الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne démocratique et populaire

وزارة التعليم العالي والبحث العلمي
Ministère de l'enseignement supérieur et de la recherche scientifique

جامعة سعد دحلب البليدة
Université SAAD DAHLAB de BLIDA

كلية التكنولوجيا
Faculté de Technologie

قسم الإلكترونيك
Département d'Électronique

Mémoire de Master

Filière : Télécommunication
Spécialité : Systèmes de Télécommunication

Présenté par

HEBIB Lounis yidhir

&

BECHKOUN Mahdi

Générateur de bruit blanc sur FPGA pour le test des performances des décodeurs LDPC

Proposé par : Mr. MAAMOUN Mountassar

Année Universitaire 2019-2020

Remerciements

Tout d'abord Merci à DIEU Allah de nous avoir donné la volonté et le courage de mener ce travail jusqu'à la fin.

On tient à remercier notre encadreur Mountassar MAAMOUN de nous avoir proposé ce sujet de recherche, et de s'être toujours montré à l'écoute tout au long de la réalisation de ce mémoire.

Nous remercions les membres de jury qui nous font l'honneur de présider et d'examiner ce modeste travail.

On remercie nos parents pour leur encouragement et leur soutien.

On remercie également nos familles et nos amis pour nous avoir prodigué des conseils et des encouragements.

Merci à nos professeur et enseignants de nous avoir énormément appris par la qualité des enseignements qu'ils nous ont dispensés.

Dédicace

Je dédie ce mémoire à mes chers parents qui m'ont soutenu et encouragé pendant ces années d'études. Je prie Allah de les bénir, de veiller sur eux, en espèrent qu'ils seront toujours fiers de moi.

À mes frères, mes sœurs. A mes amis : Amine, Hmida, Khaled, Boubayou et Djalil. Qui mon encouragé au long de mes études.

Spécialement à mon binôme Lounis Yidhir, j'espère que notre travaille garde notre amitié pour toujours.

A tous mes amis sans oublier tous les étudiants de la promotion 2019/2020 de M2 ST. A mon promoteur Mr. Maamoun Mountassar qui m'a énormément aidé durant toute l'année.

Une spéciale dédicace à cette personne qui compte déjà énormément pour moi, et pour qui je porte beaucoup de tendresse, respect et d'amour. Tu as partagé avec moi les meilleurs moments de ma vie. À toi Fella Tiourguiouine Ton amour ne m'a procuré que confiance et stabilité. J'aurais tant aimé que vous soyez présente, Que Dieu ait vos âmes dans sa sainte miséricorde.

JE T'AIME.

Mahdi

Dédicace

Je dédie cette thèse A mes chers parents. Aucune dédicace ne saurait exprimer mon respect, mon amour éternel et ma considération pour le soutien, les prières et les sacrifices qu'ils ont consenti pour mon instruction et mon bien être et mon succès qu'ils ont tant souhaité.

A mes chers et adorables frère et sœurs Dany le généreux, Djamila, l'aimable et Dihia la patiente.

A mes amis et camarades Slimane, les 2 Abderrahmane, Sid Ahmed et Oussama Il me serait difficile de vous citer tous, vous êtes dans mon cœur affectueusement.

A mon binôme Mahdi Bechkoun qui été d'un apport de qualité pour le travail qu'on a, ensembles, réalisé.

A ma meilleure amie Bouchra, pour la patience et le soutien dont elle a fait preuve pendant toute la durée de cette thèse.

Lounis Yidhir

ملخص:

تم تطوير مولد ضوضاء بيضاء "غاوسي" (GNG)، لاختبار أداء مفكك تشفير LDPC المطبق على الاتصالات الرقمية. الدقة وانخفاض تعقيد الأجهزة هي أهداف مولدنا. تعتمد التقنية المستخدمة على تعديل طريقة الانعكاس، وبشكل أكثر تحديدًا على طريقة ICDF "دالة الكثافة التراكمية الانعكاسية". تم إنشاء كود MatLab لمحاكاة البرامج، وتم إنتاج وصف VHDL للتطبيقات على منصات FPFA. أظهرت نتائج المحاكاة على Matlab والتنفيذ على Xilinx FPGA أداء المولد المنتج.

الكلمات الرئيسية: GNG، ICDF، FPGA

Résumé :

Un générateur de bruit blanc "Gaussien" (GNG), pour le test des performances des décodeurs LDPC appliquées à la communication numérique, est développé. La précision et la faible complexité matérielle sont les objectifs de notre générateur. La technique utilisée est basée sur une modification de la méthode d'inversion, plus spécifiquement sur la méthode ICDF "Inversion Cumulative Density Function". Un code MatLab a été créé pour des simulations logicielles, et une description VHDL a été réalisée pour des implémentations sur des plateformes FPFA. Les résultats de la simulation sur Matlab et de l'implémentation sur FPGA de Xilinx ont démontré les performances du générateur réalisé.

Mots clés : GNG, ICDF, FPGA

Abstract :

A "Gaussian" white noise generator (GNG), for testing the performance of LDPC decoders as part of digital communication, is developed. Precision and low hardware complexity are the objectives of our generator. The technique used is based on a modification of the inversion method, more specifically on the ICDF "Inversion Cumulative Density Function" method. A MatLab code was created for software simulations, and a VHDL description was performed for FPFA implementation. The simulation MatLab results and the Xilinx FPGA implementation demonstrate the performance of the realized generator.

Keywords: GNG, ICDF, FPGA

Listes des acronymes et abréviations

ASIC: Application Specific Integrated Circuit.

BLE: Basic Logic Element.

BM: Box-Muller.

CAM: Content-Addressable Memory.

Carte SD: Card Secure Digital.

CDF: Cumulative Density Function.

CLB: Configurable Logic Block.

CLT: Central Limit.

CORDIC: Coordinate Rotation Digital Computer.

CPLD: Complex Random-Access Memory.

CPU: Central Processing Unit.

CUN : Convertisseur Uniform Normal.

DBRAM: Dual Port Block Random Access Memory.

DDR: Double Data Rate.

DMA: Direct Memory Access.

DRAM: Dynamic Random-Access Memory.

DSP: Digital Signal Processor.

EBR: Embedded Block Ram.

EPROM: Erasable Programmable Read-Only Memory.

ESB: Integrated System Block.

FF: Flip Flop.

FIFO: First In First Out.

FPGA: Field Programmable Gate Array.

GFSR: Generalized Feedback Shift Register.

GNG: Gaussian Number Generator.

GPGPU: General-Purpose Computing On Graphics Processing Units.

GPU : Graphics Processing Unit.

GRN: Gaussian Random Number.

GVAU : Generator Variable Aléatoire Uniform.

HSM: Hardware Security Module.

I/OB: Input/output Blocs.

ICDF: Inversion Cumulative Density Function.

IGCDF: Inversion Gaussian Cumulative Density Function.

JTAG: Joint Test Action Group.

LCG: Linear Congruential Generator.

LFSR: Linear Feedback Shift Register.

LUT: Look Up Table.

PDF: Probability Density Function.

PLA: Programmable Logic Array.

PLD: Random Access Memory.

RAM: Random Access Memory.

ROM: Read Only Memory.

SRAM: Static Random-Access Memory.

TG: Tausworthe Generator.

URN: Uniform Random Number.

URNG: Uniform Random Number Gaussian.

VAN : Variable Aléatoire Normal.

VAU: Variable Aléatoire Uniform.

VHDL: Very High-Density Logic.

Table des matières

Liste des figures

CHAPITRE 1

CHPAPITRE 2

CHAPITRE 3

CHAPITRE 4

Listes des tableaux

Introduction générale

Les computations sont au cœur de l'informatique. Rapides ou lents, exacts ou approximatifs, et de divers degrés de complexité, ils font partie de tout système informatique ou système de traitement de l'information.

La quête sous-jacente de meilleures performances a conduit les systèmes informatiques à se spécialiser. Un processeur graphique (GPU) peut généralement faire un meilleur travail de tâches liées aux graphiques qu'un processeur (CPU), tandis qu'un circuit spécifique à l'application (ASIC) peut faire un ensemble encore plus limité de tâches, mais encore plus efficacement. Dans ce cadre, les circuits reconfigurables tentent de trouver un équilibre entre efficacité et réutilisabilité.

Les algorithmes pour produire des variables aléatoires gaussiens varient considérablement en termes de complexité de calcul, de ressources matérielles utilisées, de vitesse et de précision, alors il n'y a pas d'algorithme parfait unique. Chaque algorithme a ses propres (dés) avantages et compromis.

Les méthodes numériques pour générer une variable aléatoire gaussienne ont une longue histoire en mathématiques et communications. La plupart des méthodes impliquent initialement de générer des échantillons d'une variable aléatoire uniforme et puis appliquer une transformation pour la convertir au gaussiens.

L'un des méthodes numériques pour la génération des variables aléatoires est la méthode ICDF (Inversion Cumulative Density Function), une telle méthode est utilisée pour des applications de système de communication. Sur un logiciel on ne peut pas générer systématiquement la variable aléatoire avec une utilisation efficace de ressources de calcul et de temps. Cependant, un résultat contraire peut être obtenu par la mise en œuvre sur

un FPGA. La conception des générateurs GRN s'est progressivement déplacée du logiciel vers les implémentations matérielles.

L'intention de ce travail est d'étudier l'architecture du générateur de variable aléatoire gaussien existante et ajouter des améliorations pour avoir un résultat plus important, ensuit implémenter une approche moins complexe synthétisable sur FPGA.

Ce mémoire est rationné en quatre chapitres :

La première partie est dédier aux présentations des architectures et les notions des calcules sur FPGA, y compris ces composants (bloc RAM, CLB et DSP…etc.)

La deuxième partie parle des architectures du générateur de variable aléatoire gaussien, comme la méthode de Central Limites, Box Muller, Wallace et la méthode d'inversion.

La troisième partie expose la méthode proposée et son algorithme d'une façon détaillé et explique aussi sa fonctionnalité logicielle et matérielle via des organigrammes et des schémas synoptiques.

La partie finale consiste à faire des simulations sur MatLab (théorique) et sur System Generator (pratique) qui fonctionne à l'aide des descriptions VHDL puis en comparant les résultats obtenus, pour distinguer si vraiment notre implémentation est juste et plus précisément avoir si notre implémentation de la méthode proposée a donné un résultat plus précieux.

Le mémoire se finalise par un bilan général en commentant sur les résultats obtenus.

Chapitre 1 : Calcul en virgule fixe sur FPGA

1.1 Introduction :

Tous les ordinateurs personnels modernes, y compris les ordinateurs de bureau, les ordinateurs portatifs (notebooks), les smartphones et les tablettes, sont des ordinateurs à usage général qui adoptent l'approche de Von Neumann', selon laquelle l'instruction et la récupération des données (data fetch) ne peuvent se faire simultanément, mais de façon séquentielle, donc en retour les performances sont limitées.

D'autre part, nous avons les Application Specific Integrated Circuits (ASIC) qui ont des tâches comme un enregistreur vocal numérique ou une mine Bitcoin à haute efficacité Un ASIC utilise une approche spatiale pour implémenter une seule application et fournit des performances maximales.

Et puis nous avons les FPGA qui agissent comme un terrain d'entente entre ces deux paradigmes architecturaux.

Depuis l'année 1985, les FPGA's ont été présentés au marché par la compagnie Xilinx. Différents FPGA à partir de cette date, ont été développés par d'autres compagnies comme Intel, Atmel, Actel, Altera, etc.

Field Programmable Device est un terme général qui se réfère à tout type de circuit intégré utilisé pour la mise en œuvre du matériel numérique, où la puce peut être configuré par l'utilisateur pour réaliser différents designs ces architectures FPGA sont dominées par des interconnexions qui les rendent beaucoup plus flexibles mais aussi beaucoup plus complexes à concevoir.

1.2 Architectures des FPGA's :

Les Field Programmable Gate Arrays (FPGA's) sont le type le plus populaire de circuits reconfigurables, le XC2064 était le premier appareil de ce type et peut retracer ses origines aux Programmable Logic Devices (PLD) ou aux les Programmable Logic Arrays (PLA). Grâce

à l'évolution technologique ces circuits en silicium n'ont cessé de se développer. Profitant de l'évolution technologique, un FPGA peut maintenant contenir plusieurs millions de portes logiques. Ces circuits sont composés d'éléments logiques interconnectés entre eux par des ressources de routage. La particularité d'un FPGA vient de sa reconfigurabilité. Les éléments logiques ainsi que le réseau d'interconnexion peuvent être reconfigurés dans le but de supporter une autre application. Il existe plusieurs types de FPGA (les SRAM-based FPGA's, les Flash-based FPGA et les Antifuse-based FPGA's). La majorité des FPGA's utilisent des mémoires statiques et sont appelés SRAM-based FPGA est sont les plus répandus [1].

Chaque élément de mémoire statique est formé de six transistors. Quatre de ces six transistors constituent deux inverseurs mis tête-bêche et les deux derniers, commandés par la ligne d'adresse A, relient les inverseurs aux lignes de données [2].

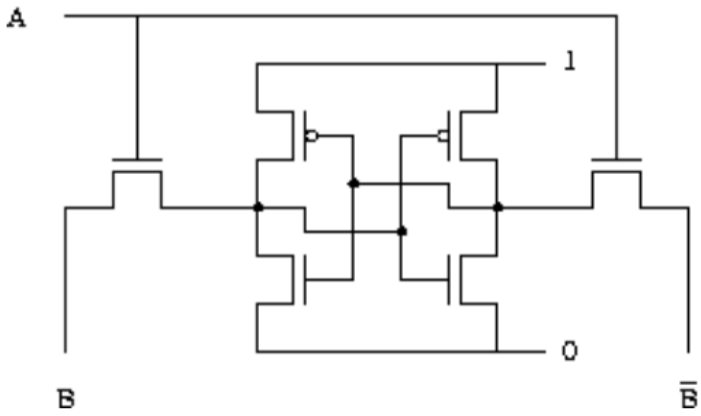

Figure 1.1. Architecture mémoire statique a 1bit.

Un FPGA est composé d'une matrice d'éléments logiques appelés Configurable Logic Block (CLB) permettant de réaliser des fonctions combinatoires et des fonctions séquentielles connectés entre eux par des ressources d'interconnexions (programmable interconnect) totalement flexibles contrairement au CPLD's [3], nous trouvons aussi des blocs entrées/sorties IOB dont le rôle est de gérer les entrées-sorties réalisant l'interface avec les modules extérieurs comme montre la figure 1.2.

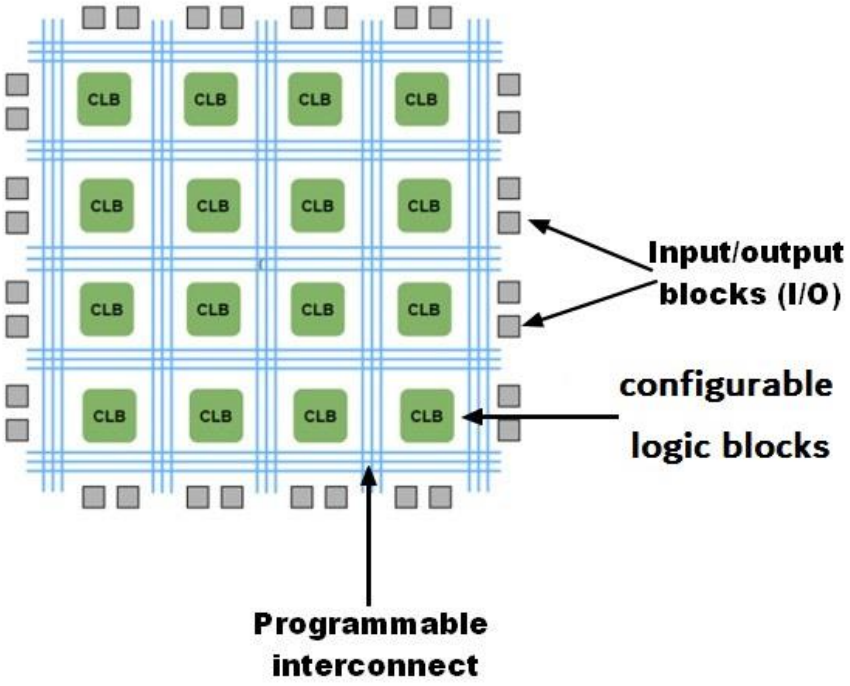

Figure 1.2. Structure basique d'une FPGA.

1.2.1 Antifuse-based FPGA's :

Ils ne peuvent être programmés qu'une seule fois. L'antifusible est un dispositif qui ne conduit pas courant initialement, mais peut être "brûlé" pour conduire le courant (le comportement antifusible est donc opposée à celle du fusible, d'où le nom). Les FPGA à base d'antifusibles ne peuvent pas être alors reprogrammé plus, car il n'y a aucun moyen de retourner un antifusible brûlé dans l'état initial. Les familles de dispositifs à base d'antifusibles comprennent Axcelerator R produit par Microsemi.

1.2.2 Flash-based FPGAs :

Tout d'abord, il ne faut pas confondre ce type de FPGA avec les FPGA's fondées sur le SRAM. Car la mémoire flash interne de ces derniers n'utilise flash qu'au démarrage pour données de chargement dans les cellules de configuration SRAM. Au contraire, un vrai flash FPGA utilise Flash comme ressource primaire pour le stockage de configuration et ne nécessite pas SRAM. Les principaux avantages de cette technologie sont une faible consommation d'énergie et meilleure tolérance aux effets du rayonnement. Les Familles de FPGA à base flash comme Igloo et ProASIC3 qui sont fabriqués aussi par Microsemi.

1.2.3 SRAM-based FPGAs

Les FPGA à base de SRAM stockent les données de configuration des cellules logiques dans la mémoire statique (organisée en un ensemble de loquets). Comme SRAM est volatile et ne peut pas conserver de données sans source d'alimentation, ces FPGA's doivent être programmés (configurés) au démarrage. Il existe deux modes de programmation de base :

Master mode (maître) lorsqu'un FPGA lit les données de configuration à partir d'une source externe, telle qu'une puce mémoire flash externe.

Slave mode (esclave) lorsqu'un FPGA est configuré par un périphérique maître externe, tel qu'un processeur. Cela peut généralement être fait via une interface de configuration dédiée ou via une interface boundary-scan (JTAG).

La mémoire de configuration basée sur SRAM est plus couramment utilisée dans les architectures reconfigurables.

Xilinx et Altera sont deux grands fabricants de FPGA basés sur SRAM. Les FPGA's de la famille Virtex Xilinx et FPGA de la famille Stratix d'Altera sont des exemples de FPGA fondées sur la SRAM. Un inconvénient de cette technologie est qu'un FPGA basé sur SRAM doit toujours être reprogrammé à la mise sous tension de la carte de circuit imprimé.

1.3 Blocs entrées et sortie (I/O)

Les blocs d'entrée et de sortie sont à peu près ce à quoi ils ressemblent. Ce sont les composants par lesquels les données sont transférées vers et hors le FPGA. L'entrée et la sortie sur la puce passent par des groupes de composants appelés banques IO (banks), qui se composent de 50 blocs IO individuels. Les blocs IO eux-mêmes sont configurables de plusieurs façons selon le type de données que l'utilisateur s'attend à recevoir ou à transmettre. Ceux-ci sont similaires aux émetteurs-récepteurs (transceiver), mais fonctionnent à des vitesses plus faibles et peuvent maintenir une grande flexibilité fonctionnelle. Une analogie simple pour distinguer les deux seraient d'envisager d'avoir un choix de véhicule entre une voiture (bloc IO) et un jet (émetteur-récepteur) pour un trajet. Même si la distance permise vous permettait de prendre de la vitesse pour décoller

dans le jet (les émetteurs-récepteurs ont des vitesses de fonctionnement minimales), ce serait très peu pratique.

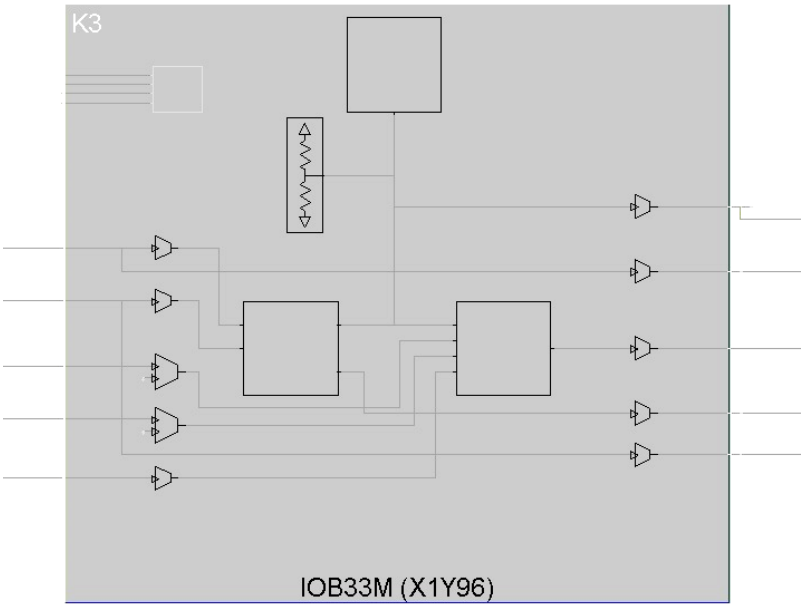

Figure 1.3. I/O bloc.

Chacun de ces composants contient son propre sous-ensemble de parties discrètes et une architecture unique qui lui permet de fonctionner [4].

1.4 **Configurable logic blocks (CLB's)**

Un bloc logique configurable (CLB) est la ressource logique de base qui se répète sur un FPGA. Lorsqu'ils sont reliés entre eux par l'acheminement des ressources, les composants des CLB exécutent des fonctions logiques complexes, mettent en œuvre des fonctions de mémoire et synchronisent le code sur le FPGA [5].

Les CLB contiennent des composants plus petits, y compris des bascules, des tables de transcodage (LUT) et des multiplexeurs.

Chaque CLB est composé de BLE's (Basic Logic Element) utilisés pour implémenter la partie logique du circuit. Un CLB est donc caractérisé par le nombre de ses entrées I et par le nombre N de BLE qu'il contient. Dans les FPGA's modernes ce nombre de BLE peut varier de 3 à 12 et chaque BLE peut être connecté à n'importe quelle entrée I du CLB ou à un autre BLE comme indiqué par la (figure 1.4.a). Un BLE est composé d'un ensemble de tables de transcodage (Look Up Table : (LUT) en anglais) ainsi que d'une bascule D pour implémenter les fonctions de bases grâce au bloc mémoire (SRAM), suivie d'un multiplexeur. Un BLE est composé de k Look Up Table ce qui permet d'implémenter une fonction logique à k entrées et une sortie et nécessite 2^k blocs mémoire pour la configuration. La bascule D quant à elle permet de choisir entre un fonctionnement logique ou séquentiel. Ensemble les LUT et la bascule D forme un Basic Logic Element (BLE illustré figure 1.4.b). Des études ont montré que la meilleure architecture de LUT en termes de performances est une LUT ayant 4 entrées.

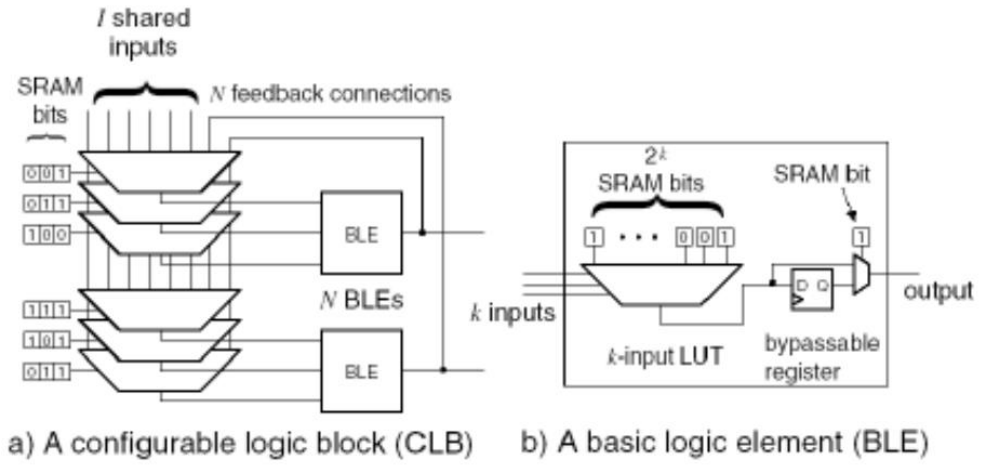

Figure 1.4. Architecture d'un CLB/BLE.

Les éléments logiques sont connectés entre eux et aux entrées/sorties grâce au réseau d'interconnexion. Ce réseau est programmé de la même manière que les éléments logiques avec des RAM's statiques. L'ensemble permet de configurer complètement le FPGA. En théorie, il est donc possible de charger n'importe quelle fonction logique sur un FPGA. Enfin, la reprogrammation de l'ensemble des SRAM's permet de reconfigurer entièrement le circuit et donc de changer l'application ou la fonction du FPGA. Pour finir, lors de la configuration du FPGA, la netlist contenant l'ensemble des connexions est envoyée au FPGA. Le placement des blocs logiques et la manière dont ils sont interconnectés est alors défini par un algorithme de placement et de routage [1].

Xilinx regroupent les éléments logiques en blocs logiques configurables (CLB) pas comme Intel qui regroupent les ressources logiques d'une manière légèrement différente. Une structure simplifie est présentée à la figure 1.5. À son tour, chaque CLB est constitué de deux slices, qui ne peuvent pas communiquer directement entre eux, mais peuvent communiquer à travers le switch matrix (des matrices d'interconnexion) vers les slices qui sont directement en dessous et au-dessus d'elles. Enfin, Un slice est repartis en 4, chaque partie contient 1 LUT, 2 Bascule, 1 multiplixuer et une carry logic qui represent un circuit pour la propagation de la retenu.

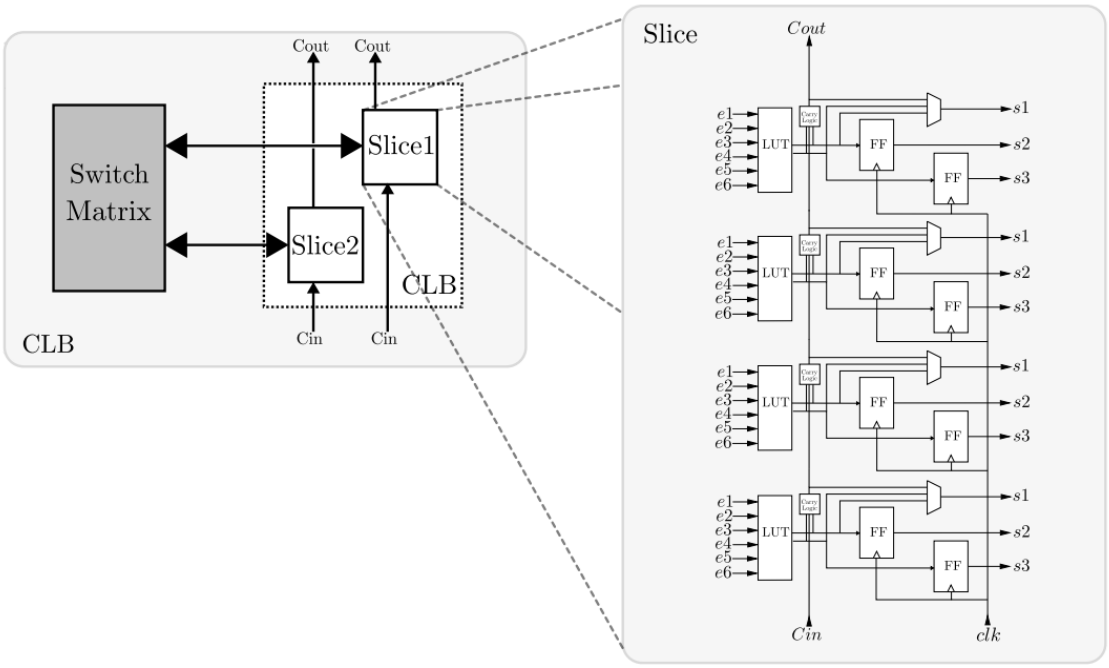

Figure 1.5. Organisation des ressources logique sur un appareil de Xilinx.

Il existe deux types de tranche (slice), SLICEL et SLICEM, où SLICEM offre des fonctionnalités supplémentaires qui lui permet d'être utilisé comme une RAM distribuée (256 bits), ou comme un registre de décalage (32 bits de décalage). Un zoom sur une vue simplifiée d'une coupe est illustré à la figure 1.6.

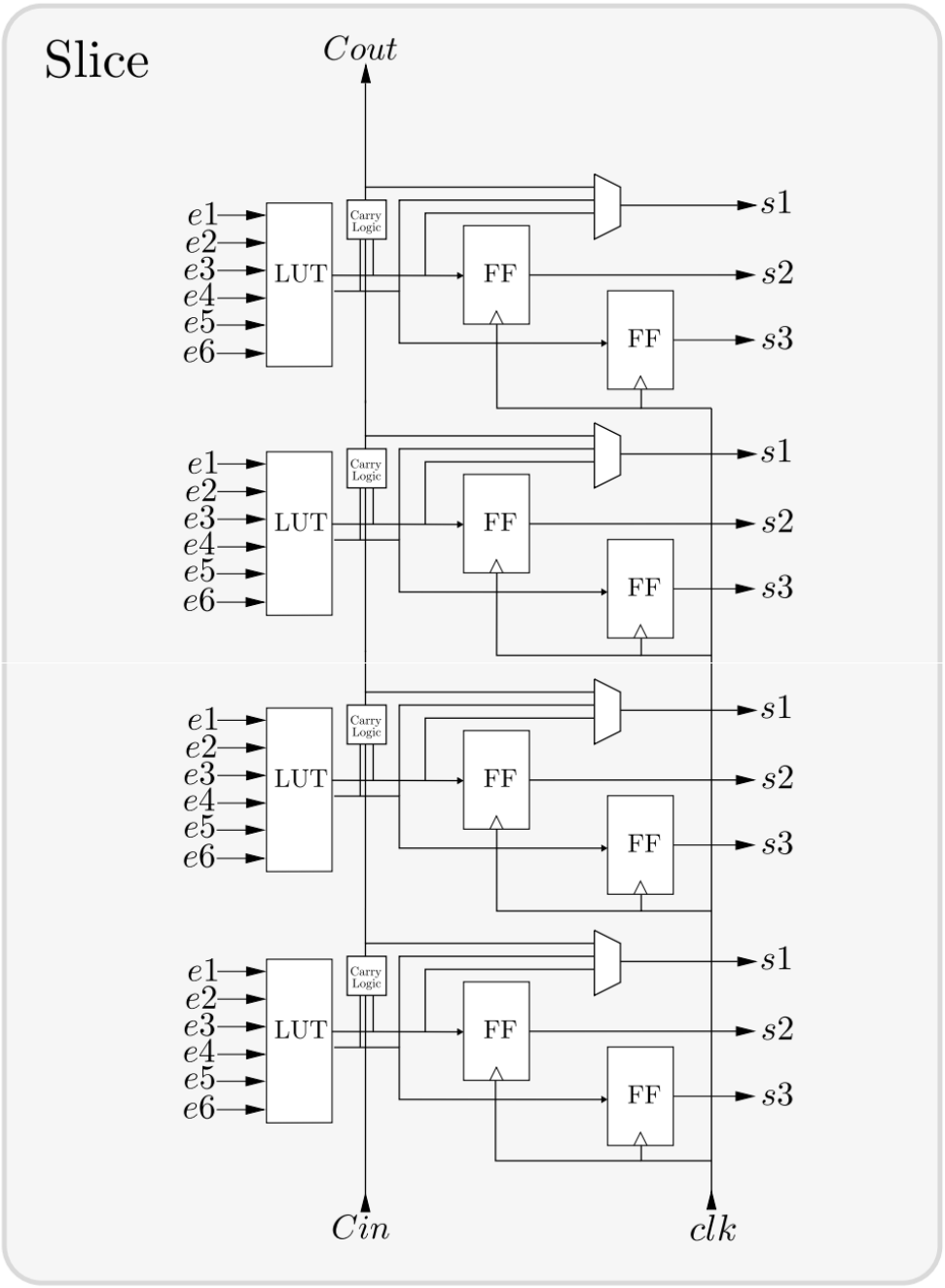

Figure 1.6. Présentation d'un SLICE sur les périphériques Xilinx.

Les éléments logiques eux-mêmes sont des tables de transcodage (LUT's) à 6 entrées. Ils peuvent être utilisés dans une variété de modes étant capable de mise en œuvre de toute fonction de jusqu'à 6 entrées indépendantes, ou 2 fonctions d'entrée indépendantes,

car chaque LUT a deux sorties. En utilisant les deux couches de multiplexeurs, les LUT's d'une Slice peuvent être combinés pour implémenter des fonctions à 7 ou 8 entrées, sans payer le coût du délai de routage accru en dehors de la tranche. Les multiplexeurs à l'intérieur d'une tranche peuvent également être utilisés pour la mise en œuvre efficace de multiplexeurs larges.

Les éléments logiques peuvent également être utilisés pour la mise en œuvre d'addition ou soustractions, en profitant également d'un circuit pour la propagation de la retenu (carry logic) rapide dédiée. Elle s'étend sur une Slice entière et est disposée verticalement dans l'appareil. Des chaînes de propagation (carry chaines) plus longues peuvent être formées, car la propagation se poursuit jusqu'à la tranche suivante, évitant ainsi les longs délais du réseau de routage général. Cette logique de propagation peut également être manipulée, utile pour la mise en œuvre de fonctions spécifiques, en conjonction avec les LUT.

Enfin, dans chaque tranche, il y a deux bascules, qui peuvent stocker la sortie de chacun des LUT. Lorsqu'il est laissé inutilisé par la LUT correspondante, un FF (Flip-Flop) peut stocker certaines données provenant d'une source différente, tant que la configuration des ressources de routage le permet [6].

1.4.1 LUT look up table

La partie fondamentale du FPGA est appelée « look up table », ou LUT qui agit en tant que générateur de fonction, chez la plupart des fabricants de FPGA une LUT contient 16 bits, A chaque LUT peut être adjoint un registre piloté par une horloge et un multiplexeur permettant d'utiliser le registre ou non.

Chaque fabricant de FPGA a par la suite réalisé des regroupements de LUT's. Et d'ajout de capacité supplémentaire, aujourd'hui, la structure la plus utilisée est basée sur une look-up Table (RAM) pour implémenter une fonction combinatoire plus une bascule D [3].

Look up table est en fait une table de vérité qui est chargé avec des valeurs comme figure 1.7 affiche, il peut être pensé comme un petit morceau de RAM qui est chargé chaque fois que l'alimentation la puce FPGA est en place. Il définit et dirige le

comportement de la logique combinatoire de la puce basée sur le code VHDL ou Verilog, en se référant aux valeurs prédéterminées pour produire les résultats souhaités.

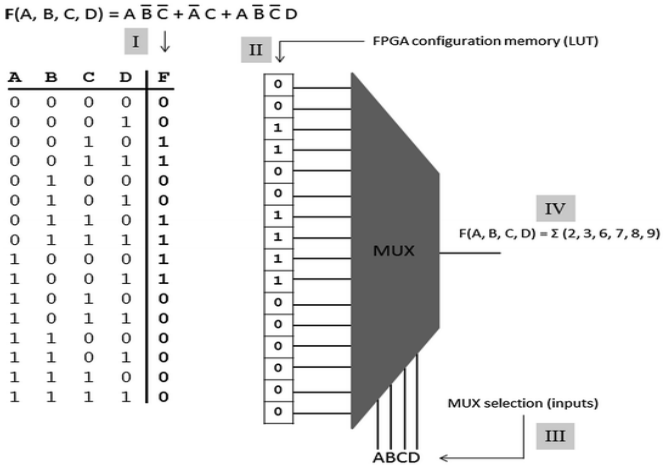

Figure 1.7. Modèle de travail LUT.

Cela signifie que les entrées de LUT sont essentiellement les lignes d'adresse pour une cellule de RAM d'un bit de large, En tant que tel, s'il utilise une porte logique de deux entrées, il serait logique qu'il soit capable de produire ou d'accommoder pour quatre scénarios combinatoires différents, ce qui le rend approprié pour une RAM 4 x 1 bit, donc l'augmentation du nombre d'entrées doit être suivie par une modification ultérieure de la RAM nécessaire.

En conclusion, LUT's servent essentiellement à agir comme des portes logiques dans diverses combinaisons. Au lieu d'avoir à connecter un certain nombre de portes NAND et NOR différentes, l'utilisation d'une look-up-table permet d'obtenir toutes les combinaisons possibles.

Dans une puce FPGA, la fonctionnalité est généralement divisée en un certain nombre de blocs logiques configurables, chacun pouvant fonctionner indépendamment de manière cohérente, chaque bloc logique a son propre ensemble de flip flops ainsi que des look-up-table et c'est ce qui définit la fonction d'une puce FPGA et la différence entre eux et leurs divers buts désignés.

Le long de ces cellules de mémoire y a des multiplexeurs de sorte que le nombre d'entrées d'alimentation sur la LUT sera le principal déterminant de la taille de l'appareil. Après réception des entrées, l'appareil recherche la valeur de sortie correspondante dans le bloc SRAM et la produit à la broche de sortie [7].

1.4.2 Flip flop

La structure de base d'une bascule flip flop comprend une entrée de données (data input), une entrée d'horloge (clock input), une activation d'horloge (clock enable), une réinitialisation (reset) et une sortie de données (data output). Pendant le fonctionnement normal, toute valeur au port d'entrée de données est verrouillée et passée à la sortie sur chaque impulsion de l'horloge. L'objectif de la broche d'activation (clock enable) de l'horloge est de permettre à la bascule de conserver une valeur spécifique pour plus d'une impulsion d'horloge. Les nouvelles entrées de données sont seulement verrouillées et passées au port de sortie de données lorsque l'horloge et l'horloge activent (clock enable) sont égales à un [8].

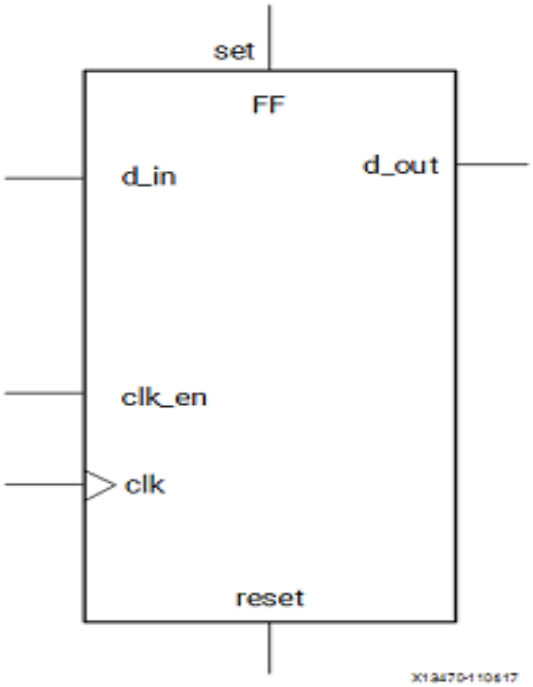

Figure 1.8. Structure d'une bascule Flip Flop.

1.5 DSP (Digital Signal Processor)

Un DSP est un type particulier de microprocesseur. Il se caractérise par le fait qu'il intègre un ensemble de fonctions spéciales. Ces fonctions sont destinées à le rendre particulièrement performant dans le domaine du traitement numérique du signal. Comme un microprocesseur classique, un DSP est mis en œuvre en lui associant de la mémoire (RAM, ROM) et des périphériques. Un DSP typique a plutôt vocation à servir dans des systèmes de traitements autonomes. Il se présente donc généralement sous la forme d'un microcontrôleur intégrant, selon les marques et les gammes des constructeurs, de la mémoire, des timers, des ports série synchrones rapides, des contrôleurs DMA, des ports d'E/S divers [9].

Les FPGA's sont efficaces pour les applications de traitement numérique du signal car ils peuvent implémenter des algorithmes personnalisés et entièrement parallèles. Les applications DSP utilisent de nombreux binaires multiplicateurs et accumulateurs. Les appareils Versal ™ disposent de nombreux DSP dédiés à faible puissance combinant haute vitesse et petite taille tout en conservant la flexibilité de conception du système. Les ressources DSP améliorent la vitesse et l'efficacité de nombreuses applications au-delà du traitement numérique du signal, comme les générateurs d'adresses mémoire et les multiplexeurs de bus larges (Dynamic Bus Shifters) [10].

La plupart des dispositifs FPGA actuels contiennent des blocs DSP dédiés qui sont considérés comme Hard-core blocs, sont conçus et intégrés de manière optimale dans les dispositifs pour faciliter la mise en œuvre de fonctions spécifiques qui, autrement, nécessitent un nombre beaucoup plus important de LUT à mettre en œuvre et offrent la possibilité de mettre en œuvre des applications qui demande de mémoire [11].

Dans les FPGA modernes de Xilinx et Altera sont équipés par des DSP's pour le calcul arithmétique afin d'accélérer les applications de traitement numérique du signal. Ces blocs peuvent être utilisés pour construire des implémentations plus efficaces en termes de performance et permettre en même temps de réduire la surface utilisée sur FPGA's.

1.5.1 DSP slice

Un DSP slice est un processeur élémentaire pré du circuit DSP et injecter dans les FPGA pour faire les opérations de multiplication et d'addition, la figure 1.9 illustre la structure générique module DSP dans les FPGA's modernes. Ces modules peuvent fonctionner via des entrées externes A, B et C ainsi qu'une valeur de retour P [12].

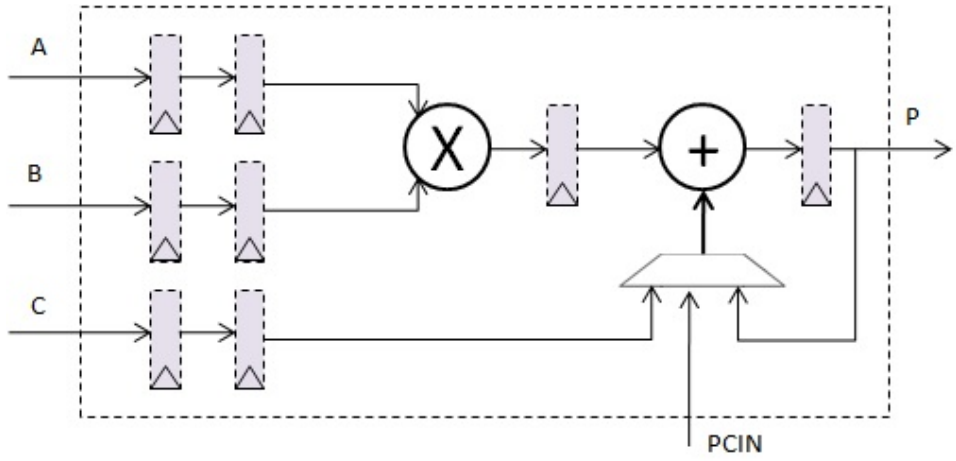

Figure 1.9. Structure des DSP's dans les FPGAs modernes.

a. DSP58

DSP58 se compose d'un multiplicateur suivi d'un accumulateur comme illustrer dans la Figure 1.10. Au moins trois registres de pipeline sont nécessaires pour que les opérations

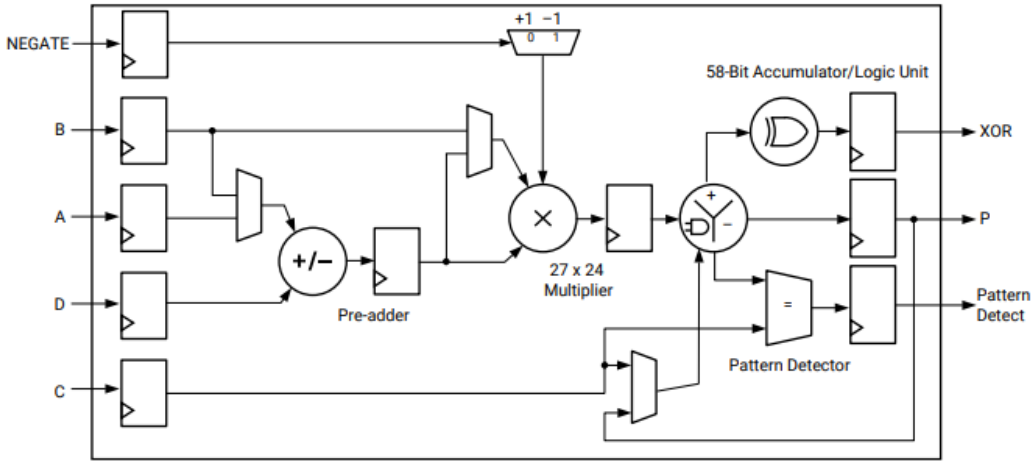

Figure 1.10. Structure des DSP's dans les FPGAs modernes.

de multiplication et de multiplication-accumulation s'exécutent à pleine vitesse. L'opération de multiplication dans la première étape génère deux produits partiels qui doivent être ajoutés ensemble dans la Deuxième étape.

Lorsqu'il n'existe qu'un ou deux registres dans la conception du multiplicateur, le registre M doit toujours être utilisé pour économiser l'énergie et améliorer les performances. Les opérations d'ajout / secondaire et d'unité logique nécessitent au moins deux registres de pipeline (entrée, sortie) pour s'exécuter à pleine vitesse. Les capacités en cascade du DSP58 sont extrêmement efficaces pour mettre en œuvre des pipelines à grande vitesse filtres construits sur les cascades d'addition au lieu d'arbres d'addition. Les multiplexeurs sont contrôlés avec des signaux de contrôle dynamiques, tels que OPMODE, ALUMODE et CARRYINSEL.

1.5.2 Avantages des DSP slice

Tous les systèmes à bases de DSP bénéficient des avantages suivants :

- Souplesse de la programmation
- Implémentation d'algorithmes adaptatifs
- Stabilité
- Répétabilité, reproductibilité

1.6 Block RAM

Les ressources mémoire sont essentielles pour de nombreuses applications FPGA avancées. Il existe deux principaux types de mémoire dans les FPGA's, la mémoire distribuée et la mémoire bloc. La mémoire distribuée tire parti du fait que les éléments LUT sont des implémentations de blocs de mémoire SRAM. La mémoire de bloc est l'implémentation de blocs de mémoire SRAM dédiés dans le FPGA.

Les éléments de mémoire intégrés dans les FPGA sont généralement appelés blocs RAM, blocs système intégrés (ESB), RAM système et mémoire adressable par contenu (CAM). Les périphériques FPGA's hautes performances ont généralement un grand nombre de banques de mémoire dédiées en plus de la fonctionnalité de mémoire distribuée inhérente. Les blocs de mémoire dédiés dans une configuration à deux ports peuvent prendre en charge les lectures et écritures asynchrones et synchrones. D'autres capacités potentielles incluent la parité, le contrôle de la synchronisation et la fonctionnalité de réinitialisation. Ils peuvent être configurés pour prendre en charge une large gamme d'applications.

Block RAMs (ou BRAM) signifies Block Random Access Memory. Les blocs RAM sont utilisés pour stocker de grandes quantités de données à l'intérieur du FPGA. Ils sont l'un des quatre composants communément identifiés sur une fiche technique FPGA. Les trois autres sont des bascules, des tables de consultation (LUT) et des processeurs de signaux numériques (DSP). Habituellement, plus le FPGA est gros et cher, plus il y aura de bloc de RAM.

Une RAM de bloc (parfois appelée mémoire embarquée, ou RAM de bloc embarquée (EBR)), est une partie discrète d'un FPGA, ce qui signifie qu'il n'y a qu'un certain nombre disponible sur la puce. Chaque FPGA a un montant différent, donc selon l'application, il peut avoir besoin de plus ou moins de Block RAM. Comme expliqué déjà, il stocke une "grande" quantité de données à l'intérieur du FPGA. Il est également possible de stocker des données en dehors de la FPGA, mais cela serait fait avec un dispositif comme un SRAM, DRAM, EPROM, carte SD, etc.

Les RAM en blocs sont de taille limitée, 4/8/16/32 kb (kilobits) sont communs. Ils ont une largeur et une profondeur personnalisables, la figure 1.11 illustre un schéma simplifier d'un BRAM.

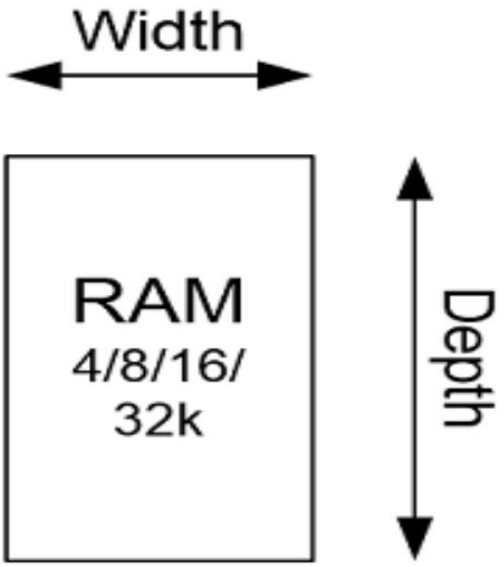

Figure 1.11. Schéma simplifier d'un BRAM.

1.6.1 BRAM a un port :

La configuration Single Port Block RAM comme montre la figure 1.12 est utile quand il n'y a qu'une seule interface qui a besoin de récupérer des données. C'est aussi la configuration la plus simple et l'utile pour certaines applications. Un exemple serait de stocker des données en lecture seule qui sont écrites à une valeur fixe lorsque le FPGA est programmé.

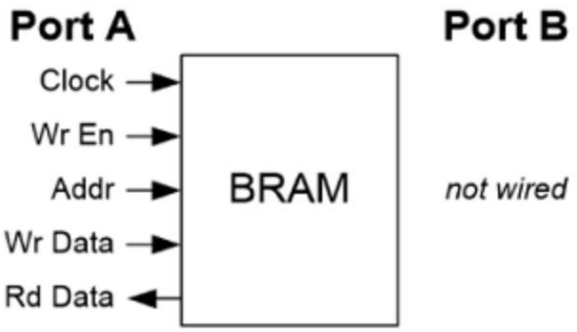

Figure 1.12. Schéma simplifier d'un BRAM a un port.

Si une application a quelques paramètres d'étalonnage qui seront écrits une fois et lus pendant le démarrage, alors un bloc de port unique RAM va faire le travail fait d'une manière idéale.

1.6.2 BRAM a deux ports

La RAM Dual Port Block (ou DPRAM) se comporte exactement de la même manière que la configuration du port unique, sauf qu'elle a un autre port disponible pour la lecture et l'écriture de données. Le port A et le port B se comportent exactement de la même façon. Le port A peut effectuer une lecture sur l'adresse 0 sur le même cycle d'horloge que le port B écrit sur l'adresse 200. Par conséquent, une DPRAM est capable d'écrire sur une adresse tout en lisant à partir d'une adresse complètement différente.

b. Utilisation de RAMB36 dans un véritable flux de données à double port

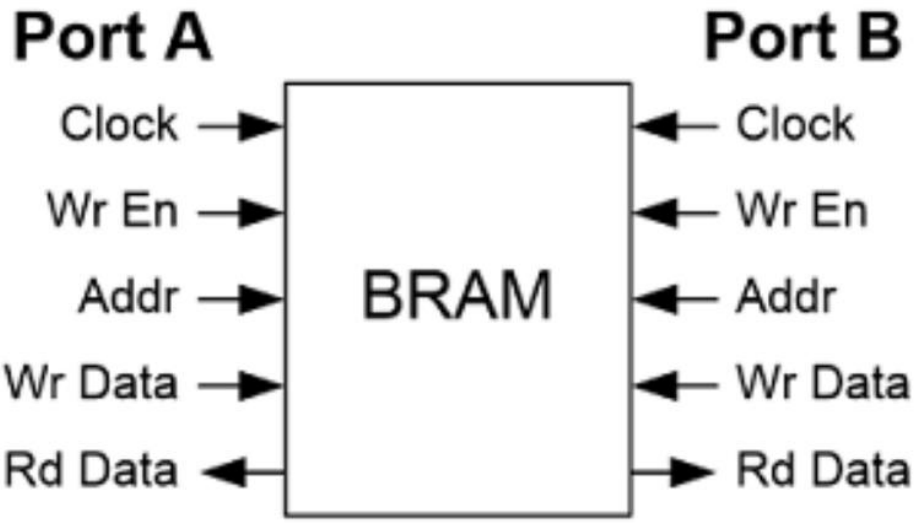

Figure 1.13. Schéma simplifier d'un BRAM deux port.

Les véritables mémoires à double port RAM bloc de 36 Ko à double port se composent d'une zone de stockage de 36 Ko et deux ports d'accès complètement indépendants, A et B. De même, chaque bloc de 18 Kb RAM double port la mémoire se compose d'une zone de stockage de 18 Ko et de deux ports d'accès totalement indépendants, A et B. La structure est entièrement symétrique et les deux ports sont interchangeables. La figure 1.14 suivante illustre le véritable flux de données à double accès d'un RAMB36 [13].

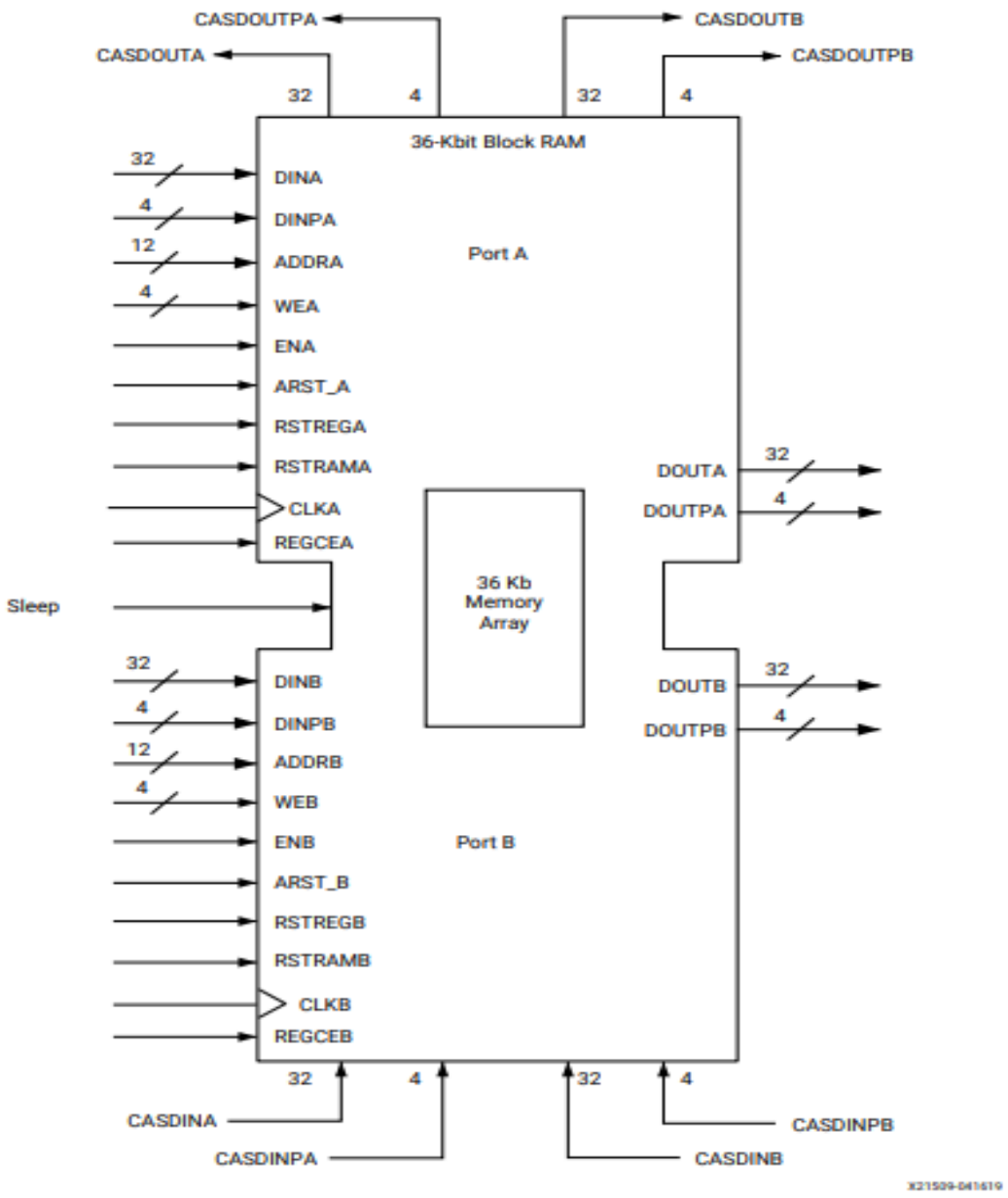

Figure 1.14. RAMB36de 36 Ko à deux ports d'accès.

Les données peuvent être écrites sur l'un ou les deux ports et peuvent être lues à partir de l'un ou des deux ports. Chaque écriture le fonctionnement est synchrone et chaque port a sa propre adresse, entrée de données, sortie de données, horloge, activation de l'horloge et autorisation de l'écriture. Les opérations de lecture et d'écriture sont synchrones et nécessitent une horloge bord. Il n'y a pas de moniteur dédié pour arbitrer l'effet d'adresses identiques sur les deux ports [10].

Le tableau suivant répertorie les fonctions du port et leur description :

Port Function	Description
ARST_[A|B]	Asynchronous reset that resets the output register for port A and B to all zeros.
DIN[A|B]	Data input bus.
DINP[A|B][1]	Data input parity bus. Can be used for additional data inputs.
ADDR[A|B]	Address bus.
WE[A|B]	Byte-wide write enable.
EN[A|B]	When inactive, no data is written to the block RAM and the output bus remains in its previous state.
RSTREG[A|B]	Synchronous set/reset of the output registers (DO_REG = 1). The RSTREG_PRIORITY attribute determines the priority over REGCE.
RSTRAM[A|B]	Synchronous set/reset of the output data latches.
CLK[A|B]	Clock input.
DOUT[A|B]	Data output bus.
DOUTP[A|B][1]	Data output parity bus. Can be used for additional data outputs.
REGCE[A|B]	Output register clock enable.
CASDIN[A|B]	Cascade data input bus.
CASDINP[A|B]	Cascade parity input bus.
CASDOUT[A|B]	Cascade data output bus.
CASDOUTP[A|B]	Cascade parity output bus.
SLEEP	Dynamic shutdown power saving. If SLEEP is active, the block is in power saving mode.

Tableau 1.1. Fonctions des ports et leurs descriptions.

1.6.3 FIFO BRAM

FIFO signifie (First In First Out) et ils sont utilisés partout dans la conception FPGA. Si certaines données doivent être tamponnées entre deux interfaces, un FIFO est utilisé. Aussi que pour croiser des domaines d'horloge, ou pour tamponner une ligne de données d'image et la manipuler, ou bien pour envoyer des données hors puce vers une mémoire DDR, tout cela nécessite l'utilisation d'un bloc FIFO de RAM.

Il est très important de comprendre les FIFO's, car elles sont vraiment fondamentales dans la création de l'FPGA [14].

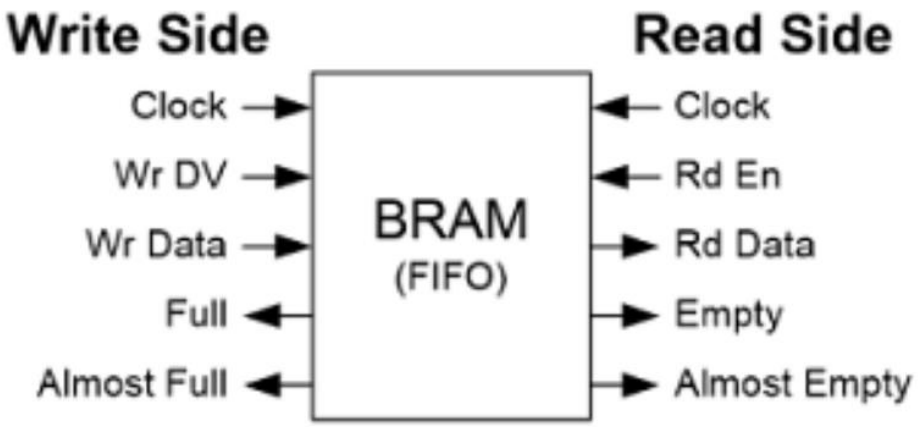

Figure 1.15. Schéma simplifier d'un BRAM FIFO.

1.7 CONCLUSION

En conclusion, nous avons présenté dans cette section les architectures FPGA et ces types et les composantes fondamentaux d'une puce FPGA, on constate que ces architectures diffèrent dans la manière dont sont interconnectés les blocs logiques au réseau d'interconnexion. On a vu que les CLB's est la base d'une FPGA est c'est une ressource logique qui se répète dans cette dernière, ces CLB contient des LUT's et des Flip Flop qui sont le cœur du CLB, les LUT's agit comme un générateur de fonction et qui est en fait une table de vérité qui sert autant que des portes logiques. Revenons au flip flop qui est simplement une bascule avec le rôle de stockées les données d'entrée quand il y a une transition de « 0 » à « 1 ».

On a prêté attention au Digital Signal Processing DSP slice qui sert à remplacer l'utilisation d'un nombre important des LUT's ouvrant la porte a les applications qui demande la mémoire. Et ce qui concerne les BRAM's ou bien EBR qui sert simplement à stocker des larges paquets de donnée.

La compréhension de l'utilisation des ressources est extrêmement utile pendant le développement, en particulier lors de l'optimisation de la taille et de la vitesse. Les FPGA's modernes sont des dispositifs sophistiqués et performants qui peuvent être quelque peu intimidants pour ceux qui sont habitués à utiliser des microcontrôleurs pour recueillir des données, contrôler les ASIC et effectuer des opérations mathématiques

Chapitre 2 : Architectures matérielles de base des générateurs de nombres aléatoires gaussiens pour le test des décodeurs LDPC

Chapitre 2 : Architectures matérielles de base des générateurs de nombres aléatoires gaussiens pour le test des décodeurs LDPC

2.1 Introduction

La disponibilité de nombres aléatoires gaussiens de haute qualité est essentielle à de nombreuses applications de simulation, graphiques et de Monte Carlo par exemple largement utilisées dans les domaines des mathématiques financières et des technologies de communication. Actuellement, la majorité de ces simulations sont effectuées à l'aide de systèmes basés sur des microprocesseurs, des processeurs de signaux numériques, une unité de traitement graphique à usage général (GPGPU) ou d'autres dispositifs programmables par logiciel. Dans ces systèmes, les fonctions trigonométriques, exponentielles et autres impliquées dans de nombreuses méthodes d'obtention de variables aléatoires gaussiennes peuvent être exécutées à l'aide de la bibliothèques logicielles. En conséquence, peu de recherches ont été rapportées concernant les méthodes matérielles efficaces pour la mise en œuvre des générateurs de bruit gaussiens. Cependant, des implémentations matérielles bien optimisées peuvent souvent faire fonctionner un ou plusieurs ordres de grandeur plus rapidement que des implémentations logicielles optimisées de la même manière. Les progrès récents dans la technologie de réseau de portes programmables sur le terrain (FPGA) ont considérablement amélioré les performances et la rentabilité des implémentations matérielles, et ils nous incitent fortement à réexaminer la question de la génération de bruit gaussien dans le matériel.

Les méthodes analogiques pour générer des échantillons aléatoires gaussiens sont basées sur certains phénomènes physiques et peuvent générer des échantillons vraiment aléatoires, mais elles sont soumises à l'influence des facteurs externes tels que la température, et elles fournissent des faibles débits. D'autre part, les méthodes numériques sont préférées en raison de leur robustesse, leur flexibilité, leur vitesse et de leur comportement prévisible et contrôlable.

Au cours de la dernière décennie, la conception des générateurs GRN est progressivement passée des logiciels aux implémentations matérielles, leurs algorithmes varient considérablement en termes de complexité informatique, des ressources matérielles utilisées, de vitesse et de précision Thomas et al. [2007] ont fourni une classification plus large des algorithmes GRN en deux grandes catégories : les algorithmes exacts et approximatifs [15].

Et ont également fourni une classification plus spécifique à quatre catégories :

- Inversion methods
- Transformation methods
- Rejection methods
- Recursion methods

2.2 Variable aléatoire

Une variable aléatoire est une application définie sur l'ensemble des éventualités, c'est-à-dire l'ensemble des résultats possibles d'une expérience aléatoire.

2.2.1 Variable aléatoire uniforme

Les variables aléatoires uniformes sont les plus simples à générer, car toutes les valeurs possibles possèdent la même probabilité d'apparition. La densité de probabilité de la loi uniforme continue est exprimée par le système d'équations.

La manière la plus simple de générer une variable aléatoire uniforme est l'utilisation des registres à décalages à rétroaction linéaire ou en anglais LFSR (Linear Feedback Shift Registre).

a. Loi uniform

La loi uniforme est la loi de probabilité continue la plus simple, définie sur un intervalle borné $[a, b]$ [15].

b. Densité de probabilité

La densité de probabilité de la loi uniforme continue est une fonction porte sur l'intervalle $[a, b]$ [15] :

$$f(x) = \begin{cases} \frac{1}{b-a} \; pour \; a \leq x < b \\ 0 \qquad\quad sinon \end{cases} \tag{2.1}$$

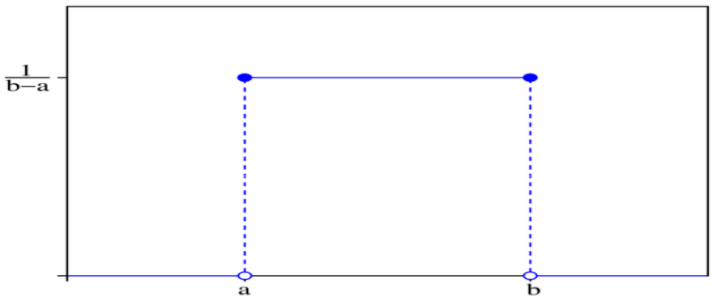

Figure 2.1. PDF de loi uniforme.

2.2.2 Loi normale

La loi normale ou loi de Laplace-Gauss :

Les premiers statisticiens ont constaté que de nombreuses distributions statistiques observées pouvaient être décrites et modélisées par une loi nommée par conséquent loi normale (cela ne signifie pas pour autant que les autres distributions soient anormales) [20].

a. Densité de probabilité

Une variable aléatoire X suit une loi normale si sa densité de probabilité a pour équation :

$$p(x) = \frac{1}{\sigma.\sqrt{2\pi}}.e - \frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2 \qquad (2.2)$$

Dont le graphe est la fameuse "courbe en cloche", et qui dépend de deux paramètres μ et σ qui ne sont autres, comme on le montrera au paragraphe suivant, que la moyenne et l'écart-type de X.

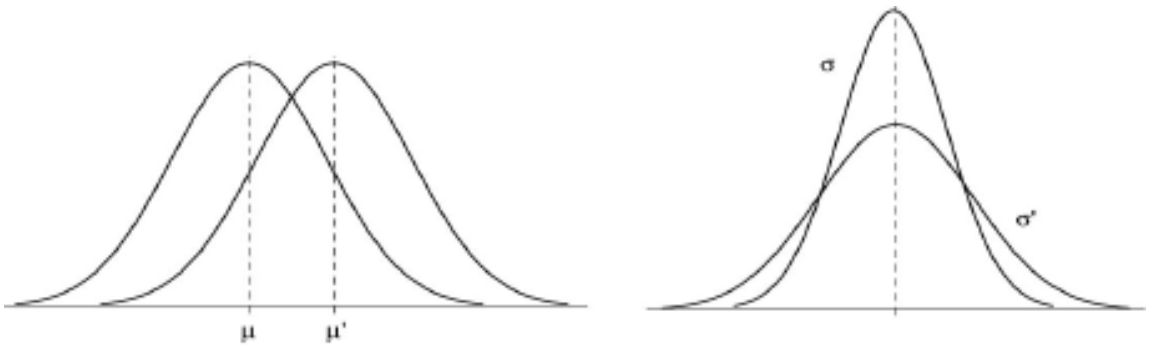

Figure 2.2. Courbe en cloche.

La distribution est symétrique par rapport à μ qui caractérise donc la tendance centrale. Quant à σ, il caractérise la dispersion de la distribution. Plus il est grand, plus la distribution est étalée de part et d'autre de μ [21].

Une des méthodes proposées pour concevoir un GNG matériel est la mise en œuvre direct du théorème de limite centrale (CLT). Les GNG de haute précision doivent résumer un grand nombre d'échantillons. Mais cette méthode n'est pas adaptée aux applications à grande vitesse [22].

2.3 **Bruit blanc**

Le bruit blanc provient de l'agitation thermique des électrons dans les conducteurs et les matériaux. Par analogie avec la lumière blanche, qui s'obtient en superposant les ondes du spectre visible, le bruit blanc est une tonalité générée de façon aléatoire combinant toutes les fréquences sonores simultanément [18].

En traitement du signal le bruit blanc est un signal aléatoire qui n'est pas corrèle, c'est-adire, que la valeur du bruit a un instant donné, n'est pas corrélée avec ses valeurs a d'autres instants, la fonction d'autocorrélation d'un bruit blanc est alors, la fonction Dirac. En d'autres termes, elle est maximale en zéro et nul en tout autre instant, Par conséquent, la densité spectrale de puissance d'un bruit blanc est une constante sur toutes les

fréquences. Il est une construction purement théorique, un bruit blanc est alors un signal aléatoire de moyenne nulle ayant la fonction Dirac comme fonction d'autocorrélation.

2.3.1 Générateur de bruit blanc

La figure 2.3 montre un schéma synoptique d'un générateur bruit blanc :

Principe de Générateur de Bruit Blanc : le but de ce générateur est la conversion de variable aléatoire uniforme vers variable aléatoire normale.

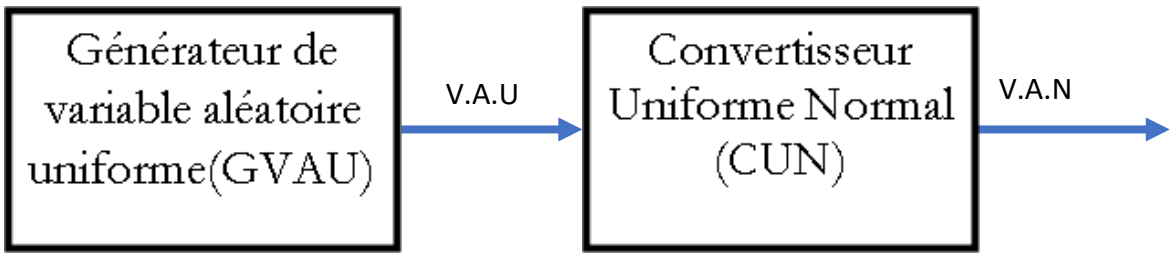

Figure 2.3. Schéma synoptique de Générateur de bruit blanc.

2.4 The Central Limit Théorème (CLT)

Le théorème central de limite est la théorie la plus fondamentale dans les statistiques modernes. Sans ce théorème, il n'existerait pas de tests paramétriques fondés sur supposition que les données de l'échantillon proviennent d'une population ayant des paramètres fixes déterminant sa distribution probabiliste.

Avec le théorème central limite, les tests paramétriques ont une puissance statistique plus élevée que les tests non paramétriques, qui ne nécessitent pas d'hypothèses de distribution de probabilité. Actuellement, plusieurs tests paramétriques sont utilisés pour

évaluer la validité statistique des études cliniques effectuées par des chercheurs médicaux ; cependant, la plupart des chercheurs ne sont pas conscients de la valeur du théorème central de la limite, malgré leur utilisation systématique des tests paramétriques. Ainsi, les chercheurs cliniques gagneraient à savoir ce qu'est le théorème central de la limite et comment il est devenu la base des tests paramétriques.

En statistique, une population est l'ensemble de tous les éléments ou événements d'intérêt. En réalité, cependant, la collecte de tous ces éléments de la population exige un effort considérable et est souvent impossible. Par exemple, il n'est pas possible d'étudier la compétence de chaque anesthésiste, dans le monde entier. Cependant, pour faire des inférences concernant la population, un sous-ensemble de la population (échantillon) peut être utilisé. Un échantillon de taille suffisante, sélectionné au hasard, peut être utilisé pour estimer les paramètres de la population à l'aide de statistiques inférentielles. Un nombre limité d'échantillons peuvent être obtenus de la population en fonction de la taille de l'échantillon et de la population elle-même [15].

Bien que connu depuis plus de deux siècles et très simple à mettre en œuvre, le théorème de limite centrale (CLT) n'a pas été utilisé pour la génération des GRN's précis jusqu'à ces derniers temps. Cela est dû au fait que pour tout nombre fini d'ajouts, le PDF de CLT s'écarte du comportement gaussien idéal [16].

Dans l'étude de la théorie des probabilités, le théorème central limite (CLT) indique que la distribution de l'échantillon se rapproche d'une distribution normale (également appelée « courbe en cloche ») lorsque la taille de l'échantillon devient plus grande, en supposant que tous les échantillons sont de taille identique, et quelle que soit la forme de la répartition de la population.

Autrement dit, la CLT est une théorie statistique selon laquelle, tous les échantillons suivront un modèle de distribution normal approximatif, toutes les variances étant

approximativement égales à la variance de la population, divisée par la taille de chaque échantillon.

Bien que ce concept ait été développé pour la première fois par Abraham de Moivre en 1733, il n'a été officiellement nommé qu'en 1930, lorsque le célèbre mathématicien hongrois George Polya l'a officiellement surnommé le théorème central des limites [17].

En règle générale, les tailles d'échantillon égales ou supérieures à 30 sont jugées suffisantes pour que le CLT soit maintenu, ce qui signifie que la distribution des moyennes d'échantillon est assez normalement distribuée. Par conséquent, plus on prend d'échantillons, plus les résultats graphiques prennent la forme d'une distribution normale.

La CLT est utilisé en finance pour examiner les rendements d'une action individuelle ou d'indices plus larges, en raison de la relative facilitée de génération des données financières nécessaires. Les investisseurs de tous types comptent sur la CLT pour analyser les rendements des actions, construire des portefeuilles et gérer les risques [17].

Voici ce que le théorème de limite centrale dit, graphiquement. L'image ci-dessous montre l'un des types les plus simples de test : rouler un dé juste. Plus vous roulez le dé, plus la forme de la distribution des moyens a tendance à ressembler à un graphique de distribution normal.

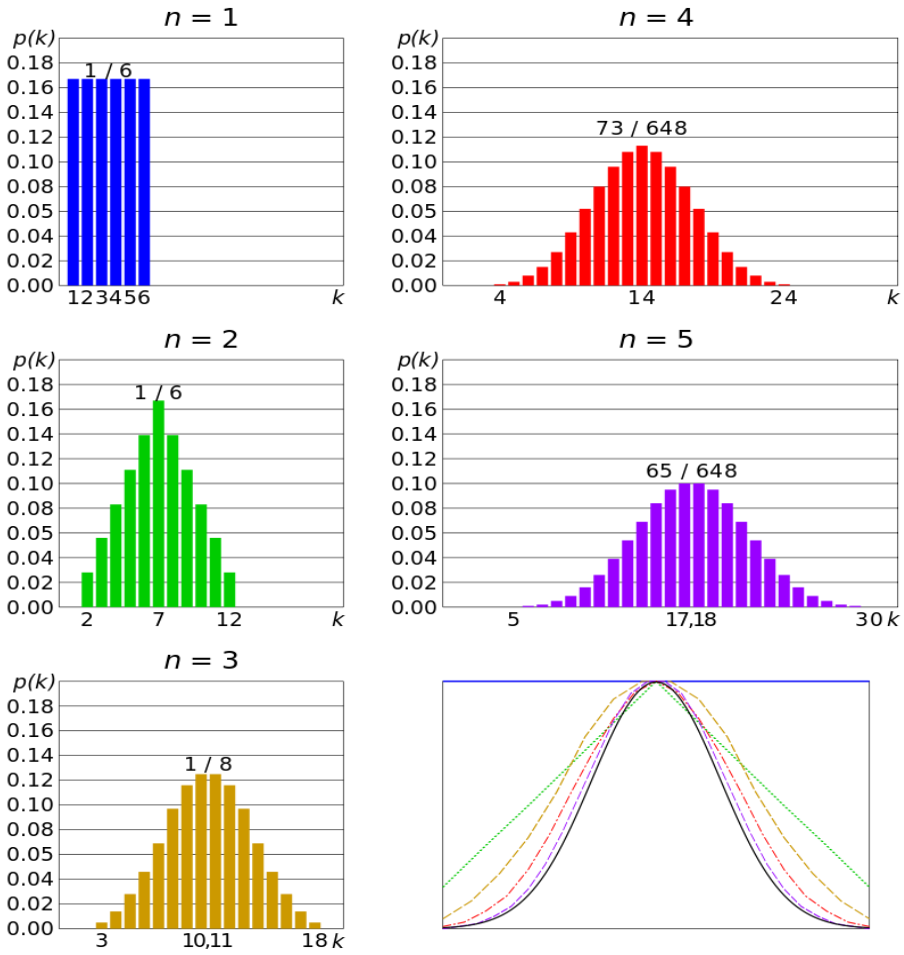

Figure 2.4. Test d'un dé juste.

Une composante essentielle du théorème de limite centrale est que la moyenne des moyennes de votre échantillon sera la moyenne de la population. En d'autres termes, additionnez les moyennes de tous vos échantillons, trouvez la moyenne et cette moyenne sera votre moyenne de population réelle. De même, si vous trouvez la moyenne de tous les écarts-types dans votre échantillon, vous trouverez l'écart-type réel pour votre population.

C'est un phénomène assez utile qui peut aider à prédire avec précision les caractéristiques d'une population [18].

En simulation, la situation typique est celle où on exécute un très grand nombre de fois une boucle, en calculant à chaque passage des réalisations de variables aléatoires indépendantes. Le résultat attendu est en général l'estimation d'une espérance. Pas plus en simulation qu'en physique ou en biologie on ne donnera un résultat sans indication sur sa précision. C'est le théorème central limite qui permet de calculer cette précision.

Soit $(X_N) \in N^*$ une suite de variables aléatoires variables aléatoires indépendantes de même loi, d'espérance μ et de variance σ^2 finies. Posons :

$$\forall n \in N^* \quad , \qquad \bar{X}_n = \frac{X_1 + \cdots .. + X_n}{n} \quad et \qquad Z_n = \frac{\sqrt{n}}{\sigma}(\bar{X}_n - \mu) \qquad (2.3)$$

Voici des captures qui montre comment la CLT converge une prédiction vers une loi gaussien [19] :

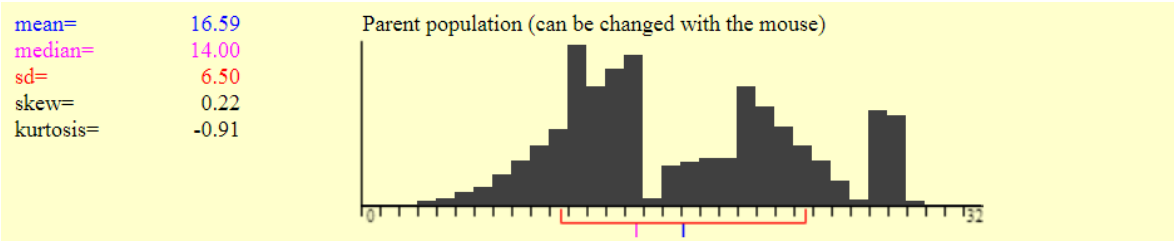

Figure 2.5. Capture d'une population aléatoire.

Pour 5 samples :

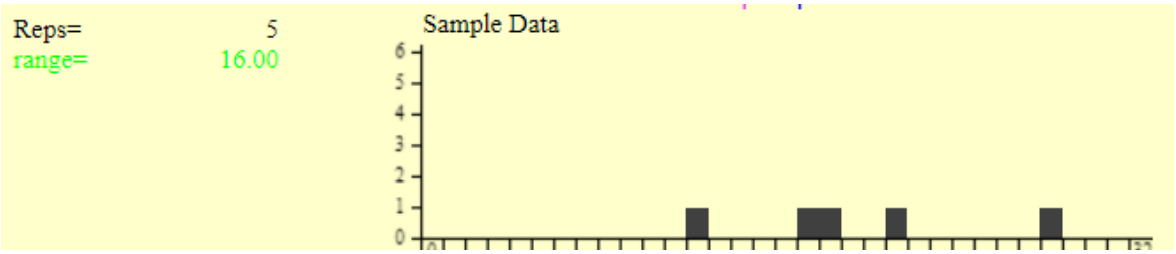

Figure 2.6. Pour 5 samples.

Pour 10 samples :

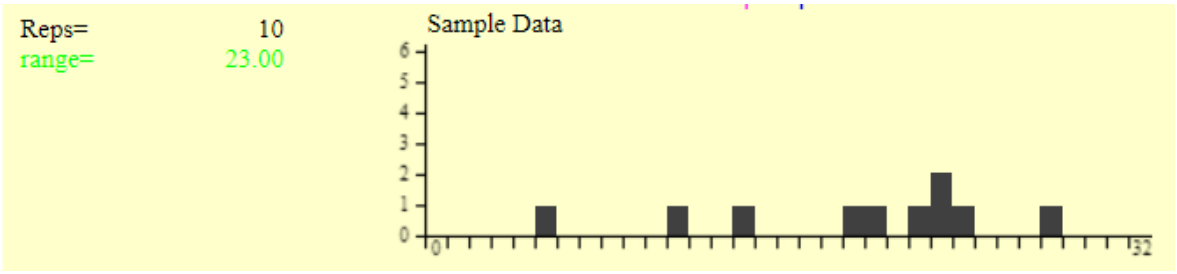

Figure 2.7. Pour 10 samples.

Pour 20 samples :

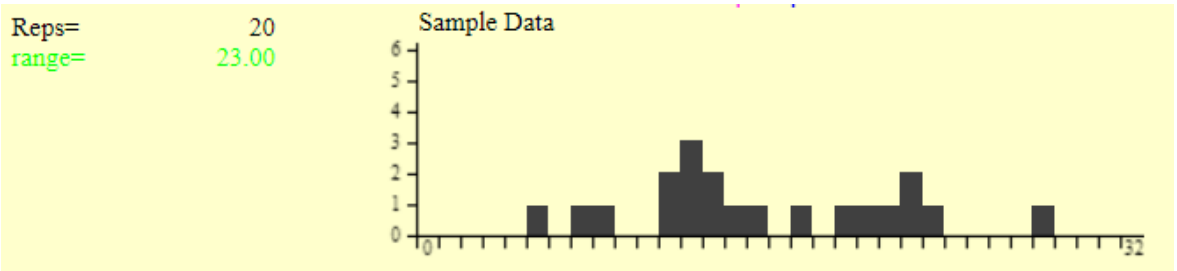

Figure 2.8. Pour 20 samples.

Avec 20 samples adding 10.000 times:

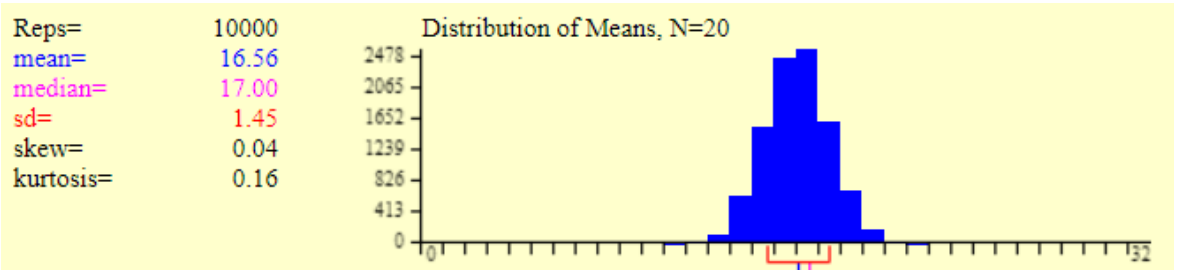

Figure 2.9. Avec 20 samples adding 10.000 times.

Avec 20 samples adding 100.000 times

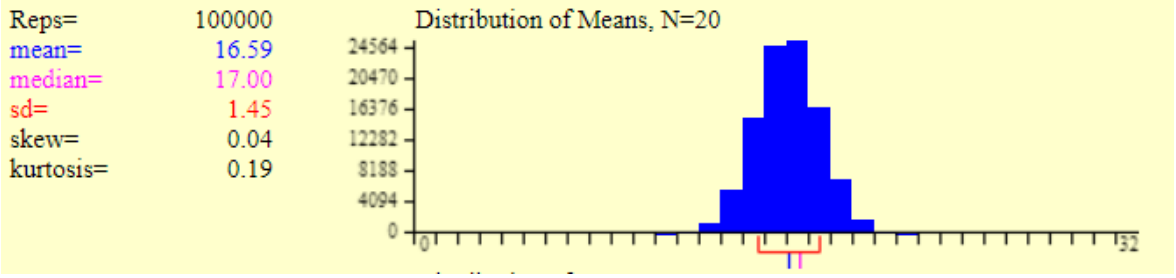

Figure 2.10. Avec 20 samples adding 100.000 times.

On voie que le résultat suit une loi normale.

2.4.1 Schéma synoptique

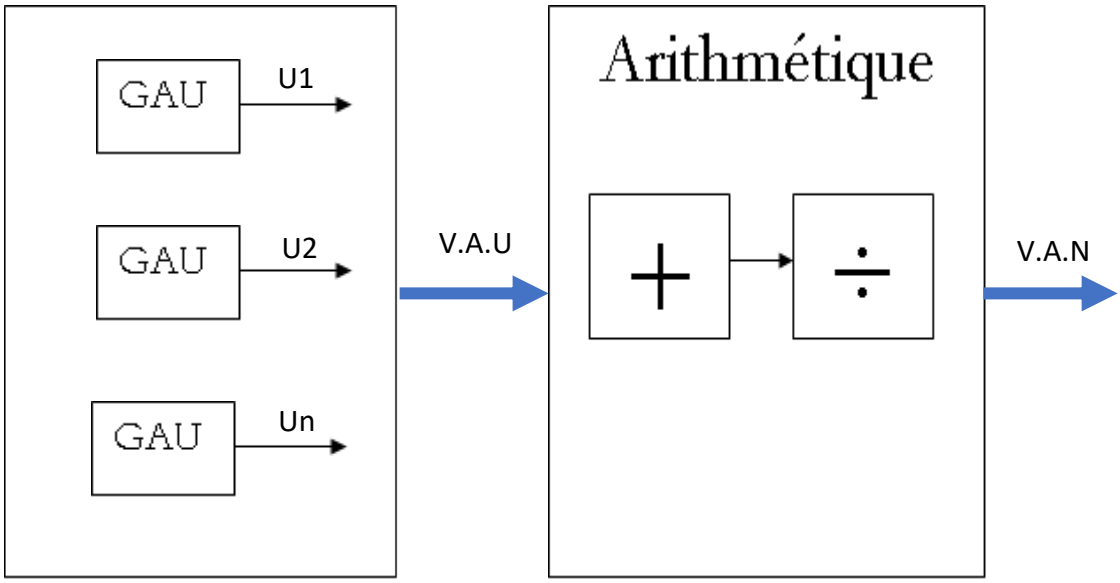

Figure 2.11. Schéma synoptique de La méthode de centrale limite.

Le schéma synoptique de la méthode centrale limite : est composé de plusieurs générateurs des variables aléatoires uniforme, et la conversion (arithmétique) des variables aléatoires uniforme vers des variable aléatoires normale est faites par des opérations (addition et la division).

2.5 Méthode de Box-Muller

La méthode de Box-Muller est l'une des méthodes de génération d'échantillons gaussiennes les plus utilisées. C'est une méthode de transformation exacte et capable de produire une paire d'un échantillon gaussien d'une paire de nombres aléatoires uniformes via une série de transformation. Le pseudo-code pour la mise en œuvre de cette méthode est décrit dans l'équation (2.4 et 2.5). Où U1 et U2 distribue sur l'intervalle (0,1) est la paire d'échantillons aléatoires uniformes de base et G1, G2 (comme indique dans l'équation) sont les deux générés Échantillons aléatoires gaussiens. On peut voir que la transformation implique quatre fonctions élémentaires, c'est-à-dire la fonction logarithmique, la racine carrée, le sinus et le cosinus, et à chaque itération lorsqu'elle est exécutée, deux échantillons gaussiens indépendants peuvent être généré [23].

$$\boldsymbol{G1} = \sqrt{-\mathbf{2}\boldsymbol{lnU1}} * \mathbf{cos}(\mathbf{2}\boldsymbol{\pi U2}) = \boldsymbol{f}(.) * \boldsymbol{g1}(.) \tag{2.4}$$

$$\boldsymbol{G2} = \sqrt{-\mathbf{2}\boldsymbol{lnU1}} * \ \mathbf{sin}(\mathbf{2}\boldsymbol{\pi U2}) = \boldsymbol{f}(.) * \boldsymbol{g2}(.) \tag{2.5}$$

Ou g(.) ∈ { $g1(.), g2(.)$ } sont des fonctions trigonométriques, il existe aussi la méthode polaire de [Marsaglia] qui sert à éviter ces fonctions trigonométriques les en transformé comme indiqué ci-dessous [16].

Étant donné x et y, une paire de variables indépendantes et uniformément distribuées, qui sont distribué sur $[-1, +1]$, soit s2 = u2 + v2. Alors, si s = 0 ou s<1, les équations (2.4) et (2.5) peut être écrit comme suit :

$$\boldsymbol{G1} = \boldsymbol{x} * \sqrt{\frac{-\mathbf{2}\ \mathbf{ln}\ \boldsymbol{s}}{\boldsymbol{s}}} \tag{2.5}$$

$$G1 = y * \sqrt{\frac{-2 \ln s}{s}} \tag{2.6}$$

2.5.1 Architecture matérielle pour la méthode Box-Muller

La figure 2.12 illustre la structure matérielle pour la méthode Box-Muller. Elle se compose de deux parties principales : les générateurs de nombres aléatoires uniformes (URNG) et l'élémentaire unités d'approximation des fonctions.

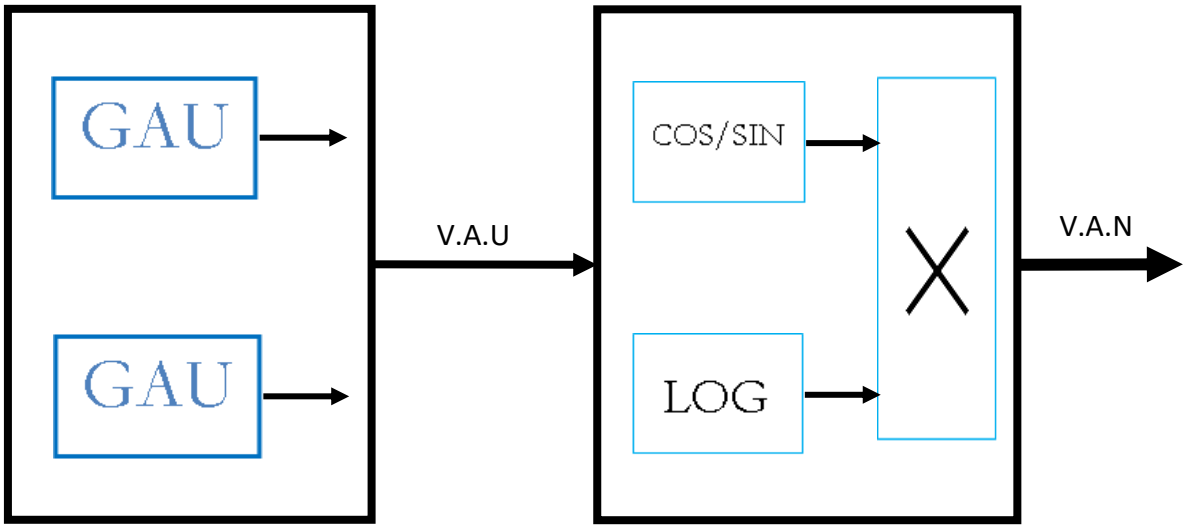

Figure 2.12. Schéma synoptique de La méthode box-Muller.

L'algorithme BM approprié pour l'implémentation du matériel car il n'exige pas de stockage paracerque il génère deux nouvelles variables par une conversion directe de deux variables d'entrée. Par conséquent, il n'est pas nécessaire de stocker les échantillons contrairement aux méthodes comme les méthodes CLT et Wallace. En plus, il n'y a pas de conditions if-else dans l'algorithme, cela garantit que les GRN sont disponibles à chaque

cycle d'horloge, en plus une computation idéal garantie l'analytique pour une distribution gaussienne l'idéal à la sortie [16].

Néanmoins, l'algorithme BM nécessite le calcul de fonctions élémentaires et trigonométrique et particulièrement dans les implémentations avec virgule fixe.

2.6 Méthode de récursion

Fournit des nombres répartis gaussiens en utilisant récursivement des combinaisons linéaires des GRN's disponibles.

2.6.1 Méthode de Wallace

Proposé par C.S. Wallace en 1996. La méthode Wallace est une nouvelle méthode qui permet de générer des échantillons gaussiens directement, sans utiliser d'abord un générateur uniforme, en évitant l'évaluation des fonctions élémentaires (exp, ln, sqrt, etc.) [16], en utilisant la récursion pour générer un ensemble de nouveaux GRN's à partir d'un ensemble existant. L'idée centrale est de prendre un pool de k nombres aléatoires déjà distribués en Gaussien, puis d'appliquer une transformation au pool, de sorte qu'un autre pool soit produit qui soit également gaussien. Après chaque transformation, le pool peut être généré pour fournir des nombres aléatoires. La principale exigence est de rendre la sortie de la transformation aussi décorrélée que possible de l'entrée, tout en préservant les propriétés de distribution [24].

L'algorithme Wallace est particulièrement adapté à la mise en œuvre matérielle à haut débit car aucune fonction transcendantale n'est requise [25].

Wallace propose un algorithme rapide pour générer des nombres pseudo-aléatoires normalement distribués qui génère les distributions cibles directement en utilisant leurs propriétés d'entropie maximale, cet algorithme est particulièrement approprié pour un débit élevé [25].

Le procédé transforme un vecteur X de longueur K pour un nouveau vecteur X1 tel que X1 = AX. Ici, A est une matrice orthogonale. Le nouveau X1 aura la même distribution que l'original X (gaussienne). L'algorithme maintient deux groupes de chiffres de la taille KL, appelée anciens et nouveaux. A chaque étape, les valeurs K de l'ancienne piscine sont transformées comme expliqué précédemment et stockées dans une nouvelle piscine. Après la transformation de tous les numéros dans l'ancienne piscine (passe), les rôles des deux piscines sont inversés [25].

La figure 2.13 décrit un aperçu de la méthode Wallace :

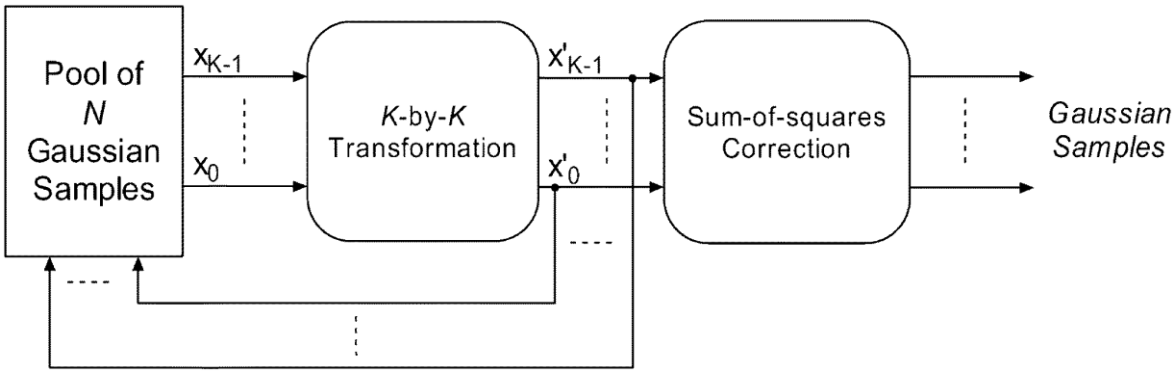

Figure 2.13. Aperçu sur la methode de Wallace.

Une des préoccupations de la méthode Wallace est la question des corrélations étant donné l'utilisation des extrants précédents pour générer de nouveaux extrants. Cela peut être problématique dans le cas de réalisations avec de grandes valeurs absolues se trouvant dans les queues (tails) du Gaussien [25].

Pour obtenir une meilleure décorrélation, Wallace suggère que R passe avant qu'un pool ne soit rendu disponible en sortie. En outre, au lieu de sélectionner X dans une piscine (pool) séquentiellement, ils sont sélectionnés aléatoirement pour fournir une décorrélation encore meilleure [16].

Voici le schéma synoptique de La méthode wallace :

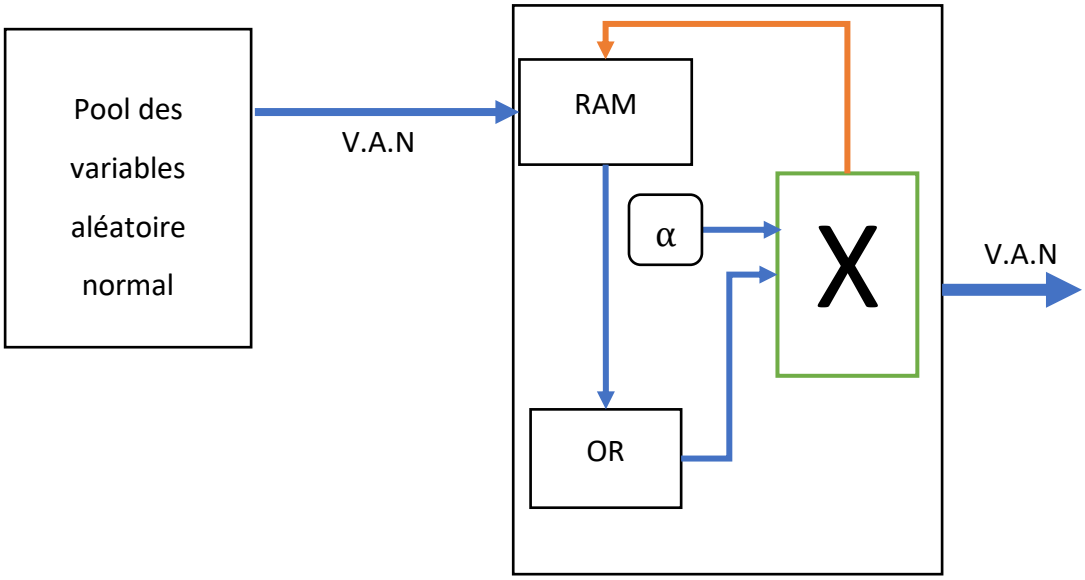

Figure 2.14. Schéma synoptique de La méthode wallace.

2.7 La méthode d'inversion

Méthodes d'inversion : Ces méthodes sont basées sur l'inversion de la densité cumulative Fonction (CDF) pour produire la distribution gaussienne il existe aussi une architecture qui combine les méthodes CLT et inversion.

2.7.1 The CDF Inversion Method

La CDF (x) décrit la probabilité que la valeur d'une variable soit inférieure ou égale à x. En termes de PDF, le CDF peut être décrit comme l'intégrale de PDF :

$$\Phi(\mathrm{x}) = \int_{-\infty}^{\mathrm{x}} \emptyset(\mathrm{x})\mathrm{dx} = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \qquad (2.7)$$

Ou Ø(x) est la Fonction de densité de Probabilité (PDF) de la distribution gaussienne standard d'une variable aléatoire x.

La fonction de distribution cumulative gaussienne inverse (IGCDF) est simplement l'inversion de $\Phi(\mathrm{x})$ dans l'équation (2.7). La figure 2.15 (b) montre l'IGCDF tracé par rapport à une variable x comprise entre zéro et un. La méthode basée sur l'inversion génère les GRN's en évaluant $\Phi^{-1}(\mathrm{x})$ par rapport à x, où 0<x<1. Cela peut être compris intuitivement à partir de la figure 2.15 (b). Tout comme x associe les URNs dans la plage de zéro à un, $\Phi^{-1}(\mathrm{x})$ est mappé aux GRN's approchant l'infini comme x atteint zéro ou un.

En termes d'implémentation, la méthode d'inversion est assez simple puisqu'il s'agit de l'évaluation d'une fonction unique ($\Phi^{-1}(\mathrm{x})$) et qu'elle est intrinsèquement sans mémoire, ce qui signifie qu'un seul URN est mappé à un seul GRN et ne nécessitant aucun stockage des échantillons. En outre, il n'y a pas de blocs (if-then-else) inhérents dans l'algorithme. Cela rend la méthode adaptée à l'implémentation matérielle car un GRN peut être généré chaque cycle d'horloge.

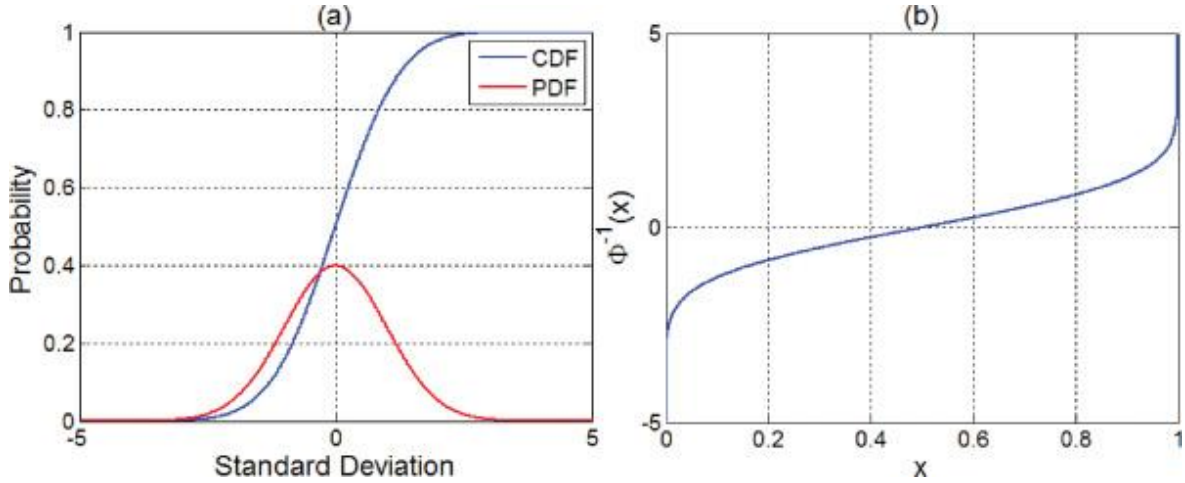

Figure 2.15. Un tracé du PDF et du CDF.

Mais cette méthode souffre de certains défauts. Premièrement, il n'existe pas de solution de forme fermée pour l'équation (2.7), ce qui implique qu'elle doit être calculée par approximation numérique. Cela conduit à des erreurs numériques lors de l'approximation, ce qui conduit à un compromis sur la qualité statistique des RNG générés.

Deuxièmement, pour obtenir une grande précision de queue, la largeur de bit de l'entrée URNG doit être maintenu à un niveau élevé, ce qui entraîne une complexité matérielle. En outre, l'approximation polynomiale avec des segments uniformes pour le calcul $\Phi^{-1}(\mathrm{x})$ conduit à des niveaux prohibitivement élevés besoins en mémoire. Un schéma de segmentation non uniforme diminuera la mémoire exigences mais fera à son tour de la génération d'adresses un problème non trivial.

Il existe de nombreuses méthodes d'évaluation des fonctions, telles que les méthodes d'addition de tables symétriques, CORDIC, l'approximation rationnelle, les méthodes polynomiales uniquement et les méthodes polynomiales de table+ [25].

a. Approximation Polynomial

L'approximation polynomiale est une clé de mise en œuvre des fonctions transcendantales f : R⟹ R, telles que exp, log, sin.

Les fonctions mathématiques transcendantales f : R⟹ R sont implémentées dans les systèmes informatiques principalement en utilisant des approximations polynomiales. Ces approximations polynomiales peuvent prendre la forme de tables de recherche, de multiplicateurs et d'ajouts lorsqu'il s'agit d'implémentation matérielle, ou peuvent simplement être codées. La raison est que, bien que l'ajout et la multiplication rapides et efficaces – les seules opérations nécessaires pour évaluer les polynômes – soient disponibles dans le matériel, les opérateurs de division – qui seraient nécessaires pour des approximations rationnelles – sont moins efficaces [26].

i. Segmentation Hierarchies de ICDF

La méthode de segmentation la plus couramment utilisée est le schéma uniforme, où toutes les longueurs de segment sont égales et le nombre de segments est généralement limité à des puissances de deux. La principale difficulté du GRN proposé est de rapprocher le ICDF d'une distribution donnée.

Nous appliquons la méthode de segmentation hiérarchique pour rapprocher efficacement les ICDF en fonction du comportement des distributions. HSM fournit quatre schémas de segmentation de base, désignés by $US, P2S_L, P2S_R$ et $P2S_{LR}$, respectivement. Aux US, les segments sont de taille uniforme. Dans $P2S_L$, les tailles de segment augmentent par des puissances de deux depuis le début de l'intervalle d'entrée jusqu'à la fin de l'intervalle, tandis que dans $P2S_R$ les tailles de segment diminuent par des puissances de deux du début à la fin de l'intervalle. Dans $P2S_{LR}$, les tailles de segments augmentent par des puissances de deux jusqu'au milieu de l'intervalle, puis diminuent par des puissances de deux jusqu'à ce que la fin soit atteinte.

Cette méthode est hiérarchique car la segmentation peut être appliquée récursivement : dans la première passe, l'intervalle entier est subdivisé en utilisant l'un des quatre schémas précédents en segments plus petits, puis dans la seconde passe, chaque

segment peut être subdivisé davantage, encore une fois en utilisant l'un des quatre régimes. Au cours du deuxième passage pour le cadre de ce document, la segmentation est fixée aux US [27].

2.8 Conclusion

Dans ce chapitre nous avons évoqué le sujet des générateurs de variable aléatoire gaussien et leurs classifications, on a vu que chaque méthode défère de l'autre par ses avantages et inconvénients et par sa mise en œuvre matérielle, car la qualité de ces nombres aléatoires joue un rôle central pour s'assurer que les résultats de la simulation sont significatifs.

Chapitre 3 : Synthèse d’un générateur de nombres aléatoires gaussiens basé sur la méthode d’inversion.

3.1 Introduction

La méthode d'inversion est très efficace pour générer des variables aléatoires gaussiens mais cette méthode demande un nombre de bits élevés ce qui rend cette méthode difficile à être implémenter. Alors notre méthode sert à réduire le nombre des bites d'entrer et les fixé cependant avoir une haute précision de la queue.

Notre outil de calcul doit produire les vecteurs à virgule flottante pour la fonction d'ICDF, qui vont être transformé en virgule fixe, et doit avoir la possibilité d'analyse des résultats. Il faut noter que la transformation en virgule fixe de résultats de l'IGCDF sert aussi dans la création des fichiers VHDL ROM pour implémentation sur FPGA.

Le MatLab et le System Generator de Xilinx sont des outils incontournables pour notre travail.

3.2 Principe de la méthode d'icdf

La fonction de distribution cumulative gaussienne inverse (IGCDF) est simplement l'inversion de $\Phi(x)$, qui décrit la probabilité d'une variable soit inférieure ou égale à x. Le CDF peut être décrit comme l'intégrale de PDF.

La ICDF consiste à générer des variables aléatoires par rapport à une variable x comprise entre zéro et un $0 < x < 1$. En utilisant la fonction ICDF de MatLab.

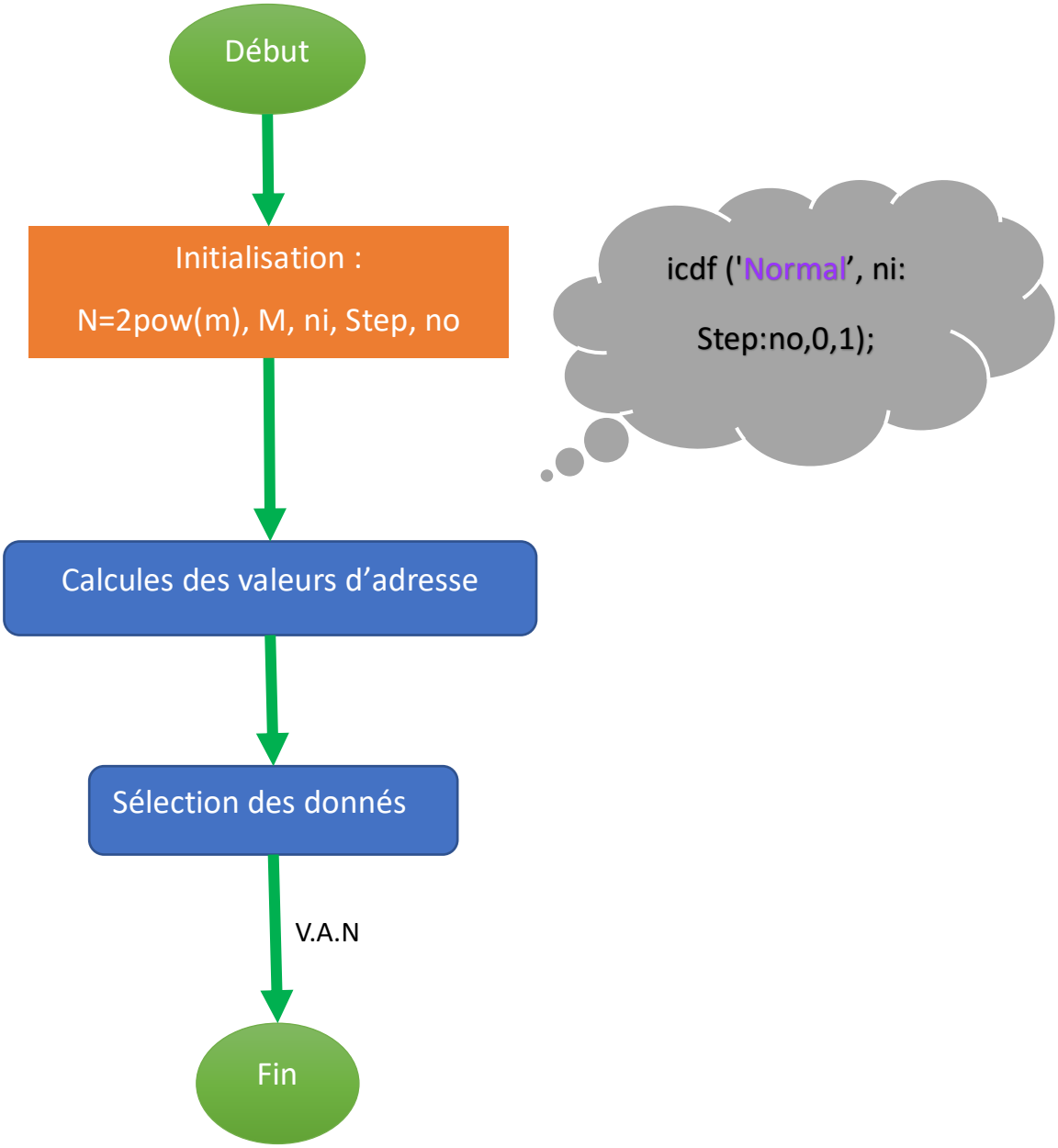

Figure 3.1. Organigramme de la méthode ICDF.

La figure 3.1 illustre l'organigramme de la méthode ICDF

Au premier pas, nous avons déclaré les entres (inputs) :

- m : c'est le nombre de bits d'adresse.

  Ce nombre est un indice pour la calcule de :

  - N=pow2(m) : représente le nombre de pas de quantification
  - ni= 1/(2*N) : c'est le début
  - no= 1-1/(2*N) : la fin du calcule
  - step= 1/N : le pas

En fin en utilise la fonction icdf pour calculer les variables aléatoires :

- va =icdf('normal', ni : step :no,0,1)

- normal : indique que la fonction d'icdf est de base d'une distribution normal.

- 0 et 1 :mu et sigma respectivement représente les paramètres de la distribution normal

3.3 Principe de la méthode proposé

Notre méthode proposée consiste à ajouter une plage à la fin de la première plage mais cette plage est considérée comme un pas diviser sur le nombre de quantification. Cette méthode résulte l'obtention d'une précision de queue (tail) plus que de la méthode ICDF, toute en fixant le nombre de bits des valeurs d'adresse.

3.4 Fonctionnement logicielle de la méthode proposée

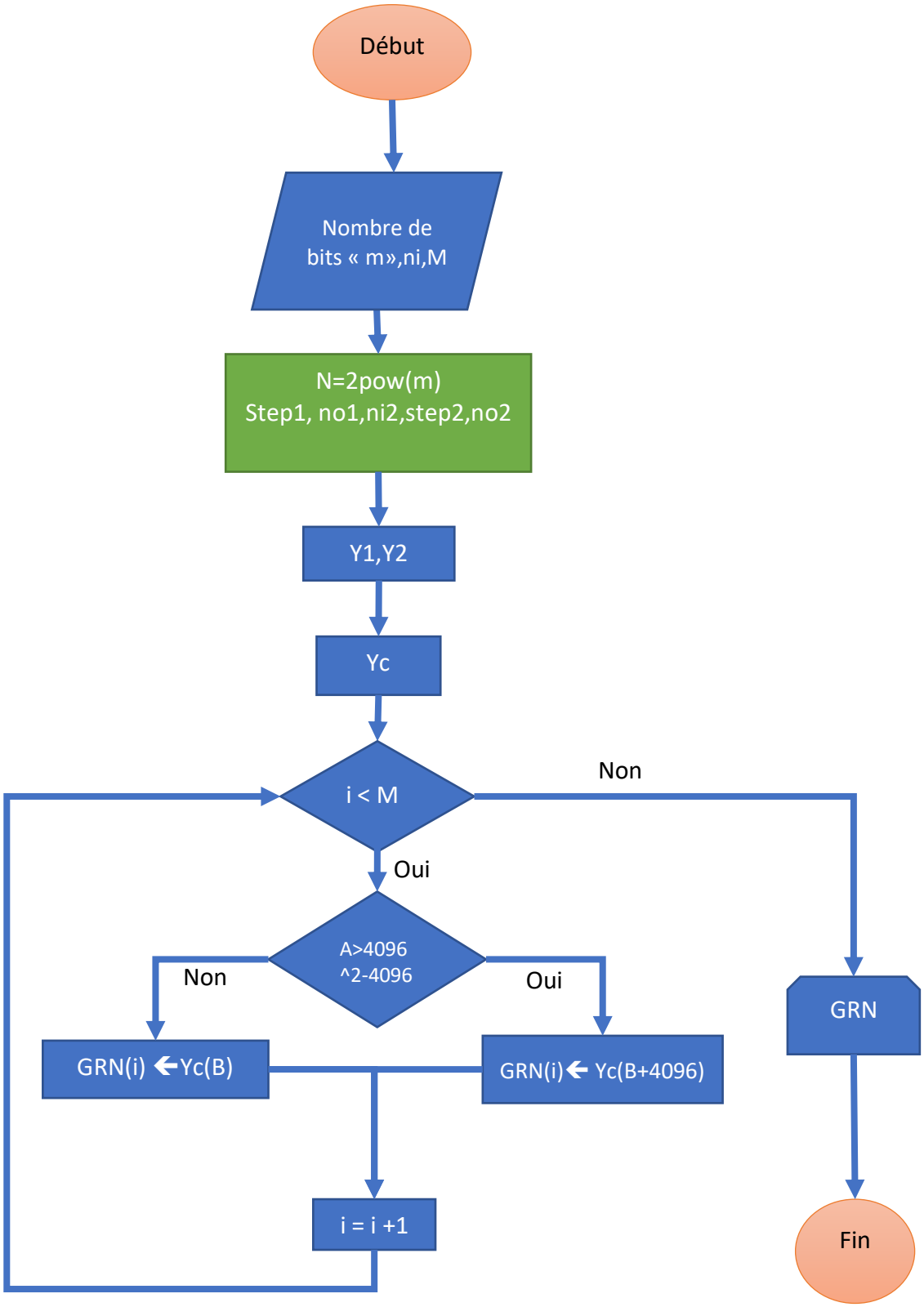

Figure 3.2. Organigramme de la méthode ICDF proposé.

La figure 3.2 montre l'organigramme de la méthode ICDF proposé

Les entrés déclaré sont :

m : c'est le nombre de bits d'adresse.

ni : égale à 0.5 représente la valeur initiale.

M=10 Million, est la quantité d'échantillon pour la variable aléatoire gaussien.

Après ces déclarations, les variables suivants sont calculées automatiquement :

N, step1, step2, ni2, no2, no1

Nous supposons que Y1 et Y2 sont deux matrices qui sont charger par le résultat de calcul de la fonction icdf. Avec Yc une matrice qui concaténé les deux matrice Y1 et Y2.

Après avoir établie nos Constans et calculer nos variables maintenant on passe ou calcules de la variable aléatoire gaussien.

Première étape :

Création d'une boucle « for » pour M itération.

Deuxième étape :

Puis le calcul de la fonction « randi » qui consiste à donner un entiers pseudo-aléatoires d'une distribution discrète uniforme entre [1, 16777216].

Mettre une condition pour déterminer à quelle plage nous allons remplir notre variable noté GRN.

3.5 Fonctionnement matérielle (FPGA) de la méthode proposée

L'implémentation de notre travail de la méthode proposé exige l'utilisation de logiciel ISE qui consiste la description matérielle VHDL.

Voici un schéma synoptique qui illustre notre travail :

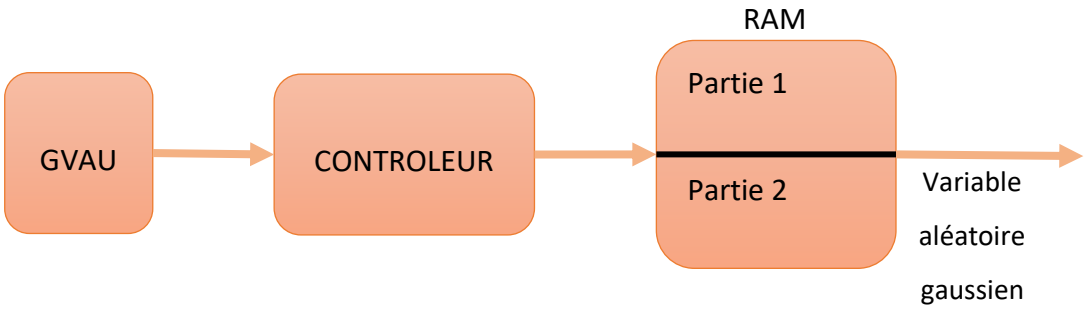

Figure 3.3. Schéma synoptique du fonctionnement matériel.

La mise en œuvre matérielle sert à utiliser un générateur de variable aléatoire uniforme sous forme LFSR.

Il existe plusieurs types pour générer un VAU comme TG (Tausworthe Generator) et LFSR (linear feedback shift register) GFSR, twisted GFSR, Mersenne twister, WELL, xorshift, polynomial LCG, etc.

3.5.1 LFSR

Un registre à décalage à rétroaction linéaire, ou LFSR (acronyme de l'anglais linear feedback shift register) est une suite récurrente linénaire, elle s'implémente électroniquement très simplement puisqu'il s'agit alors de faire un XOR ou XNOR avec certains bits [28].

Le LFSR constitue l'élément de base des générateurs pseudo-aléatoires utilisés pour la génération de la suite chiffrant.

a. Fonctionnement

Un registre à décalage à rétroaction linéaire prend une fonction linéaire, typiquement un OU exclusif, comme entrée. Un LSFR, comme les autres registres à décalage, est une cascade de circuits à bascule. Les bits qui changent d'état pour les autres de la cascade sont appelés taps. Deux des principaux systèmes de raccordement des robinets sont Fibonacci et Galois. Dans la configuration de Fibonacci, les prises sont montées en cascade et introduites dans le bit le plus à gauche. Dans une configuration Galois, nommée d'après le mathématicien français Évariste Galois, chaque tap est XOR au flux de sortie. Voire la figure ci-dessus

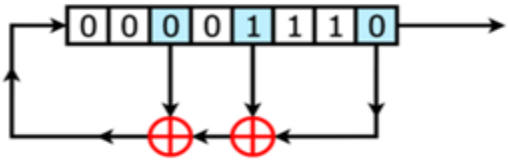

Figure 3.5: LFSR de Fibonacci à 8 bits.

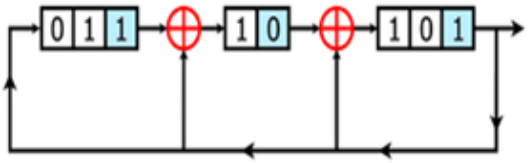

Figure 3.4 : LFSR Galois à 8 bits.

Le tableau suivant présente un exemple de LFSR de 8 bits à chaque top d'horloge :

Top d'horloge	Etat de LFSR	Output
t=0	11001101	1
t=1	11100110	1
t=2	11110011	0
t=3	01111001	0
t=4	00111100	1

Tableau 3.1. Résultat D'un LFSR de 8 bits à des instant t.

Les LSFR sont utilisés en cryptographie pour la génération de nombres pseudo-aléatoires, les séquences de pseudo-bruit. Ils sont également souvent utilisés pour les compteurs numériques car ils sont si rapides [29].

b. Architecture

Le registre à décalage à rétroaction linéaire est mis en œuvre sous la forme d'une série de bascules qui sont câblées ensemble. Plusieurs prises de la chaîne du registre à décalage sont utilisées comme entrées dans une porte XOR ou XNOR. La sortie de cette porte sert alors de retour au début de la chaîne du registre à décalage, d'où le retour dans LFSR. Voir le figure (3.6)

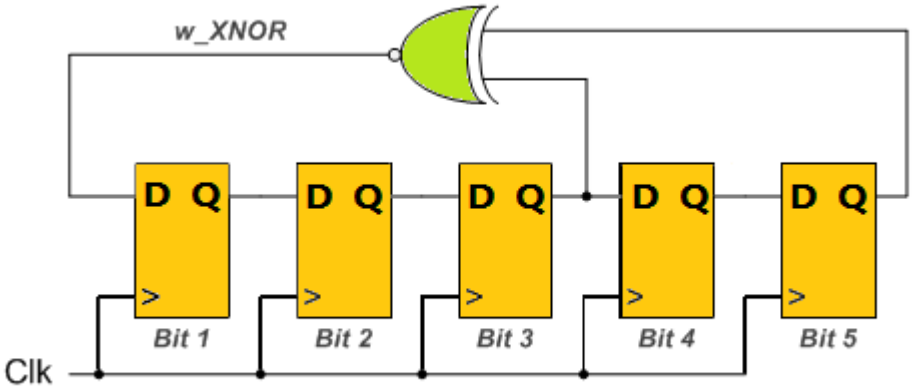

Figure 3.6. Architecture LFSR 5 bits.

3.5.2 UNITE DE MEMOIRE (ROM)

Notre implémentions matérielle demande l'utilisation d'une RAM qui contient des vecteurs en virgule fixe.

a. Conversion des vecteurs avec virgule flottante En virgule fixe

Dans notre projet nous utilisons les virgules fixe, on considère notre vecteur a virgule flottante est : Yc

Dans notre programme le vecteur Yc est créé par une boucle « for » avec « i » itération : i < M. Si i=M la boucle s'arrête.

Après M itération, la création de vecteur Yc est accomplie.

Ensuite le vecteur Yc va être multiplier par 2^n :

$$Yn = Yc * 2^n$$

La déférence entre Yc et Yn le vecteur Yc c'est un vecteur en virgule flottant, et Yn c'est un vecteur en virgule fixe.

Voici un organigramme qui présent la procédure de binarisation :

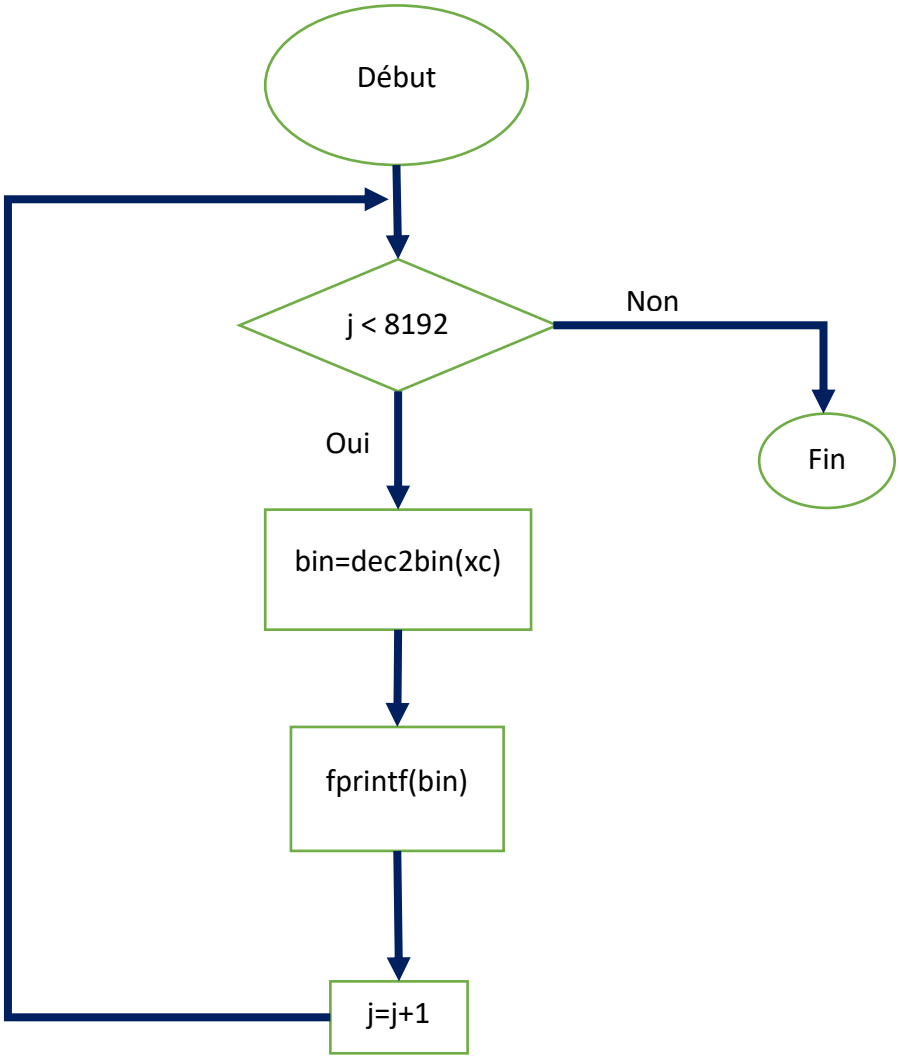

Figure 3.7. Organigramme de binarisation/création fichier ROM.

3.5.3 Partie 1 : Binarisation

L'étape qui la suit normalisation est la binarisation des valeurs de Yn pour que on les stocke dans un fichier texte qui peut considérer comme une ROM.

En utilisant une fonction précise dans une boucle « for » avec j itération définie par la taille d'unité de mémoire :

$$Ybin = dec2bin(Yn)$$

Avec Ybin est la valeur binaire de vecteur Yn en virgule fixe.

3.5.4 Partie 2 : création du fichier ROM

Le fichier ROM demande une simple instruction pour être créé, ensuite il faut utiliser le vecteur Yn en virgule fixe pour charger cette unité de mémoire. En utilisant ces étapes suivantes dans Matlab nous favorise à compléter notre objectif :

- Création d'un fichier texte nommé RAM1.
- Création de boucle « for » avec j itérations.
- Stocké les valeurs obtenues dans RAM1.

Voici la figure qui représente le fichier texte :

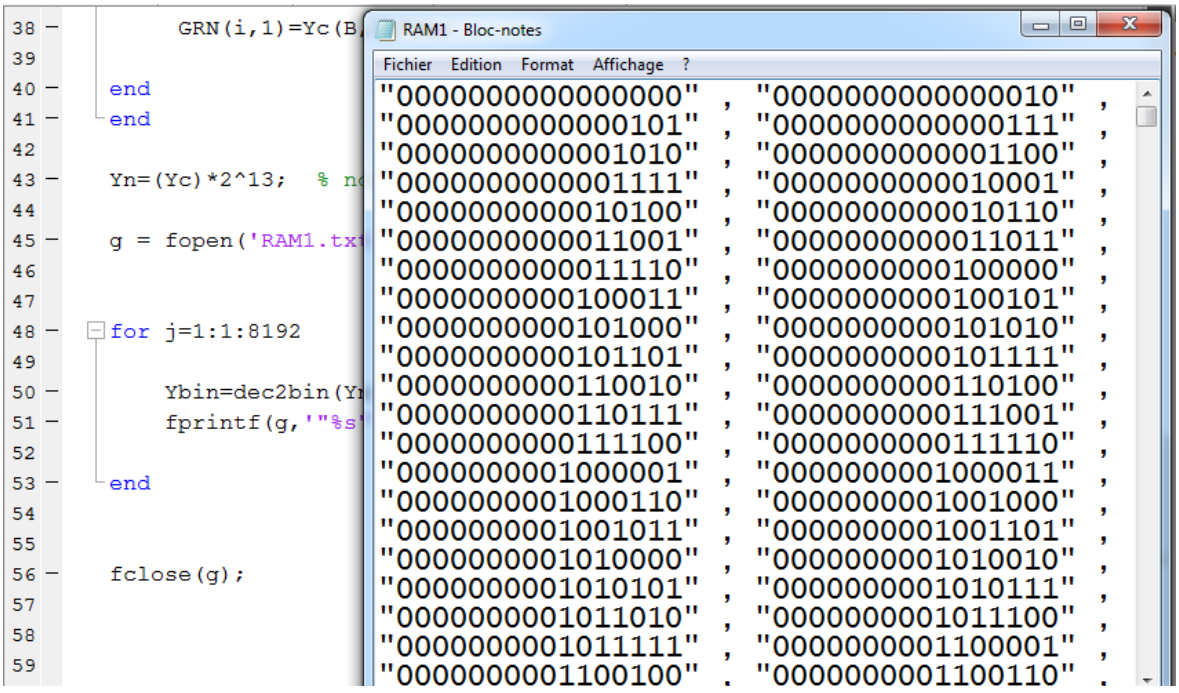

Figure 3.8. Fichier RAM1 avec les valeurs binarisé.

Le contenu de ce fichier texte va être utiliser dans la description VHDL/ROM pour la partie implémentation sur FPGA.

3.5.5 CONTROLEUR

L'adaptation de notre programme MatLab en VHDL veut dire synthétiser notre code, pour cela nous utilisons des descriptions de composant VHDL qui ressemble à faire la même fonctionnalité du code. Dans notre mise en œuvre matérielle on a choisi d'impliquer une seule ROM, ce dernier nous a imposer d'utiliser un comparateur.

a. Fonctionnalité

La fonctionnalité principale de notre comparateur est d'où choisir les valeurs, car notre RAM est divisée en 2 parties. Soit les valeurs vont être lise de partie supérieure ou inferieur.

Après que le comparateur fait son travail nous résultants des variables aléatoires gaussiens issue de la RAM.

3.6 Conclusion

Dans ce chapitre nous avons présenté notre travail du coté logiciel ou on a exprimé les déférente étapes de notre méthode proposée qui résulte à générer les variables aléatoires gaussiens, cependant la mise en œuvre matérielle est aussi impliquée d'une manière que on a défini le rôle et la fonctionnalité des composants VHDL.

Chapitre 4: Implémentation et résultats

4.1 Introduction

L'optimisation matérielle est la dernière étape de notre travail en utilisant le logiciel de Xilinx appelé ISE, alors nous allons utiliser un langage de description matérielle VHDL qui sert à synthétiser l'architecture globale de notre générateur de bruit Gaussien.

L'exploitation de System Generator (SG) nous aide à faire la simulation, l'affirmation et la validation du travail mise en VHDL.

4.2 Implémentation et test sur MatLab

Notre programme de générateur de variable aléatoire gaussien basée sur l'icdf doit passer par des tests pour valider et vérifier l'exactitude de l'architecture. Et pour cela nous avons appliqué des tests standards :

4.2.1 Teste de densité spectral

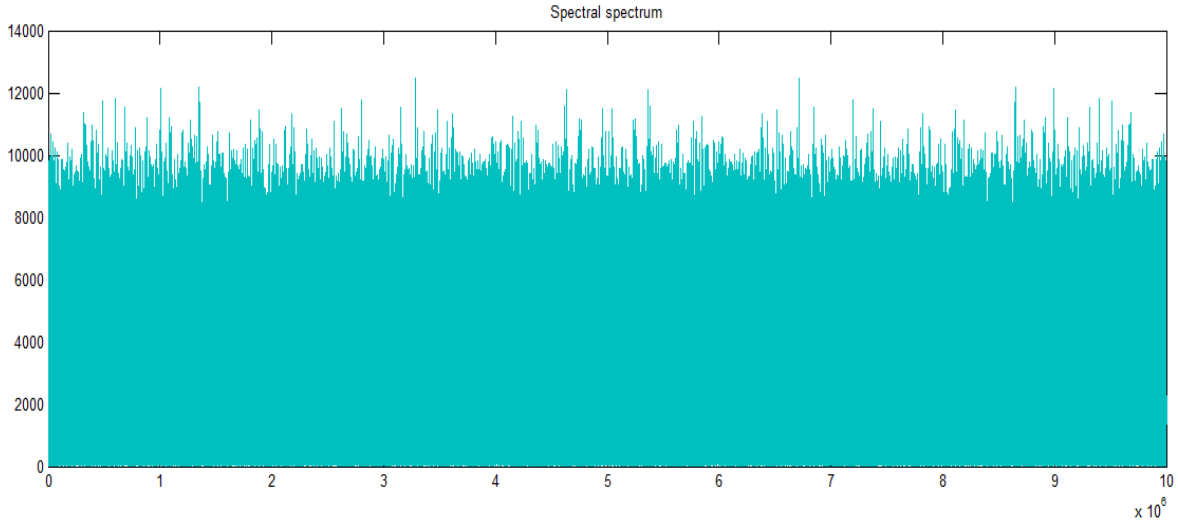

Figure 4.1. Teste spectral.

Interprétation : Le tracé spectral est un spectre uniformément distribué, la partie gauche est symétrique à la partie droite. Ce tracé obtenu est une propriété souhaitable importante de toute séquence aléatoire.

4.2.2 Test d'autocorrélation

Cette figure montre la corrélation entre les nombres successifs, la fonction d'autocorrélation montre des corrélations extrêmement faibles pour tous les décalages différents de zéro sur une plage de ± 10^7.

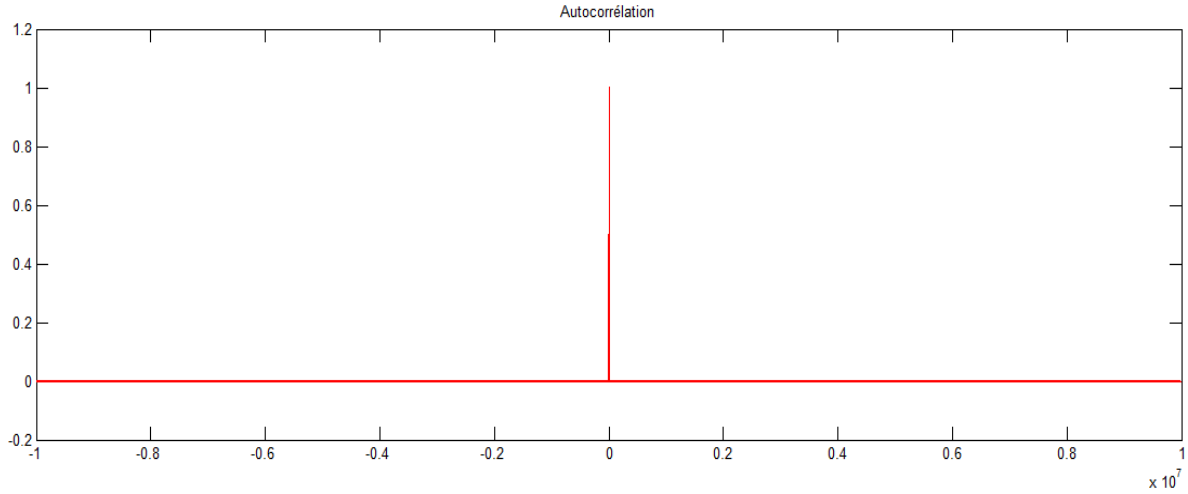

Figure 4.2. Teste d'autocorrélation.

4.2.3 Histogramme de courbe ICDF logiciel

Résultats de la partie logicielle nous donne cet histogramme adjacent avec la courbe de gausse idéal (TH), la simulation est faite avec 10 Million échantillons.

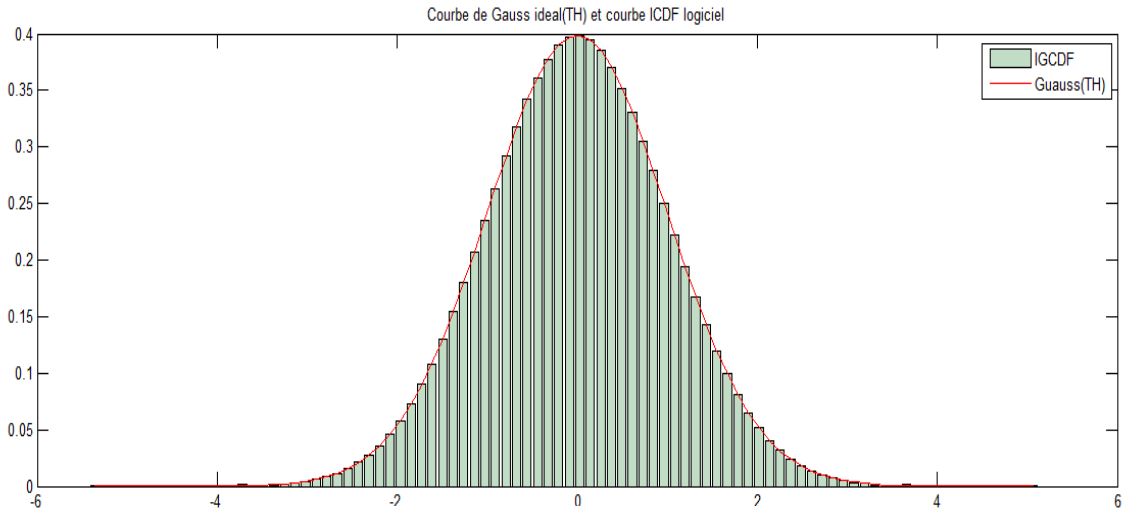

Figure 4.3. Courbe ICDF issue de code MatLab.

4.3 Implémentation sur FPGA

Comme décrit le schéma synoptique de la figure on utilise les descriptions VHDL suivantes :

4.3.1 LFSR

On a utilisé LFSR 52 bits

```
entity LFSR52 is

   Port (C : in  std_logic;

         CE : in  std_logic;

          SEED : in  std_logic;

        u1 : out  std_logic_vector (23 downto 0);

        sgn : out  std_logic);

end LFSR52;

architecture Behavioral of LFSR52 is

   signal tmp: std_logic_vector(51 downto 0);

   signal SI: std_logic;

begin

process (C)
```

```
begin

  if (C'event and C='1') then

        if SEED ='1' then

        tmp(9 downto 0) <= (others =>'0');

        tmp(20 downto 10) <= (others =>'1');

        tmp(29 downto 20) <= (others =>'0');

        tmp(39 downto 30) <= (others =>'0');

        tmp(51 downto 40) <= (others =>'1');

        else

        if CE='1' then

    for i in 0 to 50 loop

      tmp(i+1) <= tmp(i);

    end loop;

    tmp(0) <= SI;

  end if;

        end if;

        end if;
```

```
    end process;

    SI <= tmp(51) xor tmp(50) xor tmp(48) xor tmp(45);

     u1 <= tmp(33 downto 10);

    sgn <= tmp(44);

     end Behavioral;
```

On a choisi un LFSR de 52 bits de type XOR, la sortie de LFSR est choisie par les du 10 à 33, et la bits signe choisi du bit numéro 44.

Schéma RTL du LFSR à 52 bits :

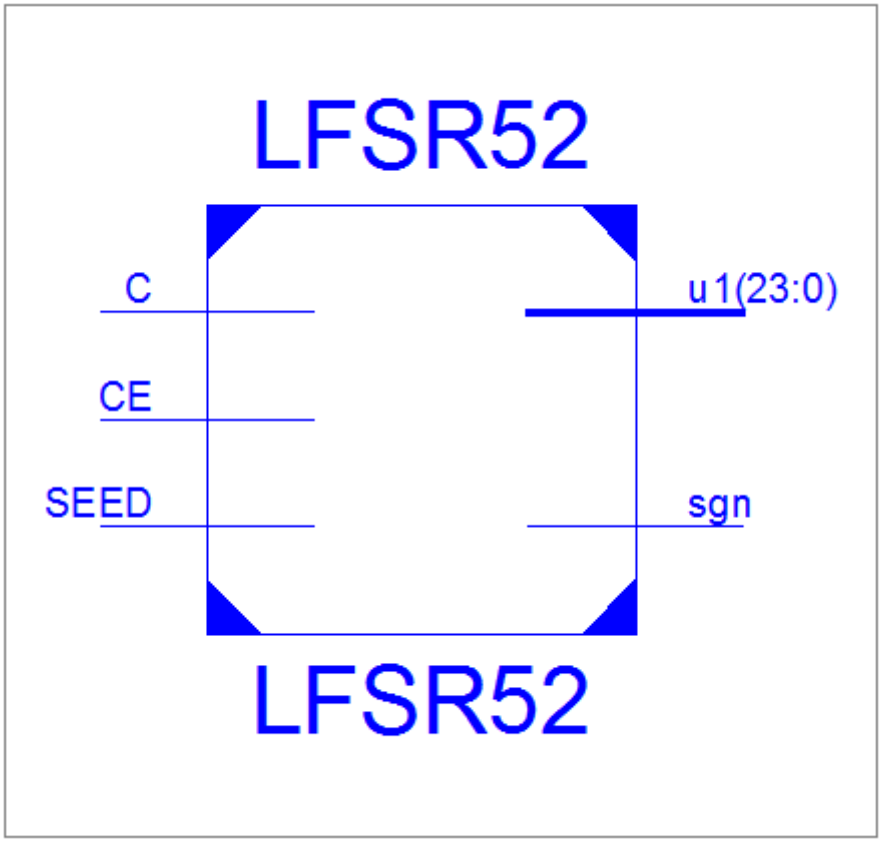

Figure 4.4. RTL Schematic d'un LFSR 52 Bits.

4.3.2 Unité de mémoire

L'unité de mémoire utilisé sous forme RAM contient les valeurs en binaire.

Description VHDL d'une RAM:

```
entity RAM1 is

port (C : in std_logic;

CE : in std_logic;

    EN : in std_logic;

    ADDR : in std_logic_vector(12 downto 0);

    DATA : out std_logic_vector(15 downto 0));

end RAM1 ;

architecture Behavioral of RAM1 is

type rom_type is array (8191 downto 0) of std_logic_vector (15 downto 0);

  signal ROM : rom_type:= ( "0000000000000000" , "0000000000000010" , … ,
"1010100101101110" , "1010110101110000" );

signal rdata : std_logic_vector(15 downto 0);

begin

  rdata <= ROM(conv_integer(8191-ADDR));

  process (C)

  begin
```

```
        if (C'event and C = '1') then
                if (CE = '1') then
          if (EN = '1') then
             DATA <= rdata;
          end if;
        end if;
                end if;
                 end process;
end Behavioral;
```

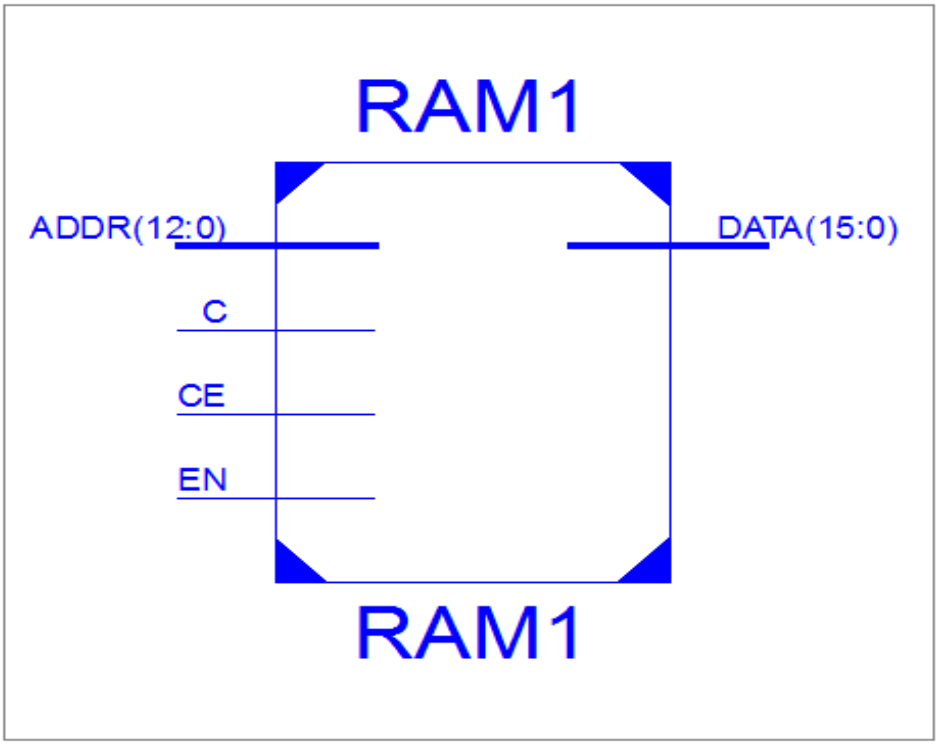

Figure 4.5. RTL Schematic de la RAM.

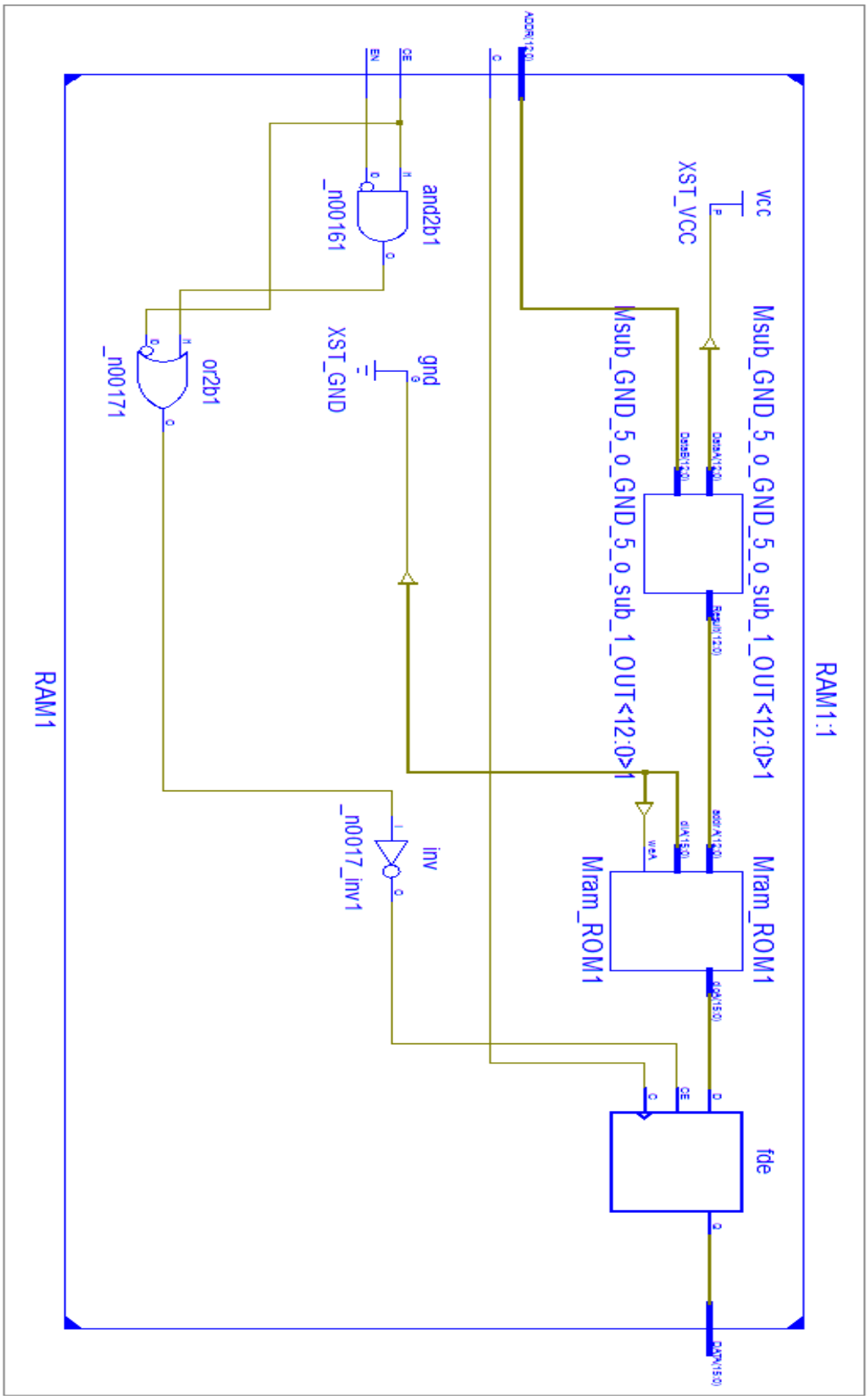

Figure 4.6. RTL Schematic de la RAM.

4.3.3 Contrôleur

Ce contrôleur a un rôle important dans l'utilisation d'une seul RAM, ce ne sera pas le cas si 2 RAM ont été employé.

```
entity CRT_RAM1 is

   Port ( in_crt : in  std_logic_vector (23 downto 0);

          out_crt : out  std_logic_vector (12 downto 0));

end CRT_RAM1;

architecture Behavioral of CRT_RAM1 is

begin

process(in_crt)

begin

if (in_crt > 16773119) then

out_crt <= in_crt(12 downto 0);

else

out_crt(11 downto 0) <= in_crt(21 downto 10);

out_crt(12) <= '0';

end if;
```

```
    end process;


    end Behavioral ;
```

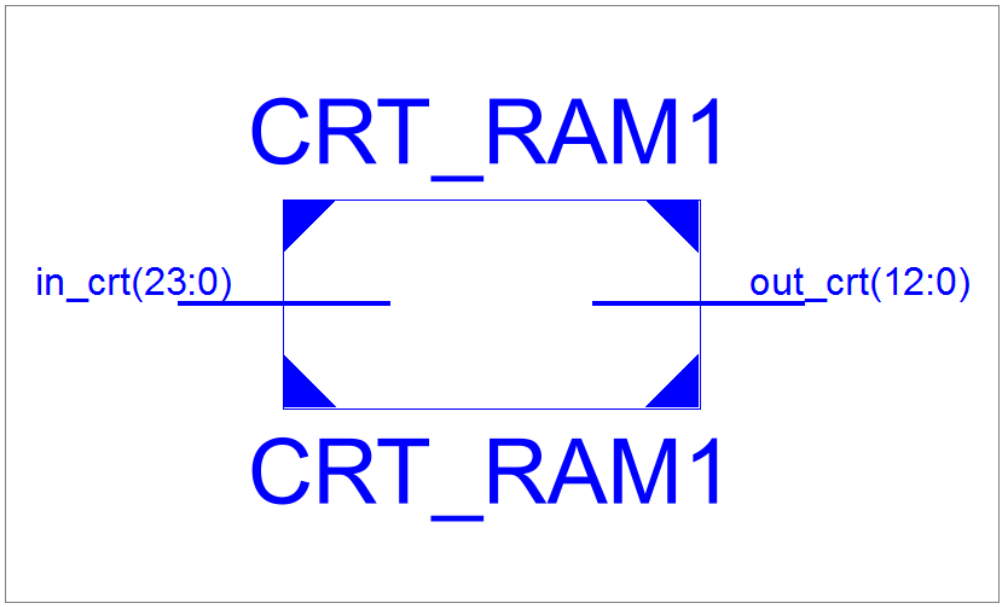

Figure 4.7. RTL Schematic de notre contrôleur.

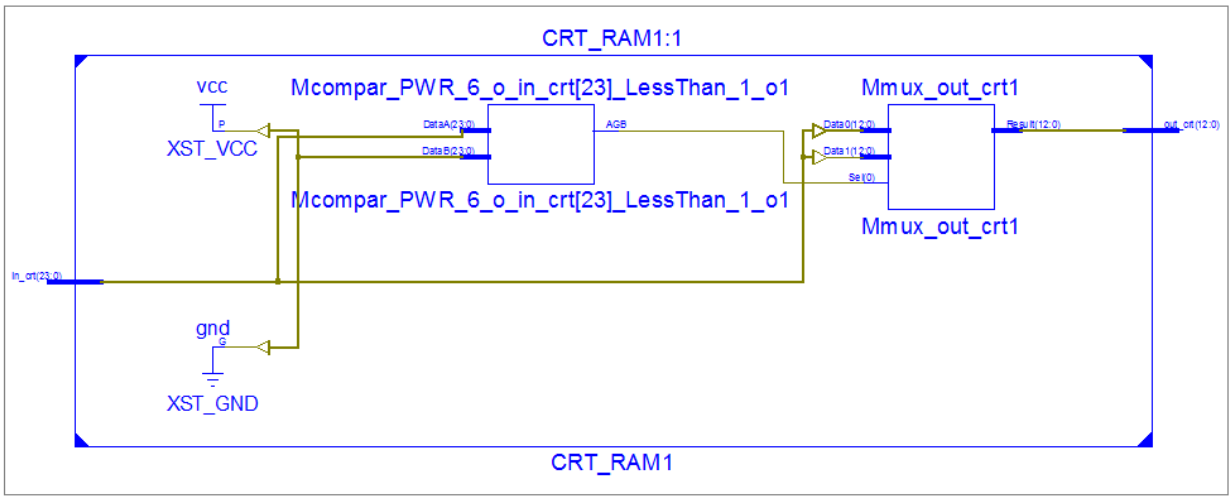

Figure 4.8. RTL Schematic de notre contrôleur.

4.4 Simulations et résultats

L'étape finale du travail implique l'utilisation de Simulink plus spécifiquement le System Generator qui est un toolbox développé par Xilinx qui est intégré dans l'environnement Matlab Simulink et qui laisse l'usager créer des systèmes hautement parallèles pour FPGA. Les modèles créés sont affichés sous forme de blocs, et peuvent être raccordés aux autres blocs et autres toolbox de Matlab-Simulink [32].

4.4.1 Architecture de générateur de bruit gaussien avec SG

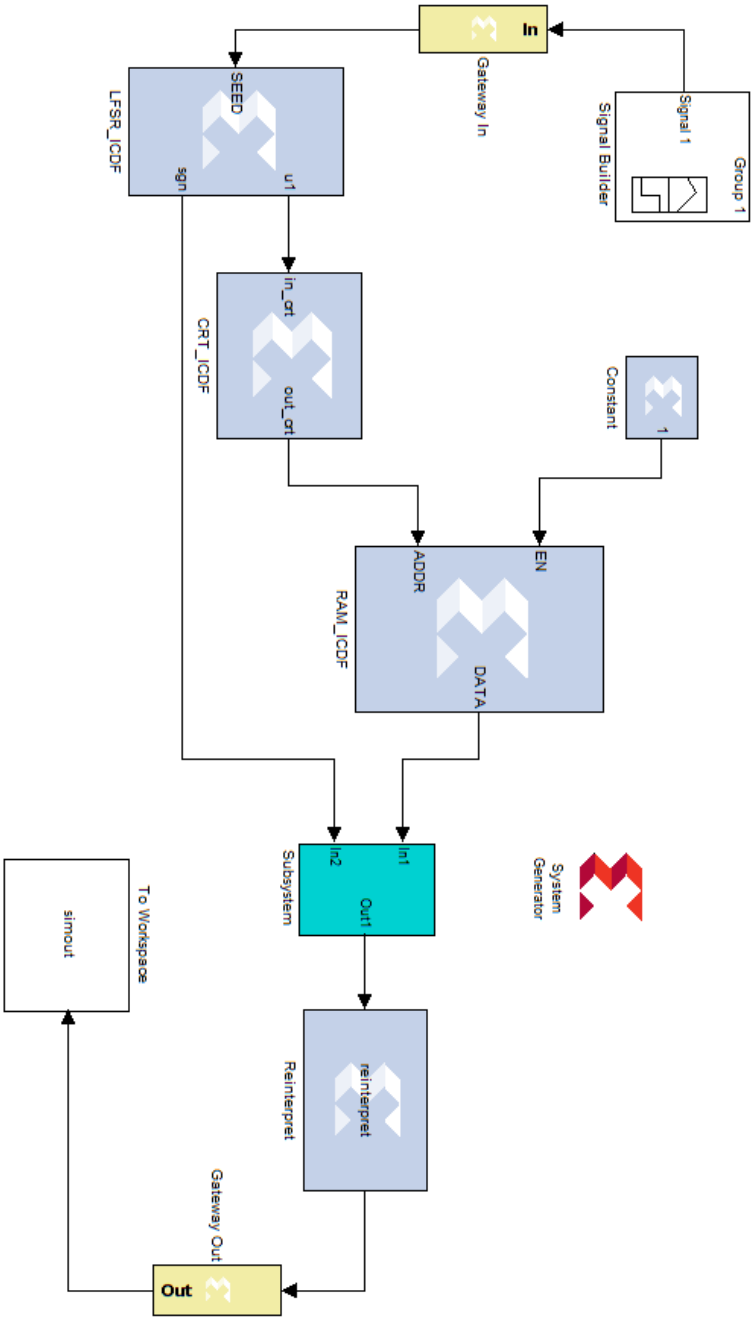

Figure 4.9. Architecture de générateur bruit gaussien avec SG.

Tous les modelés Simulink contenant les blocs de Xilinx doivent contenir au moins un bloc SG :

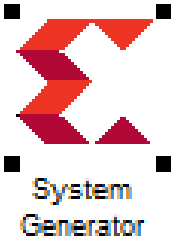

Figure 4.11. Bloc system Generator dans Simulink.

a. Configuration de bloc system generator

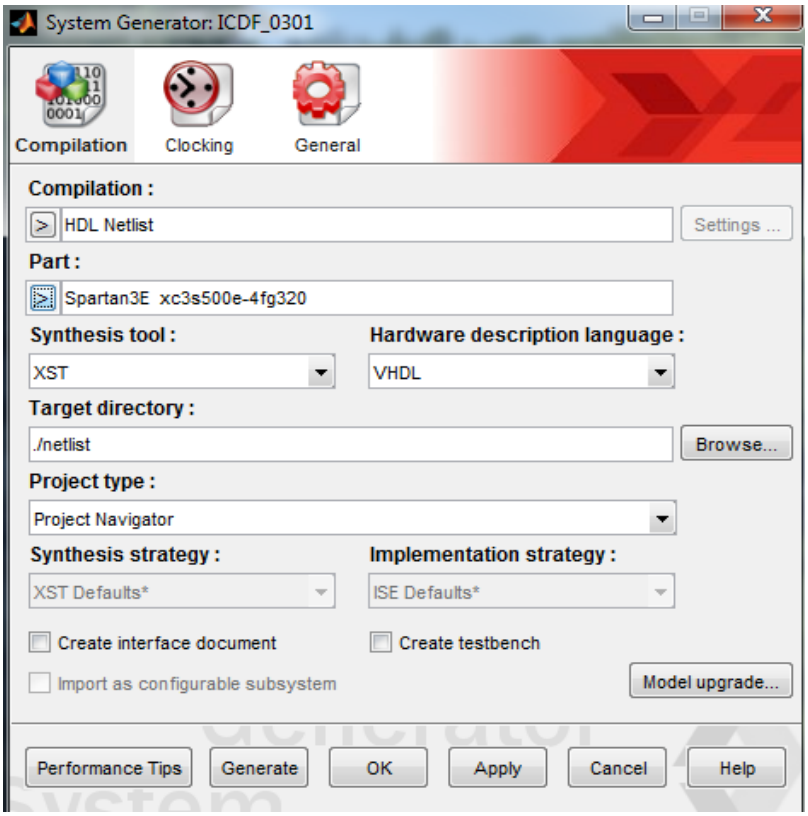

Figure 4.10. Paramètre de bloc System Generator.

4.4.2 Résultats de simulation de system Generator

En ajoutant un scope à la sortie (après le bloc Gateway) pour voire le signal de bruit gaussien :

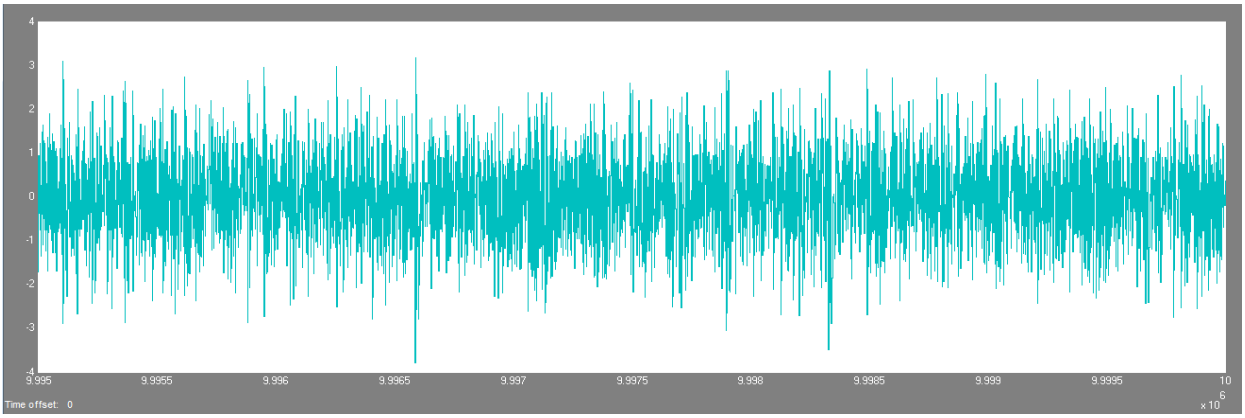

Figure 4.12. Résultats à la sortie (scope).

A la fin de la simulation les résultats obtenus sont stockés dans Workspace dans un variable nommé « simout ». Voici la courbe de variable aléatoire gaussien avec System Generator avec 10 millions échantillons :

4.4.3 Histogramme de courbe ICDF par SG

En faisant les mêmes tests que de la partie logicielle :

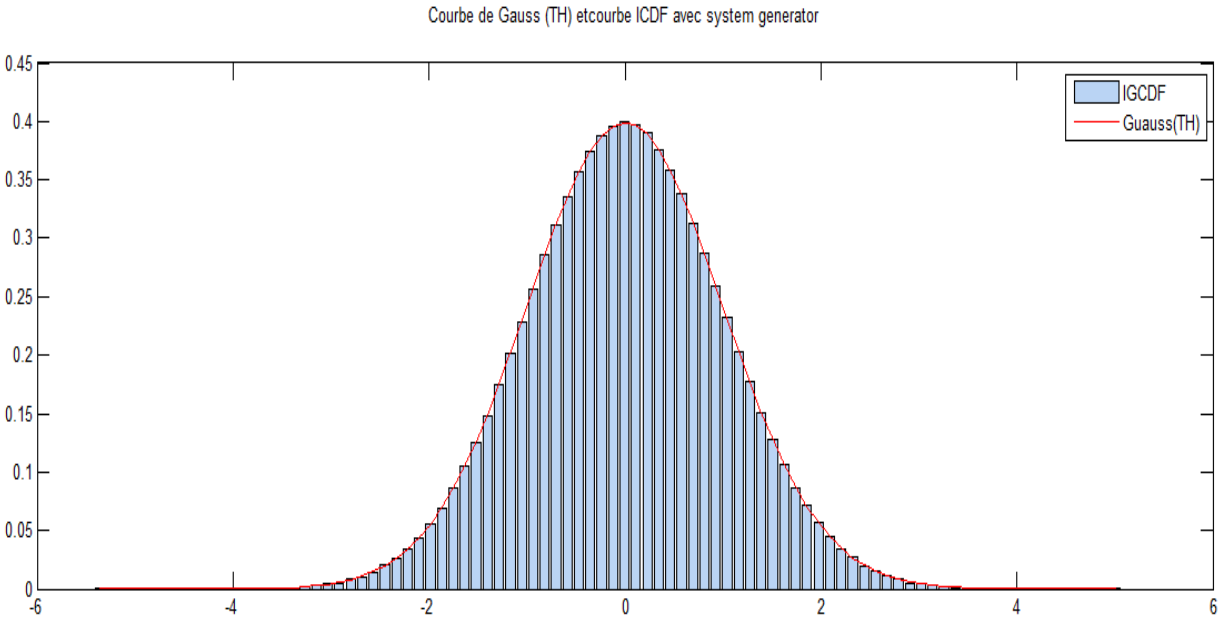

Figure 4.13. Distribution d'une va gaussienne icdf par SG.

4.4.4 Teste de densité spectral

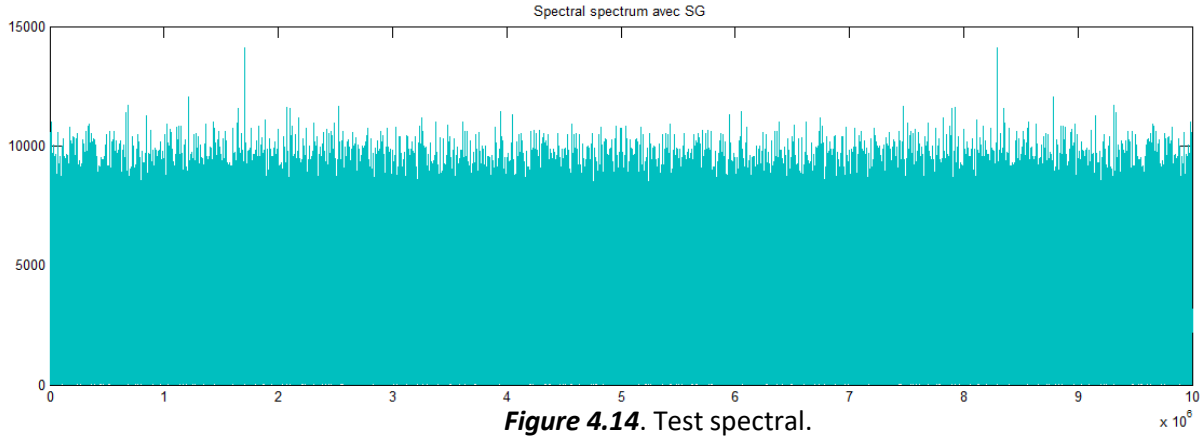

Figure 4.14. Test spectral.

4.4.5 Test d'autocorrélation

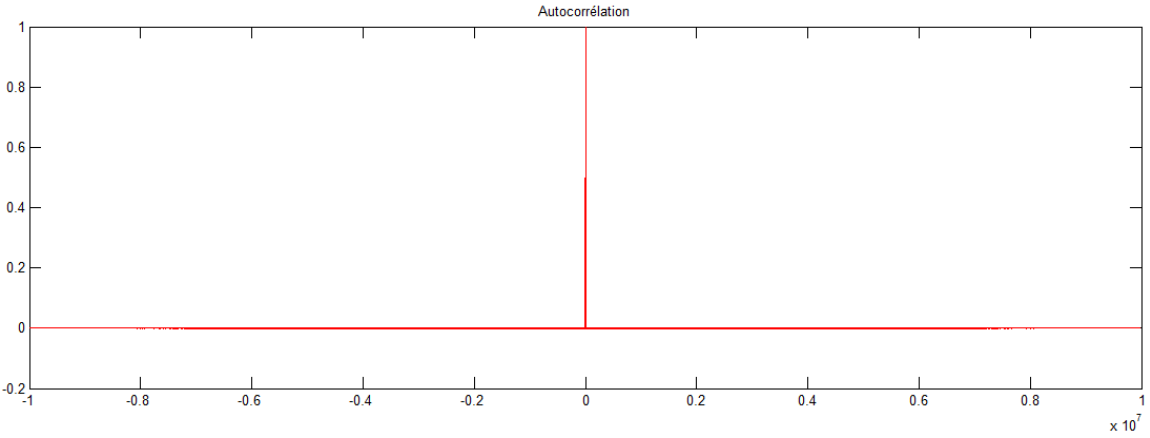

Figure 4.15. Test d'autocorrélation.

4.4.6 Comparaison du résultats courbe icdf logicielle et pratique

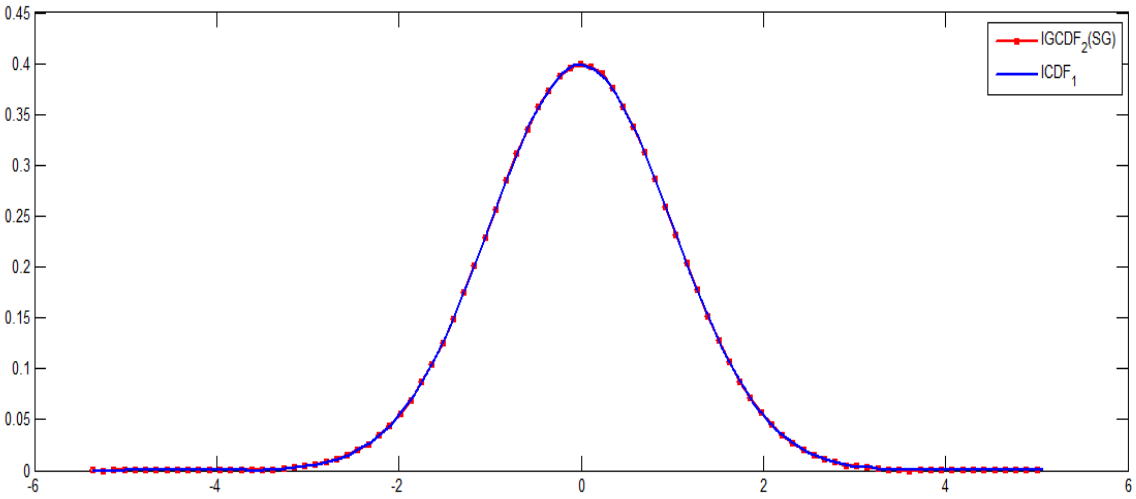

Figure 4.16. Comparaison entre les 2 courbes.

Interprétation

Ce graphe montre que la courbe obtenue la simulation de SG est presque identique que celle obtenue de simulation du code MatLab.

4.5 Conclusion

Tout au long de ce chapitre, divers description VHDL ont été décrit, plusieurs simulations ont été fait sur MatLab, ISE et SG. En examinant les résultats obtenus, nous avons pu en déduire que notre générateur de bruit gaussien via l'implémentation matérielle est sous norme.

L'avantage principal de notre travail est d'avoir les résultats plus rapidement tous en utilisant le minimum de composant matérielle.

Le principale objective de ce mémoire est l'amélioration de la méthode ICDF.

Le travail est faire l'exploration des architectures de calcules sur FPGA, connaitre les différentes méthodes pour la génération des nombres aléatoires gaussiens, et l'implémentation d'une méthode choisie en matérielle.

Nous avons présenté les bases composant d'une carte FPGA et le rôle de chaque composant et leur caractéristique, et on a bien spécifier les composant utilisé dans notre travail, les FPGA sont utilisés dans diverses applications nécessitant de l'électronique numérique (télécommunications, aéronautique, transports…)

On a parlé de certaine architecture matérielle qui génère le bruit gaussien et leur classification, il existe plusieurs méthodes et chaque un ses propriétés, et propre avantage et inconvénients. La conception présentée dans ce projet développe une nouvelle méthodologie inspirée par la méthode ICDF.

Le projet nécessite un algorithme en langage MatLab qui exprime les différentes étapes de notre méthode proposée pour qu'elle puisse être transformé en langage matérielle. On a choisi le langage VHDL pour la mise en œuvre matérielle, cependant un autre logiciel était nécessaire pour compléter le travail et tester les résultats.

Après que tous simulation ont été faite et tous les résultats ont été reçus et tous les calculs sont réaliser, on a conclu que l'objectif du projet est atteint.

[1] A. Blanchardon, «Synthèse d'architectures de circuits FPGA tolérants aux,» paris, 2015.

[2] «Enseignement/Architecture/Cours/Memory,» [En ligne]. Available: https://www.irif.fr/~carton/Enseignement/Architecture/Cours/Memory/.

[3] A. SAHOUR, «IMPLEMENTATION SUR FPGA D'UN ALGORITHME DE,» Annaba, 2013.

[4] N. Eastland, «Structure of an FPGA,» [En ligne]. Available: https://blog.digilentinc.com/structure-of-an-fpga/.

[5] «Configurable Logic Blocks (CLBs) on an FPGA,» 27 02 2020. [En ligne]. Available: https://www.ni.com/documentation/en/labview-comms/latest/fpga targets/configurable-logic blocks/#:~:text=A%20configurable%20logic%20block%20(CLB,synchronize%20co e%20on%20the%20FPGA.&text=A%20flip%2Dflop%20is%20the%20smallest%20s orage%20resource%20on%20t.

[6] M. V. Iştoan, "High-Performance Coarse Operators for FPGA-based Computing," LYON, 2017.

[7] «Overview of Lookup Tables in FPGA Design,» 4 04 2019. [En ligne]. Available: https://hardwarebee.com/overview-of-lookup-tables-in-fpga-design/.

[8] «Understanding FPGA Architecture Flip Flop,» 2018. [En ligne]. Available: https://www.xilinx.com/html_docs/xilinx2017_4/sdaccel_doc/ksg1504034293914.html.

[9] technologuepro, «cours-dsp/chapitre-1-DSP-introduction.,» 17 04 2011. [En ligne]. Available: https://www.technologuepro.com/cours-dsp/chapitre-1-DSP-introduction.pdf.

[10] «Versal ACAP DSP Engine Architecture Manual,» 16 july 2020. [En ligne]. Available: https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf.

[11] Z. E. B. K. H. A. Zeinab Seifoori, «Introduction to Emerging SRAM-Based FPGA Architectures in Dark Silicon Era,» Department of Computer Engineering, Sharif University of Technology, Tehran, Iran, Iran, 2018.

[12] A. MRABET, l'École Nationale d'Ingénieurs de Tunis, tunis, 2017.

[13] Xilinx, «Versal ACAP Configurable Architecture Manual,» xilinx, 2020.

[14] «What is a Block RAM (BRAM) in an FPGA? Tutorial for beginners,» [En ligne]. Available: https://www.nandland.com/articles/block-ram-in-fpga.html.

[15] helene.guerin, «perso.univ-rennes1.fr,» Université Rennes 1, [En ligne]. Available: https://perso.univ-rennes1.fr/helene.guerin/enseignement/.

[16] khan academy, «La loi normale - Savoirs et savoir-faire,» [En ligne]. Available: https://fr.khanacademy.org/math/be-6eme secondaire2h2/xa29f433c00318f09:probabilites/xa29f433c00318f09:loi-binomiale/a/normal-distributions-review.

[17] T. Verdel, «Décision et Prévision Statistiques,» (2007).

[18] V. T. a. J. V. R. Gutierrez, «Hardware Architecture of a Gaussian Noise generator Based on the Inversion Method,» IEEE, 2012.

[19] futura-sciences, «Les bienfaits du bruit blanc,» futura-sciences, [En ligne]. Available: https://www.futura-sciences.com/sciences/definitions/physique-bruit-blanc-4053/.

[20] J. H. K. Sang Gyu Kwak, «Central limit theorem: the cornerstone of modern statistics,» Daegu, 2017.

[21] J. S. M. A. HEMANI, «Gaussian Random Number Generation: A Survey on Hardware architecture,» Royal Institute of Technology, Sweden, 2016.

[22] A. GANTI, «Central Limit Theorem (CLT),» 13 septembre 2019. [En ligne]. Available: https://www.investopedia.com/terms/c/central_limit_theorem.asp.

[23] S. Glen, «Central Limit Theorem: Definition and Examples in Easy Steps,» [En ligne]. Available: https://www.statisticshowto.com/probability-and-statistics/normal-distributions/central-limit-theorem-definition-examples/.

[24] onlinestatbook, «Sampling Distributions,» [En ligne]. Available: http://onlinestatbook.com/stat_sim/sampling_dist/index.html.

[25] ,. J. J. Z. S. W. Yuan Li, «A Word-Length Optimized Hardware Gaussian Random Number Generator Based on the Box-Muller Method».

[26] D. T. Lee Howes, «Chapter 37. Efficient Random Number Generation and Application Using CUDA,» [En ligne]. Available: https://developer.nvidia.com/gpugems/gpugems3/part-vi-gpu-computing/chapter-37-efficient-random-number-generation-and-application.

[27] A. A. G. M. a. W. L. Dong-U Lee, «Optimizing Hardware Function Evaluation,» IEEE, 2005.

[28] C. Lauter, «Rigorous Polynomial Approximation,» Sorbonne Universite, Paris, 2018.

[29] D.-U. L. W. L. a. J. D. V. Ray C. C. Cheung, «Hardware Generation of Arbitrary Random Number Distributions From Uniform Distributions Via the Inversion Method,» IEEE, 2007.

[30] math.univ-paris13, «LFSR et RSA,» [En ligne]. Available: https://www.math.univ-paris13.fr/~boyer/enseignement/java-td/td2-rsa.html.

[31] .techopedia., «Linear Feedback Shift Register (LFSR),» [En ligne]. Available: https://www.techopedia.com/definition/5687/linear-feedback-shift-register-lfsr.

[32] J.-G. MAILLOUX, «PROTOTYPAGE RAPIDE DE LA COMMANDE VECTORIELLE SUR FPGA À L'AIDE DES OUTILS SIMULINK - SYSTEM GENERATOR,» UNIVERSITÉ DU QUÉBEC, QUÉBEC, MARS 2008.

[33] C. B. B. AHMED, «Fault-mitigation strategies for reliable FPGA,» Rennes, 2016.