réunies (ex : System_START et data_ready passent à '1'), vous pouvez observer que la FSM quitte l'état S_IDLE et passe à l'état S_KEYGEN sur le prochain front montant de l'horloge."**
- **"Ceci est la preuve définitive que la synchronisation est garantie à un cycle d'horloge près. Il n'y a pas de latence inutile, ce qui est crucial pour le temps réel du drone."**

CONCLUSION : ARCHITECTE DU CONTRÔLE

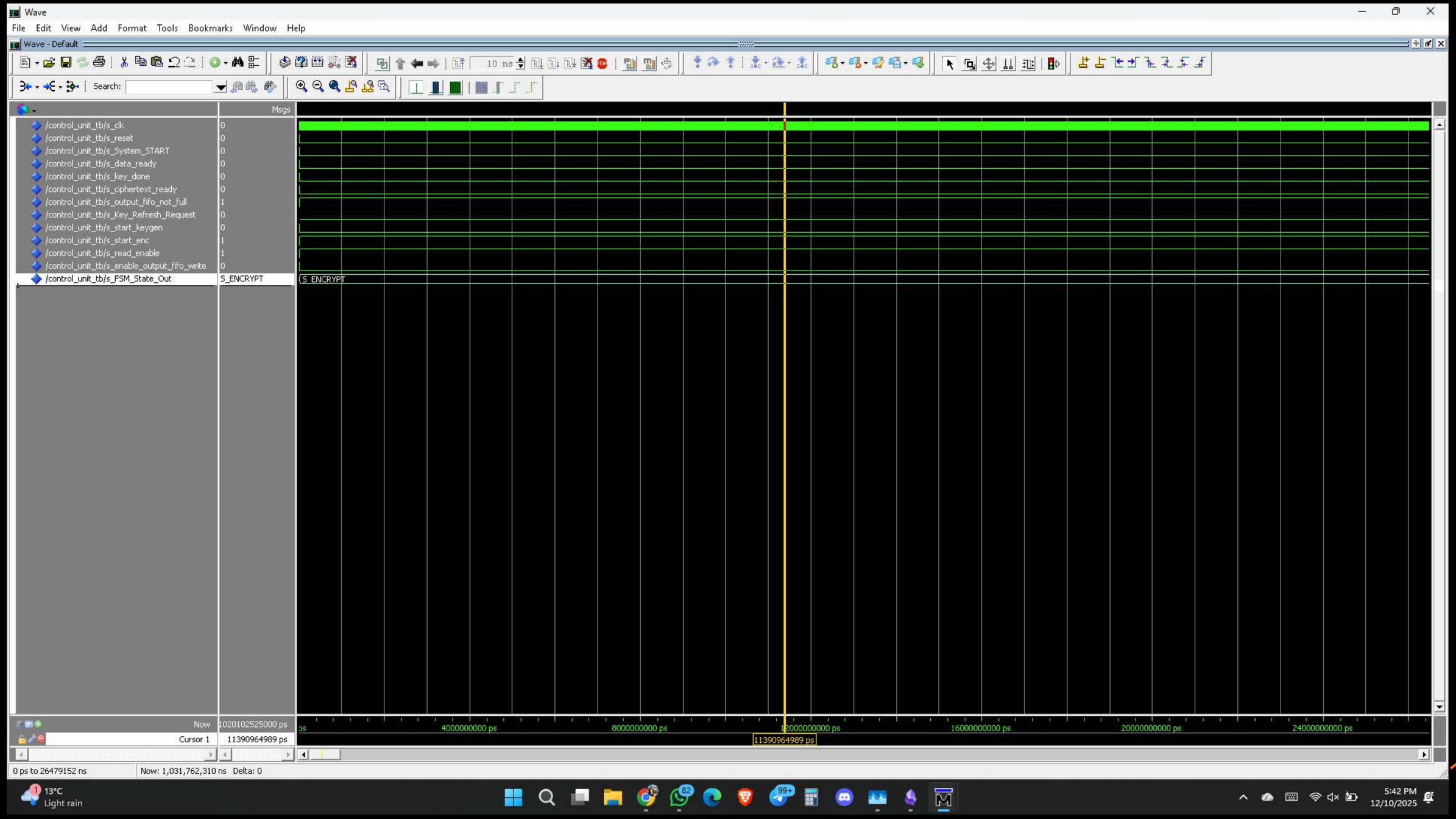
"En résumé, notre Unité de Contrôle FSM a validé les trois objectifs critiques du projet :

- Sécurité et Fiabilité : Nous avons garanti l'intégrité du flux en gérant les conditions de sécurité critiques (prévention de l'Overflow).
- Performance et Optimisation : La séquence de S_STREAMING est prouvée par le chronogramme. L'action simultanée de lecture et d'écriture assure l'atteinte du Débit Maximal (Fmax).
- Validation et Intégration : Le Code VHDL et le Schéma Top-Level confirment notre rôle de pilote central et notre capacité à synchroniser l'ensemble du CryptoCore à un cycle d'horloge près."

"Notre module est le cerveau fiable et performant, prêt pour l'implémentation FPGA."

CONCLUSION : ARCHITECTE DU CONTRÔLE

"Je vous remercie de votre attention. Je suis à votre disposition pour vos questions."


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
/control_unit_tb/s_clk 0
/control_unit_tb/s_reset 0
/control_unit_tb/s_System_START 0
/control_unit_tb/s_data_ready 0
/control_unit_tb/s_key_done 0
/control_unit_tb/s_ciphertext_ready 0
/control_unit_tb/s_output_fifo_not_full 1
/control_unit_tb/s_Key_Refresh_Request 0
/control_unit_tb/s_start_keygen 0
/control_unit_tb/s_start_enc 1
/control_unit_tb/s_read_enable 1
/control_unit_tb/s_enable_output_fifo_write 0
/control_unit_tb/s_FSM_State_Out S_ENCRYPT
S_ENCRYPT
Now 1020102525000 ps
Cursor 1 11390964989 ps
4000000000 ps
8000000000 ps
12000000000 ps
16000000000 ps
20000000000 ps
24000000000 ps
11390964989 ps
0 ps to 26479152 ns
Now: 1,031,762,310 ns Delta: 0
13°C
Light rain
5:42 PM
12/10/2025