Adapted Modular Number System (AMNS)

La représentation AMNS est une représentation originale proposée par Plantard en 2005 [Pla05]. Elle permettrai d'améliorer la parallélisation des calculs.

Définition 1. *(Base AMNS)*
On appelle base AMNS une base composée de l'ensemble $\mathcal{B} = (p, n, \gamma, \rho, \lambda)_E$ *tel que :*

- p *est premier*
- $E(X) = X^n - \lambda$ *est le polynôme de réduction modulaire*
- $\gamma^n = \lambda$ *mod (p)*
- λ *est "petit".*
- $\forall a \in \{0, 1, \ldots, p-1\}, \exists a(\delta) = \sum_{a_i}^{X^i}$ *avec :*
  $a(\gamma) = a$ *mod(p),*
  $||a||_{\inf} = max_{i=1\ldots l-1}|a_i| < \rho$

La multiplication dans cette représentation est inspirée de la multiplication de Montgomery comme le montre l'algorithme 1

Algorithme 1. *Multiplication AMNS*

Data: $\mathcal{B} = (p, n, \gamma, \rho, \lambda)_E$, Une base AMNS.
m such as $m(\gamma) = 0$ mod (p)
an integer $\phi \geq 2n\gamma\lambda\rho$ and $m' = -m^{m-1}$ mod(p)
Input: $a, b \in \mathcal{B}$
Result: z such as $z(\gamma) = a(\gamma)b(\gamma)\phi^{-1}$ mod(p)
begin
  $c \leftarrow a \times b$ mod(E);
  $q \leftarrow c \times m'$ $mod(P_{\mathcal{B}}, \phi)$;
  $z =\leftarrow c + (q \times m)$ $mod(P_{\mathcal{B}})/\phi$;
  Return z;
end

Première étape : multiplication polynômial

La base qui servira d'exemple sera $\mathcal{B} = (p, n, \gamma, \rho, \lambda)_E$, avec :

— p=19001325514169087268406693340194127777457227427109835115440
089655205287947805423172915761409766796895644242980498091
— n=7
— γ=819565097393106699276394531328135313924035846311612770313252246264741388124798942066576667993769686760392900372460 0
— $\rho = 2^{59}$
— $\lambda = 2$
— $E = X^n - \lambda$
— $\phi = 2^{64}$

Un script python détaillant les différentes opérations (exemple.py) vous a été fournis. vous pouvez vous référer à ce script ainsi qu'à l'article [DDV20] afin de comprendre comment réaliser la multiplication polynomial.
l'accélérateur sera composé de plusieurs unités de calcul nommées PE (pour Processing élément).
Lorsque l'on multipliera deux polynômes $A = a_0 + a_1\gamma^1 + \cdots + a_{n-1}\gamma^{n-1}$ et $B = b_0 + b_1\gamma^1 + \cdots + b_{n-1}\gamma^{n-1}$ modulo le polynôme E, chaque PE se chargera de calculer le calcul associé à un des coefficients de A (voir figure 1). Le polynôme B sera lui chargé en plusieurs coups d'horloge et les résultats intermédiaires seront sauvegardés dans le registre cyclique S.

La multiplication se fera donc en $n + 1$ coups d'horloge de la manière suivante (voire table 1) :

Cycle	$b_i =$	$s_0 =$	$s_1 =$	$s_i =$	$s_{n-1} =$
0	b_0	$s_0 = s_1 + b_0a_0$	$s_1 = s_2 + b_0a_1$	$s_i = s_{i+1} + b_0a_i$	$s_{n-1} = s_0 + b_0a_{n-1}$
1	b_1	$s_0 = s_1 + b_1a_0$	$s_1 = s_2 + b_1a_1$	$s_i = s_{i+1} + b_1a_i$	$s_{n-1} = s_0 + b_1a_{n-1}$
⋮	⋮	⋮	⋮	⋮	⋮
j	b_j	$s_0 = s_1 + b_ja_0$	$s_1 = s_2 + b_ja_1$	$s_i = s_{i+1} + b_ja_i$	$s_{n-1} = s_0 + b_ja_{n-1}$
⋮	⋮	⋮	⋮	⋮	⋮
n	0	$s_0 = s_1$	$s_1 = s_2$	$s_i = s_{i+1}$	$s_{n-1} = s_0$

TABLE 1 – table

Le registre permettant de stocker les opérations intermédiaires est un registre cyclique. Chaque mot du registre étant envoyé sur une des unités de calculs (PE). Pour la première implémentation, les polynômes a et b doivent être écrits directement dans le code VHDL. afin de vous aider, les tailles des différents chemins de données sont notés sur la figure 1.

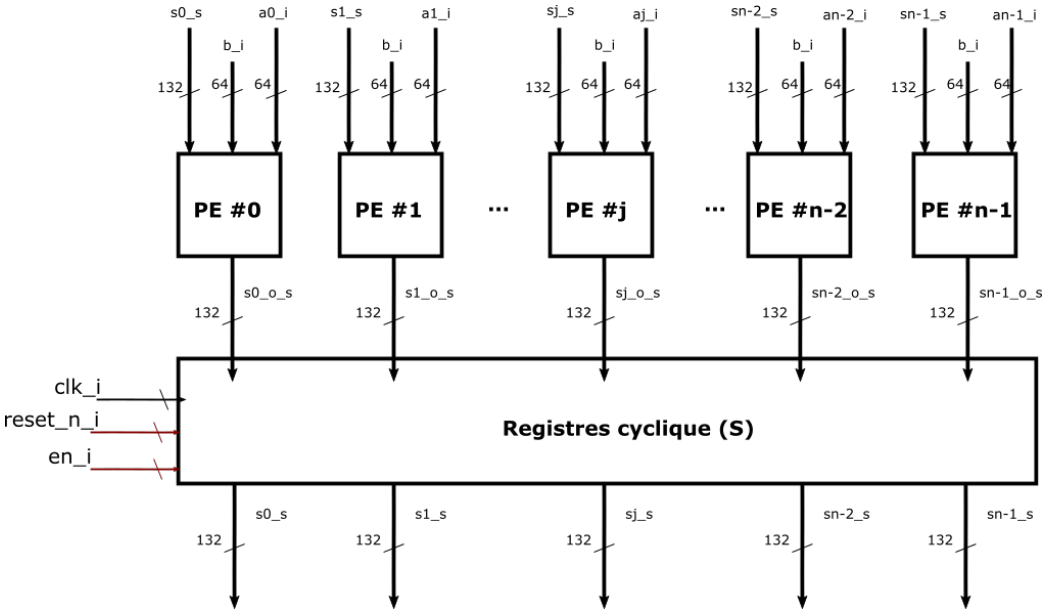

FIGURE 1 – Architecture matérielle pour la multiplication polynomiale

Les PE doivent calculer une des opérations suivantes :

— $s_o = s_i + a\_ib\_i$ si en0_i='0' et en1_i=0
— $s_o = s_i + \lambda a\_ib\_i$ si en0_i='1' et en1_i=0
— $s_o = s_i + a\_ib\_i \bmod 2^{64}$ si en0_i='0' et en1_i=1
— $s_o = s_i\lambda + a\_ib\_i \bmod 2^{64}$ si en0_i='0' et en1_i=1
L'architecture des PE est décrite dans la figure 2.

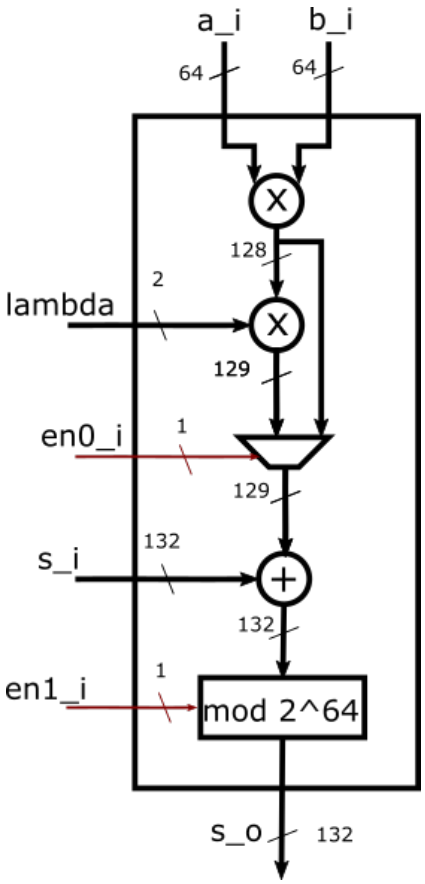

FIGURE 2 – Architecture d'une unité de calcul (PE)

L'architecture des différents éléments peut être amener à évoluer si le besoin s'en fait sentir. Cependant, vous devrez être capable de motiver les raisons qui vous ont pousser à effectuer ces modifications.

Références

[DDV20] Laurent-Stéphane Didier, Fangan-Yssouf Dosso, and Pascal Véron, *Efficient modular operations using the adapted modular number system*, Journal of Cryptographic Engineering (2020).

[Pla05] Thomas Plantard, *Arithmétique modulaire pour la cryptographie*, Ph.D. thesis, Université Montpellier II - Sciences et Techniques du Languedoc, 2005.