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 Projet de Fin d'Études

Présenté par

Abdi Naima

&

Skander Sarah

Pour l'obtention du diplôme d'ingénieur d'état en Électronique option communication (contrôle)

Étude et implémentation sur FPGA d'un générateur de nombres aléatoires Gaussiens avec la méthode Box-Muller segmentée.

Proposé par : **Mr. Maamoun Mountassar**.

Co-promoteur : **Mr.Dahmani Samir**.

Année Universitaire 2021-2022

Remerciement

Nos remerciements sincères vont en premier à ALLAH le tout puissant qui nous a donné la force et le courage pour effectuer et mener à bien ce modeste travail de recherche.

*Nos vifs remerciements vont directement à notre encadreur de recherche Monsieur « **Mountassar Maamoun** » Qui nous a fait part de son savoir, ainsi que l'écoute, son aide précieuse et pour ces conseils avisés tout au long de la réalisation de ce mémoire.*

Nos remerciements vont aux membres de jury qui ont accepté de lire ce travail et l'évaluer.

Nous remercions aussi tous les enseignants et les enseignantes de département d'Électronique de Blida -1-.

Nos remerciements s'adressent également aux membres de nos familles surtout nos parents et toutes les personnes qui nous ont soutenu de près et loin à la réalisation et la finalisation de ce travail.

Abdi Naima.

Skander Sarah.

Dédicace

Je dédie ce modeste travail de recherche

A Ma source de joie et de bonheur, mes chères parents «Meziane » et « Ouardia »

Pour leur soutien, aide et sacrifice et l'encouragement pour me voir réussir,

Qu'Allah vous garde en bonne santé et vous protège.

A Mes chères Sœurs que j'aime beaucoup « Zora » et «Soumia» et « Sarah ».

D'avoir était toujours là pour m'encourager.

A Ma petit frère « Abd al hadi ».

A Mes chers beaux-frères que je respecte «Fateh» «Oussama »

Pour leur encouragement et aide.

A Mes chers neveux « Amir » et « Younes »

Ma petite princesse Ma nièce « Ines »

La joie de la Famille.

A Mon binôme que j'aime « Skander Sarah » pour son sérieux et sa confiance.

A tous ceux qui m'ont soutenu de près ou de loin à la réalisation de ce modeste travail.

Abdi Naima.

Dédicace

Je dédie ce mémoire à : Tout d'abord, je rends grâce à Allah, le tout puissant, que sa lumière nous guide vers lui, et que son nom soit l'élixir de nos peines et douleurs. Avec joie et plaisir, fierté et respect.

Je dédie ce modeste travail à : mes très chers parents, jamais je ne saurais m'exprimer quant aux sacrifices et aux dévouements que vous avez consacré à mon éducation et à mes études. Les mots expressifs soient-ils restés faibles pour énoncer ma gratitude hautement profonde.

Mon cher papa, « Nour Eddine », qui a toujours la source de tendresse, de noblesse, de patience et d'encouragements.

Ma chère mère, « Malika », que je ne cesse de remercier pour tout ce qu'elle m'a donné. Vous représentez pour moi le symbole de la bonté par excellence, la source de tendresse qui n'a pas cessé de m'encourager. Que Dieu la récompense pour tous ces bienfaits.

Ma belle grande mère « Houria » Mes petites princesses « Arwa,Iline » Mes douces sœurs « Hanane, Maria, Amira ». Mon cher frère « Mohammed ». J'aime beaucoup mes chèresQue Dieu Les Protège.

A mes chères amies « Marwa, Bouchra, Rima, Naima ». Merci à tous les amis avec qui j'ai partagé de nombreux bons moments au cours des années d'étude. Je m'excuse auprès de ceux que j'aurais pu oublier....

Skander Sarah.

ملخص: يقدم هذا العمل دراسة بنية مولد الأرقام العشوائية'Box–Muller Gaussien' باستخدام منطق 'FPGA' كتل ذاكرة الوصول العشوائي'FPGA' مرتبطة بمضاعف لتقليل حجم الذاكرة المطلوبة لتنفيذ خوارزمية'Box–muller' يتم إنتاج المتغيرات العشوائية المنتظمة ومحدد الإرسال بواسطة وحدة قائمة على سجل التحول'LFSR'.

يمكن تقليل حجم الذاكرة المطلوب من خلال الحفاظ على الدقة التقليدية وتظهر جميع النتائج التي تم الحصول عليها أن هيكل 'Box–Muller'على يقلل من حجم الذاكرة المطلوب، عندما يتم تنفيذه على'FPGA'

Résumé : étudedune structure de générateur de nombres aléatoires gaussiens Box-Muller, utilisant la logique PFGA. Les Blocs RAM FPGA, sont associes à un multiplexeur pour réduire la taille de mémoire requise pour implémenterl'algorithme de Box-Muller.Les variables aléatoires uniformes et le sélecteur de multiplexeur sont produits par une unité basée sur un registre à décalage LFSR. On peut réduire la taille de mémoire requise en conservant la précision conventionnelle et tout les résultats obtenus montrent que la structure de Box-Muller fait réduire la taille de mémoire requise, lorsqu'elle est implémentée sur FPGA.

Mots clés :PFGA, Blocs RAM FPGA, LFSR, Box-Muller.

Abstract: study of a Box-Muller Gaussian random number generator structure, using PFGA logic. FPGA RAM Blocks are associated with a multiplexer to reduce the memory size required to implement the Box-Muller algorithm.

The uniform random variables and the multiplexer selector are produced by an LFSR shift register-based unit. One can reduce the required memory size by keeping the conventional precision and all the obtained results show that the Box-Muller structure makes reduce the required memory size, when it is implemented on FPGA.

Keywords :PFGA, Blocs RAM FPGA, LFSR, Box-Muller.

Listes des acronymes et abréviations

PLA : Programmable Logic Array.

SSI: Small Scale integration.

MSI: Medium Scale integration.

LSI: large scaleintegration.

ASIC: Application Specific integrated circuit.

FPGA: Field Programmable Gate Array.

DSP: Digital Signal Processor.

CLB: Configurable Logic Blocs.

LUT: Luck Up Table.

RAM: Random Access Memory.

ROM:ReadOnlyMemory.

PLD: Programmable Logic Device.

Asic: Application specificintegrated circuit.

LAB: Logic array module.

CMT: Module de Gestion d'horloge.

FIFO: First In First Out.

ADC: Analogue-to-digital converter.

Carte SD: Secure Digitale.

RNG: RandomNumberGenerator.

TCL: théorèmelimite centrale.

BM: Box-Muller.

PDF: Probability Density Fonction.

CORDIC: COordinate Rotation Digital Computer.

GRNG: générateurs de nombres aléatoires gaussiens.

Va: Variable Aléatoire.

LFSR: LinearFeedbackShiftRegister.

ISE: IntegratedSoftwareEnvironment.

HDL:HardwareDescriptionLangage.

PCB: Printed Circuit Board.

RTL: Register Transfer Language.

FIPS: Federal Information Processing Standard.

NIST:TechnologyStatistical Test Suite.

SIPO: Serial In - Parallel Out.

$\boldsymbol{\sigma}$: La variation fixe sigma.

Table des matières

Chapitre 2 : Architectures matérielles classiques des générateurs de nombres aléatoires gaussiens

Chapitre 3 : Etude de la méthode Box-Muller à découpage d'intervalles

Liste des figures

Chapitre 4 : Implémentation et résultats.

Liste des tableaux

1. Introduction générale

L'électronique représente un marché très vaste et lucratif, et un domaine dynamique plein de nouvelles découvertes et de nouvelles applications. Un domaine caractérisé par la miniaturisation et la simplification.

Lesméthodes numériques et les algorithmes de génération de nombres (GRNG) ont une longue histoire en mathématiques et communications. Systèmes de télécommunications, et dans décodeurs spécifiques qui bénéficient de cette technologie Progrès dans GRNG pour fournir des rendements très souhaitables. Ces Les générateurs varient considérablement en termes de calculs complexité, utilisation des ressources matérielles, vitesse et précis.

L'une des principales raisons d'implémenter un FPGA est utilisation logique de l'efficacité et de l'équilibre, montrant que l'utilisation relative des blocs de RAM et des tranches DSP Utilisation des LUT.

Notre travail contribue au développement technologique, c'est générer des variables aléatoires gaussiens à la base d'un registre à décalage LFSR (pseudo aléatoire),il est basé sur l'utilisation optimisée des circuits programmables des FPGAs, pour réduire la taille de mémoire requise et cela se fait très rapidement.

Pour la présentation de notre travail, nous allons répartir cette présentation sur quatre chapitres :

Dans le chapitre 1, nous allons présenter la logique programmable qui a un impact sur l'accélération du développement de la technologie des mémoires. Puis nous allons faire tout d'abord la présentation de la DSPs sur FPGA. Ensuite nous allons parler sur les structures de blocs logiques configurables et Blocks RAM pour faire court dans le stockage général des données.

Dans le chapitre deux,nous allons présenter l’architecture matérielles classiques des générateurs de nombres aléatoires gaussiens qui est fait pour produire une séquence binaire aléatoire indépendante, imprédictible et uniformément répartie. Les RNGs sont nécessaires dans toutes sortes d'applications, par exemple : télécommunications.

Dans le chapitre trois,nous allons présenterla méthode de Box Muller est l'une des méthodes de génération de nombres aléatoires ga.

Chapitre 1 Logique programmable et calcul sur FPGA

1.1. Introduction

Dans ce chapitre nous présenterons la logique programmable qui a un impact sur l'accélération du développement de la technologie des mémoires. Puis on va présenter Les FPGA qu'ont évolué rapidement au fil des années et sont capables de répondre à de nombreuses exigences de conception en matière de flexibilité, de vitesse de traitement et d'alimentation, ce qui les rend utiles pour une vaste gamme d'applications.

Nous allons faire tout d'abord la présentation de la DSPs sur FPGA. Ensuite nous allons parler sur les structures de blocs logiques configurables et Blocks RAM pour faire court dans le stockage général des données.

1.2. Les modules DSPs sur FPGA

1.2.1. Définition de FPGA [1]

FPGA est « *Field-programmable gatearray* », ce qui peut être traduit par « réseau de portes programmables sur site ». C'est un circuit intégré fait pour être (re)programmé par l'utilisateur après sa fabrication en utilisant un langage informatique spécifique, donc sans modifier le matériel.

Un FPGA se distingue ainsi d'un Asic (Application specificintegrated circuit), un circuit intégré spécialisé qui ne peut pas être reprogrammé une fois construit. Les FPGA sont performants, rapides et reprogrammables, ce qui permet de faire des économies de temps et d'argent. Contrairement aux microprocesseurs traditionnels, un FPGA peut exécuter plusieurs opérations en parallèle.

Figure 1.1 Field-programmable gate array. ***[2]***

1.2.2. Comment fonctionne un FPGA ? [1]

L'architecture d'un FPGA est composée :

- D'un ensemble de blocs logiques configurables.
- D'un canal de routage programmable contenant des commutateurs.
- De blocs I/O transportant les signaux à l'intérieur ou à l'extérieur du FPGA.
- De blocs de mémoire.

Ces blocs créent un réseau physique de portes logiques qui peuvent être personnalisées pour effectuer des tâches informatiques spécifiques. De nombreux autres éléments peuvent être ajoutés, comme des CPU (processeurs), des contrôleurs de mémoire, des contrôleurs USB ou des cartes réseau.

Les blocs logiques configurables constituent le cœur d'un FPGA. Ils contiennent différents éléments selon le constructeur. Xilinx utilise des CLB (Configurable logic block), bloc logique configurable, en français. Altera utilise des LAB (Logic array module), ce qui signifie « module de réseau logique ». Toutefois, un bloc logique configurable contient toujours une ou plusieurs tables de conversion (des LUT) à trois ou quatre entrées, un D Flip-Flop (un circuit électronique numérique utilisé pour retarder le changement d'état de son signal de sortie) et une sortie.

1.2.3. Le Digital Signal Processor DSP [3]

Un DSP pour Digital Signal Processor en anglais est un processeur spécialisé dans le traitement numérique du signal. Son architecture est optimisée pour traiter une grande quantité de données en parallèle à chaque cycle d'horloge. Ce mode de fonctionnement est très efficace pour traiter des signaux numériques (filtrage, compression, extraction, etc.).

Le premier DSP a été produit en 1982 par Texas Instruments. Depuis, cinq autres générations de DSP sont apparues. Les processeurs des générations 1, 2 et 5 sont en virgule fixe, les générations 3, 4, 6 en virgule flottante.

Les principaux fabricants de DSP sont :

• Texas Instruments •AnalogDevices• Motorola •Zilog• Lucent • Nec • Zoran •Zsp•Microchip

Figure 1.2 Le Digital Signal Processor DSP. **[4]**

1.2.4. Rôle du DSP [5]

Un traitement numérique du signal, qu'il provienne du son ou d'une image vidéo, est rendu accessible par le DSP grâce à son unité de calcul spécifique multiplicateur / additionneur / accumulateur de données.

En effet, tout DSP est prévu pour effectuer le plus rapidement possible, en principe en un seul cycle d'horloge, l'opération multiplication/addition sur des grandeurs numériques : MR=X·Y +R

Où X et Y sont soit des données, soit des constantes et R une donnée, une constante ou un résultat précédent. MR est alors le résultat de l'opération arithmétique. Si le DSP fonctionne en virgule fixe avec des données sur 16 bits, le résultat MR est alors sur 32 bits (ou plus, selon l'architecture). Si l'utilisateur ne conserve que les 16 bits de poids fort, le calcul est alors effectué en simple précision. Si les 32 bits sont utilisés, on parle de double précision : le temps de calcul est alors plus long. Si le DSP fonctionne en virgule flottante avec des données en 32 bits, le résultat MR est alors sur 40 bits (ou plus, selon l'architecture). L'utilisateur ne prend en compte que les données de 32 bits en ignorant les bits de poids faibles de la mantisse.

1.2.5. Classification des DSP [6]

On ne peut jamais faire une classification définitive des DSP, parce que chaque constructeur met sur le marché tous les ans un nouveau composant qui surclasse les anciens ou les concurrents par la puissance de calcul, la rapidité, le nombre de registres, de Timers, de ports série…

1.2.5.1 DSP à virgule fixe

Les données sont représentées comme étant des nombres fractionnaires à virgule fixe (exemple -1.0 à 1.0), ou comme des entiers classiques, le programmeur doit rester vigilant à chaque étape d'un calcul. Ces DSP sont plus difficiles à programmer.

1.2.5.2. DSP à virgule flottante

Les données sont représentées en utilisant une mantisse et un exposant : n=mantisse*2exposant Digital Signal Processing, Les DSP à virgule flottante fournissent une très grande dynamique et sont plus chers et consomment plus d'énergie.

En termes de rapidité, les DSP à virgule fixe se placent d'ordinaire devant leurs homologues à virgule flottante, ce qui constitue un critère de choix important. Digital Signal Processing.

1.2.6. Performance des DSP [6]

Plus que pour un microprocesseur classique, les performances d'un DSP conditionnent son domaine d'application. La plupart des DSP sont particulièrement destinés à des applications « temps réel » et spécialisées, c'est à dire des applications où le temps de traitement est bien sûr primordial, mais où la diversité des événements à traiter n'est pas notablement importante.

De ce point de vue, l'approche DSP s'apparente plus à une étude « électronique » visant à réaliser une ou des fonctions de traitements de signal, que d'une approche informatique temps réel et/ou multitâche traditionnelle. Il existe cependant des applications ou le DSP assure à la fois des fonctions de traitements numériques du signal et les fonctions générales d'un microprocesseur au cœur d'un système informatique classique. Dans tous les cas, les performances du DSP sont critiques. Le concepteur d'un système à base de DSP doit évaluer d'une part la « puissance » nécessaire pour réaliser les traitements numériques voulus, et d'autre part les performances des DSP disponibles pour réaliser son application.

1.3. Les CLBs

1.3.1. Définition des CLBs (Configurable Logic Blocs) [7] [8] [9]

Un bloc logique configurable (CLB) est le bloc logique répétitif de base sur un FPGA. Il existe des centaines de blocs logiques similaires disponibles sur le FPGA connectés via des ressources de routage.

Qui contient une structure de table de recherche flexible qui peut implémenter une logique plus des éléments de stockage tels que des bascules et des verrous, exécuter diverses fonctions logiques et stocker des données.

Une vue d'ensemble LCB de haut niveau est présentée tandis que le CLB fournit la capacité logique, le routage d'interconnexion flexible, achemine le signal entre les CLB et vers et depuis les broches d'entrée/sortie (E/S).

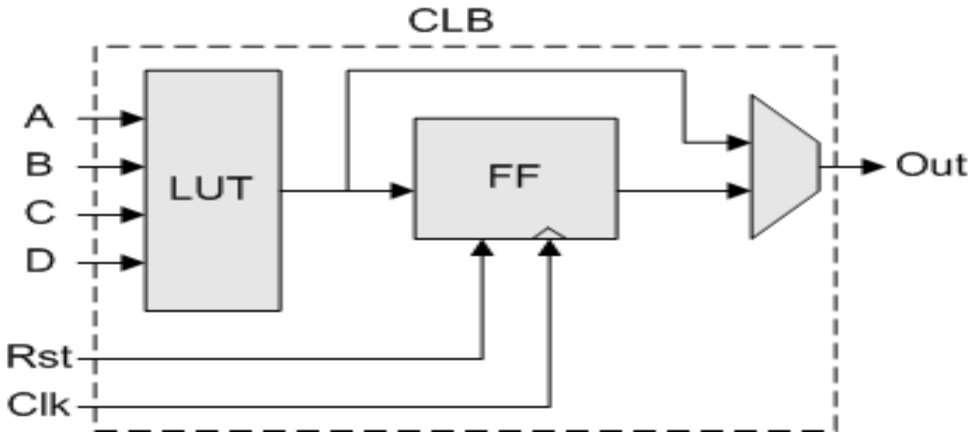

*Figure 1.3*configurable logic block (CLB).**[9]**

1.3.2.1. SLICEM et SLICE :

Un CLB est le composant fondamental d'un FPGA, permettant à l'utilisateur d'implémenter pratiquement n'importe quelle fonctionnalité logique dans la puce. Ceci est réalisé par l'utilisation de deux ensembles de composants similaires dans un bloc, appelés tranches. Il existe deux types de tranches différents, appelés SLICEM et SLICEL, et chaque CLB peut contenir soit un SLICEM et un SLICEL, soit deux SLICEL. Il y a environ deux fois plus de SLICEL que de SLICEM sur une puce.

- Premièrement, les tranches d'un CLB ne sont pas connectées les unes aux autres. Ils sont physiquement orientés de manière similaire au diagramme afin qu'ils puissent être connectés avec le même type de tranche (SLICEM ou SLICEL) dans les CLB au-dessus ou au-dessous, créant des colonnes. Cela permet des interconnexions entre SLICEM ou SLICEL dans une colonne pour créer des fonctions à grande échelle.
- La caractéristique distinctive des deux types de tranches est la configurabilité du SLICEM. SLICEM peut être configuré de sorte que lestables de consultation qu'il contient puissent agir comme des registres à décalage ou comme stockage de données (créant une mémoire distribuée sur la puce) en plus de sa fonctionnalité logique normale.

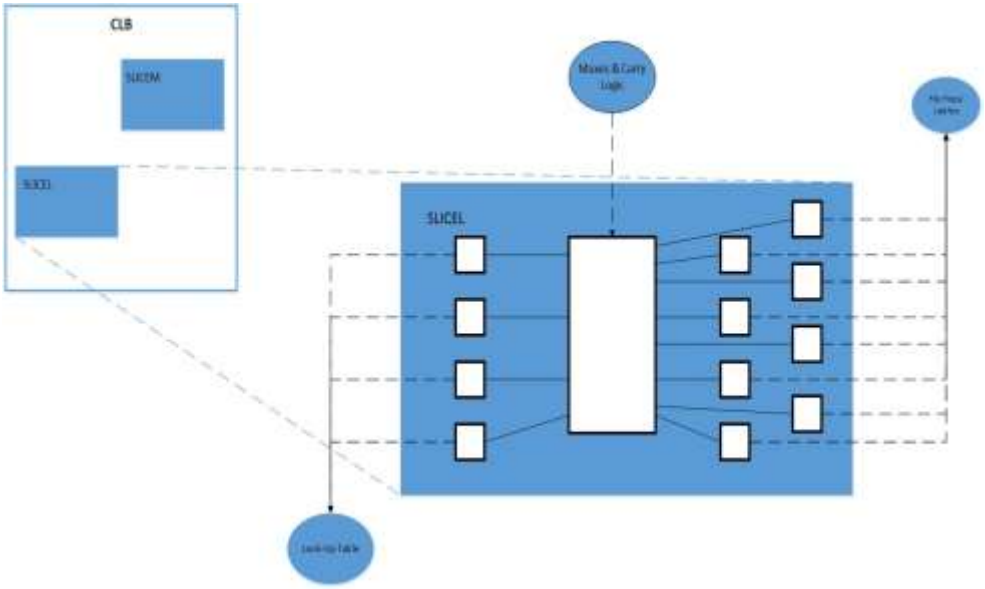

Figure 1.4Un schéma grossier illustrant structure interne de CLB. ***[11]***

1.3.3. Les composantes essentielles des CLBs [7]

1.3.3.1Tongs/Loquets :

Une bascule est un périphérique de stockage primitif qui peut stocker un seul bit d'information. Chaque tranche contient huit de ces éléments de stockage. Quatre sont disponibles exclusivement en tant que bascules (mémoire synchrone) et les quatre autres peuvent être configurées soit en bascule standard, soit en bascule (mémoire asynchrone). Une dernière mise en garde est que lorsque les quatre qui peuvent être configurées comme un verrou le sont, les quatre autres bascules de la tranche deviennent inutilisables.

1.3.3.2Table de consultation (LUT)

Une LUT est le cœur du FPGA. Il contient toutes les sorties logiquement possibles de votre conception. Les LUT sont une architecture basée sur le multiplexeur dans laquelle les entrées sont les sorties possibles en fonction de la sélection correcte sur les lignes de sélection du multiplexeur.

1.3.3.3Multiplexeur

Un circuit effectue les transitions entre différentes sorties en fonction des entrées de lignes sélectionnées.

- Le FPGA Xilinx est principalement composé des structures unitaires suivantes : bloc logique configurable (CLB), module de gestion d'horloge (CMT), mémoire (RAM/FIFO), module de traitement du signal numérique (DSP) et certains modules uniques.

CLB est la principale ressource contenant la logique de conception dans FPGA et les principales fonctionnalités de conception logique. Les CLB réalisent la logique du FPGA configuré dans une baie. Chaque partie CLB est connectée à un réseau de commutateurs et contrôlée par celui-ci pour mettre en œuvre la logique, comme illustré à la figure 5.

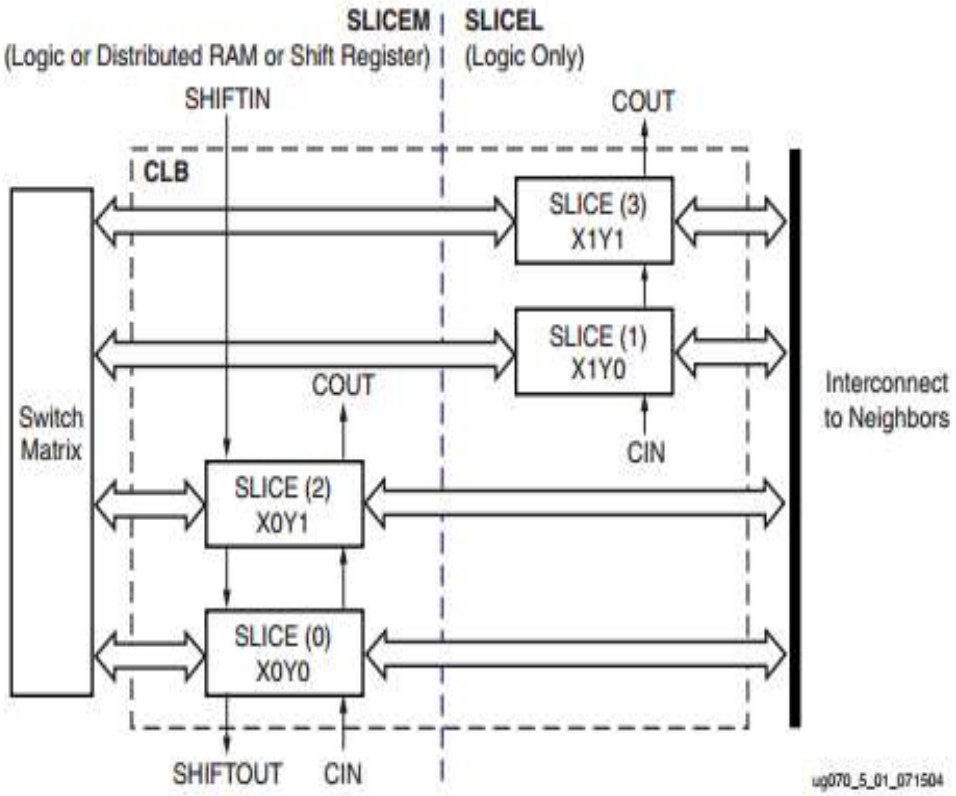

Figure 1.5Xilinx CLB. ***[7]***

1.3.4. Performances des CLBs [12]

Les blocs logiques sont de mettre en œuvre une logique combinatoire et séquentielle :

1.3.4.1. Logique combinatoire

La partie combinatoire permet de réaliser des fonctions arithmétiques et logiques de complexité variable. Il est en effet possible d'utiliser plusieurs méthodes de synthèse dont les principales sont : la synthèse de fonctions à 4 ou 5 variables avec des portes classiques ET, OU et NON, la synthèse de fonctions combinatoires à l'aide de mémoires vives. Dans ce dernier cas, on dit aussi réalisation de fonctions logique par LUT (Look-Up Table) signifiant table de réalisation (ou d'observation).
La configuration du bloc logique combinatoire consiste en trois types différents de tables de consultation : ces trois types couvrent les principaux modes de configuration des blocs logique des FPGA actuels. Ces modes combinatoires sont décrits en détail dans COMGen : mappage direct de composantes arbitraires des FPGA basés sur LUT.

1.3.4.2. Logique séquentielle

La partie séquentielle comporte en règle générale une ou deux bascules de type D. Afin de prendre en charge une grande variété de blocs logique séquentiels caractéristiques d'un modèle séquentiel flexible, ce sont les signaux de données, les signaux d'horloge, les entrées de commande asynchrones. La méthode d'implémentation pour les parties séquentielles se trouve dans la section.

1.4. Les BRAMs

1.4.1. Définition des BRAMs [13] [14]

Block RAMs (ou BRAM) signifie Block Random Access Memory. Les blocs RAM sont utilisés pour stocker efficacement de grandes quantités de données à l'intérieur de votre FPGA, telles que des images ou des vidéos, pour des machines d'état hautes performances ou un tampon FIFO, pour les grands registres à décalage, une grande table de consultation ou des ROM à l'intérieur du FPGA. C'est une partie discrète du FPGA, ce qui signifie qu'il n'y en a qu'un nombre limité sur la puce.

Le fpga peut avoir la RAM distribuée et la RAM en bloc. La RAM distribuée peut être réalisée à l'aide des LUT, et les BRAM sont réalisées à l'aide des blocs BRAM dédiés qui peuvent être programmés par l'outil de conception spécifique au fournisseur pour la configuration et la taille requises, la réalisation de RAM à port unique et de RAM à double port est possible en utilisant les BRAM. La capacité de BRAM dépend de l'architecture. Habituellement, plus le FPGA est gros et cher, plus il aura de RAM en bloc.

Un BRAM est utilisé pour stocker une grande quantité de données. Un bloc RAM a une largeur et une profondeur et peut être initialisé à une valeur non nulle lors de l'implémentation.

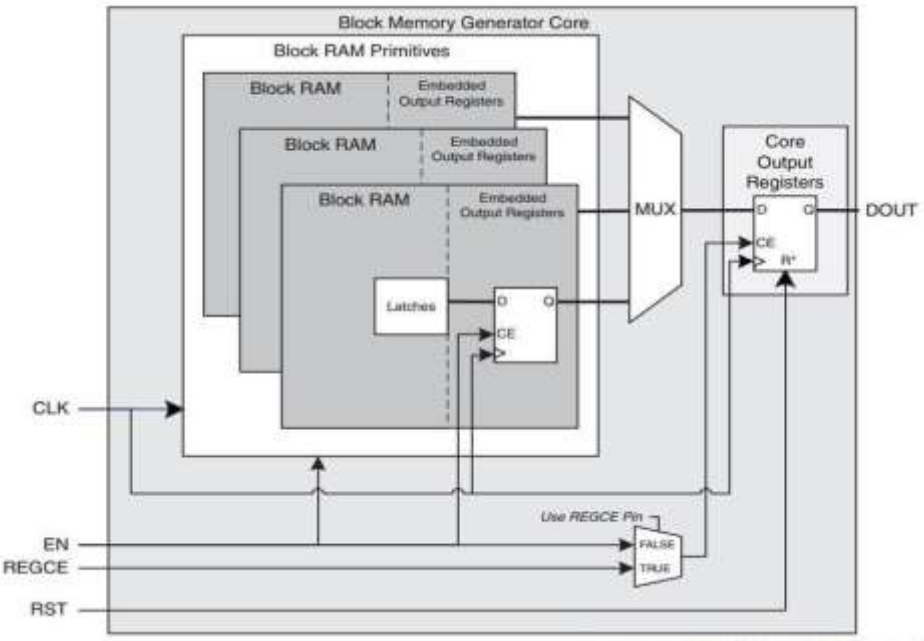

*Figure 1.6structure détaillée d'un BRAM.**[15]***

1.4.2. L'utilisateur des blocks RAM [13]

- Stockage de grandes tables de recherche (par exemple, conversion de Celsius en Fahrenheit)
- Stockage de données en lecture seule telles que les paramètres d'étalonnage
- Stockage des données lues sur un périphérique externe tel qu'une ADC ou un convertisseur Flash
- Création d'un FIFO pour stocker des données temporaires telles que la vidéo brute
- Croisement de domaines d'horloge à l'aide d'un FIFO
- En général, stocker une grande quantité de données.

1.4.3. Configuration BRAM

1.4.3.1. Port unique [13]

La configuration Single Port Block RAM est utile lorsqu'une seule interface doit récupérer des données. C'est aussi la configuration la plus simple et elle est utile pour certaines applications. Un exemple serait de stocker des données en lecture seule qui sont écrites sur une valeur fixe lorsque le FPGA est programmé.

- La façon dont ils fonctionnent est basée sur une horloge. Les données seront lues sur le front positif du cycle d'horloge à l'adresse spécifiée par Addr tant que le

signal WrEn n'est pas actif. Les valeurs lues sortent sur Rd Data, ce sont les données stockées dans la BRAM. Notez que vous ne pouvez lire qu'une seule valeur Rd Data par cycle d'horloge. Donc, si votre Block RAM contient 1024 valeurs, il faudra *au moins* 1024 cycles d'horloge pour lire le tout.

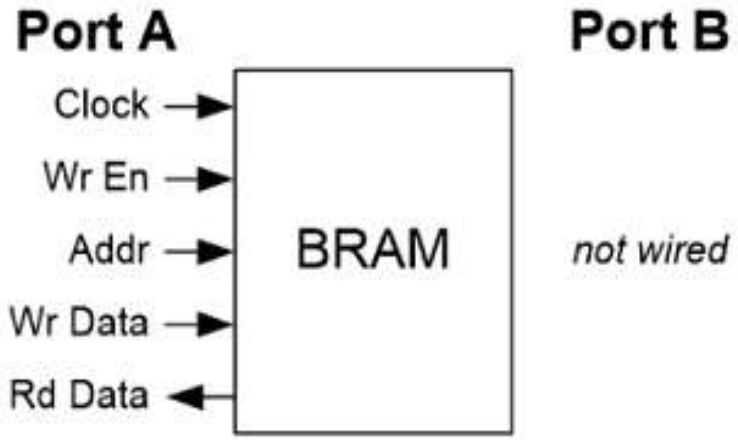

*Figure 1.7Configuration BRAM à port unique.**[13]***

1.4.3.2. Deux ports [13]

La configuration Dual Port Block RAM (ou DPRAM) se comporte exactement de la même manière que la configuration à port unique, sauf que vous avez un autre port disponible pour lire et écrire des données. Le port A et le port B se comportent exactement de la même manière. Le port A peut effectuer une lecture sur l'adresse 0 sur le même cycle d'horloge que le port B écrit à l'adresse 200. Par conséquent, une DPRAM est capable d'effectuer une écriture sur une adresse tout en lisant à partir d'une adresse complètement différente. Personnellement, je trouve que j'ai plus de cas d'utilisation pour les DPRAM que pour les RAM à port unique.

Un cas d'utilisation possible serait de stocker des données sur un périphérique externe. Par exemple, si vous souhaitez lire des données sur une carte SD, vous pouvez les stocker dans une RAM à double port, puis les lire plus tard. Ou peut-être souhaitez-vous vous connecter à un convertisseur analogique-numérique (ADC) et aurez-vous besoin d'un endroit pour stocker les valeurs ADC converties. Une DPRAM serait idéale pour cela. De plus, les RAM à double port sont généralement transformées en FIFO, qui sont probablement l'un des cas d'utilisation les plus courants pour Block RAM sur un FPGA.

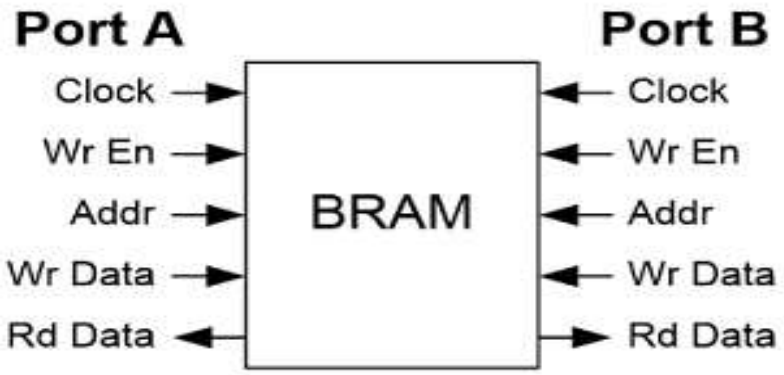

Figure 1.8Configuration BRAM à Deux ports. ***[13]***

1.5. Conclusion

Dans ce chapitre nous avons présenté les principes des circuits analogiques programmables et calcul sur FPGA.

Ce chapitre nous a permis de connaitre l'architecture globale de FPGA (CLB, I /O, BRAM), et ces caractéristiques :

- La flexibilité.
- Fonctionnement en parallèle.
- Sa vitesse (très rapide).
- Elle a plusieurs entrées et sorties INPUTS, OUTPUTS.
- Interagir avec plusieurs programmes comme : VHDL, VERILOG.

Les FPGA sont les plus couteux et plus compliqués à utiliser, mais ils offrent beaucoup plus de puissances et de flexibilité.

Chapitre 2 Architectures matérielles classiques des générateurs de nombres aléatoires gaussiens

2.1. Introduction

Dans ce chapitre nous présenterons l'architecture matérielle classique des générateurs de nombres aléatoires gaussiens qui est fait pour produire une séquence binaire aléatoire indépendante, imprédictible et uniformément répartie. Les RNGs (RNG, Random Number Generator) sont nécessaires dans toutes sortes d'applications, par exemple : télécommunications.

Nous allons faire tout d'abord la présentation de la méthode de la limite centrale qu'est sans doute et le plus important des statistiques, c'est le théorème qui nous est le plus utile pour les expérimentations que nous faisons dans le monde. Ensuite nous allons parler sur les architectures de : la méthode BOX-Muller puis la méthode récursive et la méthode du rejet.

2.2. Générateurs de nombres aléatoires gaussiens [16]

Les générateurs de nombres aléatoires gaussiens (GRNG) sont utilisés dans un grand nombre d'applications de modélisation et de simulation à forte intensité de calcul, notamment le transfert de chaleur et les systèmes de communication. Et la programmation évolutive. Compte tenu de leur importance. Il y a eu étonnamment peu de recherches sur leur implémentation matérielle efficace. McCCollum et al. Utilisé une table de recherche suivie d'une interpolation linéaire pour calculer la fonction de distribution cumulative inverse afin de générer des variables aléatoires avec une distribution arbitraire. Proposé d'utiliser l'algorithme Box-Muller pour générer des nombres aléatoires normalement distribués. Les fonctions élémentaires impliquées dans sa mise en œuvre sont exécutées à l'aide d'une approximation polynomiale non uniforme par morceaux. La méthode Wallace consiste à transformer un pool de nombres aléatoires en un nouveau pool de nombres aléatoires. A réalisé une implémentation matérielle de la méthode Walface et a montré les avantages de vitesse et de taille par rapport à la méthode Box-Muller. Nous proposons la méthode Ziggurat comme algorithme efficace à utiliser dans un GRNG. À ce jour, aucune implémentation matérielle de cette méthode n'a été signalée. L'algorithme Ziggurat permet d'utiliser des opérations rapides sur les entiers pour produire la plupart de ses sorties en un seul cycle. Pour un petit pourcentage de sorties, des fonctions élémentaires telles que le logarithme naturel et la fonction exponentielle doivent être calculées. Ceux-ci sont implémentés à l'aide d'approximations polynomiales utilisant une unité logique arithmétique (ALU) et une machine d'état. Le système résultant peut générer 169 millions de nombres aléatoires normalement distribués par seconde sur un appareil Xilinx XC2VP30-6.

2.3. Architecture de la méthode de la limite centrale

2.3.1. Définition de la limite centrale (TCL) [17] [18] [19]

Le théorème central limite est un théorème fondamental de la théorie statistique. Dans sa forme la plus simple, il prescrit que la somme d'un nombre suffisamment grand de variables aléatoires indépendantes et distribuées identiquement suit approximativement une loi normale.

Le théorème central limite stipule que chaque fois qu'un échantillon aléatoire de taille n est tiré d'une distribution avec moyenne et variance, la moyenne de l'échantillon sera approximativement distribuée normalement avec moyenne et variance. Plus la valeur de la taille de l'échantillon est grande, meilleure est l'approximation de la normale.

Ainsi Le TCL permet de réduire les erreurs d'approximation lors du calcul des fonctions rencontrées dans l'algorithme BM.

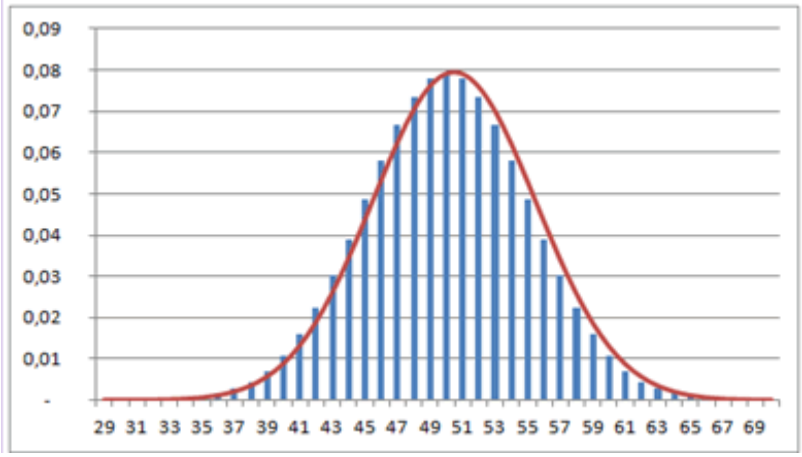

Figure 2. 1la limite centrale. ***[20]***

2.3.2. Démonstration du théorème central limite [21]

Pour un théorème d'une telle importance en statistiques et en probabilité appliquée, il existe une démonstration particulièrement simple utilisant les fonctions caractéristiques. Cette démonstration ressemble à celle d'une des lois des grands nombres. Pour une variable aléatoire Y d'espérance 0 et de variance 1, la fonction caractéristique de *Y* admet le développement limité :

$$\varphi Y(t) = 1 - \frac{t^2}{2} + 0(t^2), \qquad t \to 0$$

Si Y_i vaut $\frac{X_i - \mu}{\sigma}$il est facile de voir quela moyenne centrée réduite des observations X_1, X_2, ..., X_n est simplement :

$$Z_n = \frac{\overline{X} - \mu}{\sigma/\sqrt{n}} = \sum_{i=1}^{n} \frac{Y_i}{\sqrt{n}}$$

D'après les propriétés élémentaires des fonctions caractéristiques, la fonction caractéristique de Z_n est :

$$\left[\varphi Y\left(\frac{t}{\sqrt{n}}\right)\right]^n = \left[1 - \frac{t^2}{2n} + 0\left(\frac{t^2}{n}\right)\right]^n \to e^{-t^2/2} \quad lorsque\ n \to +\infty$$

Mais cette limite est la fonction caractéristique de la loi normale centrée réduite $N(0,1)$, d'où l'on déduit le théorème de la limite centrale grâce au théorème de continuité de Lévy, qui affirme que la convergence des fonctions caractéristiques implique la convergence en loi.

2.3.3. Intérêt de ce théorème [21]

On peut parfois lire dans la presse générale que la courbe en cloche représente la loi du hasard, ce qui n'a pas grande signification. Le succès sans égal de la loi de Gauss est la conséquence directe du théorème de la limite centrale et il est renforcé par la commodité relative d'utilisation de cette loi.

En elle-même, la convergence vers la loi normale de nombreuses sommes de variables aléatoires lorsque leur nombre tend vers l'infini n'intéresse que le mathématicien. Pour le praticien, il est intéressant de s'arrêter un peu avant la limite : la somme d'un grand nombre de ces variables est presque gaussienne, ce qui fournit une approximation souvent plus facilement utilisable que la loi exacte.

En s'éloignant encore plus de la théorie, on peut dire que bon nombre de phénomènes naturels sont dus à la superposition de causes nombreuses, plus ou moins indépendantes. Il en résulte que la loi normale les représente de manière raisonnablement efficace.

À l'inverse, on peut dire qu'aucun phénomène concret n'est vraiment gaussien car il ne peut dépasser certaines limites, en particulier s'il est à valeurs positives.

2.3.4. Applications du théorème de la limite centrale [18]

1. La distribution de l'échantillon est supposée normale lorsque la distribution est inconnue ou n'est pas normalement distribuée selon le théorème de la limite centrale. Cette méthode suppose que la population donnée est distribuée normalement. Il aide à l'analyse des données.
2. L'écart moyen de l'échantillon diminue à mesure que nous augmentons les échantillons prélevés sur la population, ce qui aide à estimer la moyenne de la population avec plus de précision.
3. La moyenne de l'échantillon est utilisée pour créer une plage de valeurs qui inclut probablement la moyenne de la population.
4. Le concept du théorème de la limite centrale est utilisé dans les sondages électoraux pour estimer le pourcentage de personnes soutenant un candidat particulier comme intervalles de confiance.
5. Le CLT est utilisé pour calculer le revenu familial moyen dans un pays donné.
6. Il est utilisé pour lancer de nombreux dés identiques et impartiaux.
7. La distribution de probabilité pour la distance totale parcourue dans une marche aléatoire se rapprochera d'une distribution normale.
8. Retourner de nombreuses pièces entraînera une distribution normale pour le nombre total de têtes (ou le nombre total équivalent de queues).

9. En examinant la distribution de l'échantillon, CLT peut dire si l'échantillon appartient à une population particulière.
10. Il nous permet de tirer des conclusions sur les paramètres de l'échantillon et de la population et aide à construire de bons modèles d'apprentissage automatique.

2.3.5. Hypothèses du théorème central limite [18]

- ✓ L'échantillon doit être tiré au hasard selon les conditions de randomisation.
- ✓ Les échantillons prélevés doivent être indépendants les uns des autres. Ils ne doivent pas influencer les autres échantillons.
- ✓ Lorsque l'échantillonnage est effectué sans remise, la taille de l'échantillon ne doit pas dépasser 10 % de la population totale.
- ✓ La taille de l'échantillon doit être suffisamment grande.

2.4. Architecture de Box-Muller

2.4.1. La théorie de Box-Muller [22] [23]

En générale c'est une méthode de génération de variables aléatoires normales afin de produire des sorties gaussiennes.

L'algorithme BM a été initialement proposé par Box et Muller [1958] considérant fournir le meilleur compromis entre précision et ressources matérielles. Cette Fournit deux nombres aléatoires gaussiens indépendants en effectuant une transformation sur une paire de nombres aléatoires indépendants et uniformément distribués. L'algorithme est Une brève description suit. Considérons une paire de variables aléatoires indépendantes et uniformément distribuées U1 et U2, elles sont réparties sur l'intervalle (0, 1). La transformation BM fournit deux nouvelles variables indépendantes et gaussiennes G1 et G2 de variance unité

Tel que : $$G1 = \sqrt{-2\ln U1} * \cos(2\pi U2) = f(.) * g1(.) \quad (1)$$

$$G2 = \sqrt{-2\ln U1} * \cos(2\pi U2) = f(.) * g2(.)$$

Où $f(.) = \sqrt{-2\ln U1}$et $g(.) \in \{g1(.), g2(.)\}$ sont des fonctions trigonométriques. Une variante logicielle plus conviviale de l'algorithme BM est la méthode des coordonnées polaires de Marsaglia. [Marsaglia et Bray 1964], en évitant les fonctions trigonométriques par transformation La formule (1) est comme indiqué ci-dessous.

Étant donné x et y, une paire de variables indépendantes et uniformément distribuées, elles sont distribuées sur [−1, +1], soit$s^2 = u^2 + v^2$. Alors, si s ≠ 0 ou s < 1, l'équation (1) peut être écrit comme :

$$G_1 = x * \sqrt{\frac{-2\ln s}{s}} \quad (2)$$

$$G_2 = y * \sqrt{\frac{-2\ln s}{s}}$$

Un avantage évident de l'équation (2) par rapport à l'équation (1) est l'évitement des fonctions trigonométriques. Cependant, en raison de la limitation de s, certains échantillons ($1 - \frac{\pi}{4}$exactement) doivent être jetés. Avant l'introduction de l'algorithme Ziggurat, La méthode des coordonnées polaires de Marsaglia est la méthode la plus populaire pour générer des GRN avec précision dans le logiciel.

2.4.2. AlgorithmedeBox-Muller [24]

Soient U1 et U2 deux variables aléatoires indépendantes uniformément distribuéesdans] 0,1].

Soient :

$$X1 = R\cos(\theta) = \sqrt{-2\ln U1}\cos(2\pi U2)$$
$$X2 = R\sin(\theta) = \sqrt{-2\ln U1}\sin(2\pi U2)$$

$\theta = (2\pi U2)$
$R = \sqrt{-2\ln U1}$

AlorsX1etX2sont indépendantes teste de même loiN(0,1).

2.4.3. Avantages et inconvénients de Box-Muller [23]

La transformée Box-Muller est le générateur GRN (Gaussian Random Number) matériel le plus rapporté dans la littérature publiée. Trois avantages rendent l'algorithme Box-Muller approprié pour l'implémentation matérielle :

1. Aucun stockage requis : la méthode crée deux variables en convertissant directement les deux variables d'entrée. Par conséquent, il n'y a pas d'exigence de données préalables. Contrairement aux méthodes comme les méthodes CLT et Wallace.

Un chemin de données fixe : contrairement à la Ziggurat, il n'y a pas de condition if-el se dans l'algorithme. Cela garantit que GRN est toujours disponible sur chaque cycle d'horloge.

2. Méthode exacte : L'algorithme Box-Muller appartient à la classe exacte de la génération GRN. Cela implique que (en supposant un calcul idéal) une garantie analytique, et puis la distribution gaussienne de la sortie est possible.

L'algorithme BM nécessite le calcul de fonctions élémentaires y compris ln et sqrt ainsi que des fonctions trigonométriques. Aussi, et surtout dans l'implémentation en virgule fixe, la précision de la queue dépend directement de la bande passante en bits de U1 dans l'équation (1).

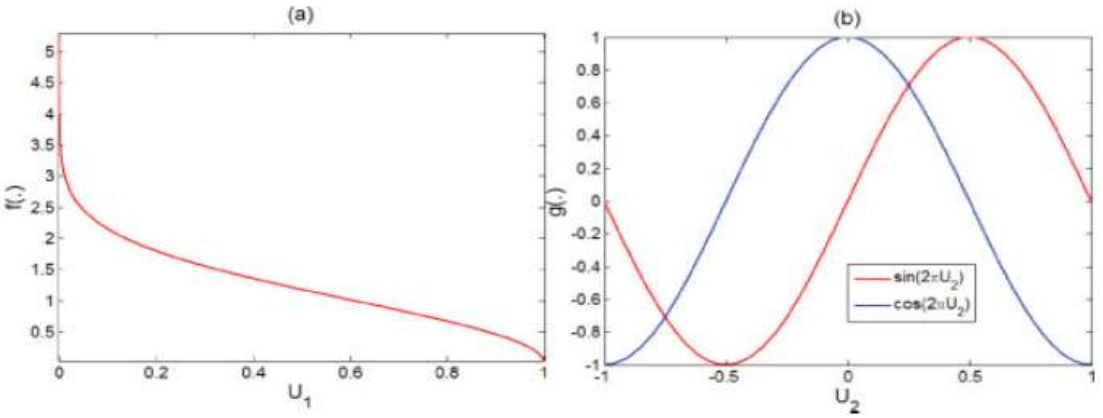

*Figure 2. 2*f(.) et g(.) fonctions dans l'algorithme Box-Muller.**[23]**

La figure 10 montre f(.) et g(.) tracés par rapport à U1 et U2, respectivement. Le principal défi de l'implémentation matérielle est d'explorer l'architecture optimale qui peut calculer ces fonctions. La fonction g(.) a une forme fermée, et g1(.) et g2(.) sont symétriques sur $\pi/4$ périodes et des versions déphasées l'une de l'autre. Ainsi, Calculez simplement un quart de n'importe lequel d'entre eux, qui peut ensuite être mappé à n'importe quel quadrant de g1(.) et g2(.) en utilisant une simple transformation. En général, La fonction g(.) est implémentée à l'aide d'un ajustement de courbe polynomiale avec recherche Table ou algorithme CORDIC.

La fonction f(.) est beaucoup plus difficile à calculer.

- Premièrement, la fonction est ouverte et tend vers l'infini quand U1 atteint zéro.
- Deuxièmement, la fonction est hautement non linéaire et toute approximation polynomiale régulière est hautement sous-optimale.

En particulier, lorsque U1 devient proche de zéro (ce qui correspond à des régions σ élevées), mémoire d'une taille prohibitive est nécessaire si nous voulons générer des GRN précis avec de longues queues.

Pour modéliser les fonctions f(.) et g(.), ils ont utilisé des LUT et ont pu obtenir une précision de queue de 4σ avec une erreur élevée et une utilisation inefficace des ressources matérielles.

2.4.4. Comparaison des performances GRNG :

Tableau 1 : Comparaison de la performance de GRNG utilisant la méthode BM[25].

	[BDG03]	[Xil02]	[LLVC04]	[ABS05]	[Lee06]	[AFCS08]
FPGA ciblé	Altera Flex 10KE	XilinxVirtex-II XC2V4000-6				
Cellules logiques	437	480	2514	702	1528	534
Blocs RAM	0.5	5	2	5	3	2
MULT 18x18	Inconnu	5	8	1	12	3
%occupation	8	2%	10.9 %	3%	6.6%	2.3%
Fréquence max.	98	245	133	165	233	248
Débit (Mbits/s)	24.5	245	133	165	466	496
σmax	4σ	4.8σ	6.7σ	5σ	8.2σ	6.66σ

2.5. Architecture de la méthode récursive

2.5.1. Définition de la méthode récursive [26] [27]

Une méthode récursive s'appelle avec des valeurs d'entrée plus petites et renvoie le résultat pour l'entrée actuelle en effectuant des opérations de base sur la valeur renvoyée pour l'entrée plus petite. Généralement, si un problème peut être résolu en appliquant des solutions à des versions plus petites du même problème, et que les versions plus petites se réduisent à des instances facilement solubles, alors le problème peut être résolu à l'aide d'une méthode récursive.

- ✓ D'une autre manière, nous pouvons dire que **La récursivité** se produit lorsqu'une méthode s'appelle encore et encore jusqu'à ce qu'elle atteigne une condition d'arrêt spécifiée.

Une méthode récursive est plus utile pour les tâches qui peuvent être définies en termes de sous-tâches similaires.

2.5.2. Format d'une méthode récursive [27]

Chaque méthode récursive consiste en2 les pièces :

A. **Cas de base :** le cas de base est l'endroit où l'appel à la méthode s'arrête, ce qui signifie qu'il n'effectue aucun appel récursif ultérieur.
B. **Cas récursif :** Le cas récursif est l'endroit où la méthode s'appelle encore et encore jusqu'à ce qu'elle atteigne le cas de base.

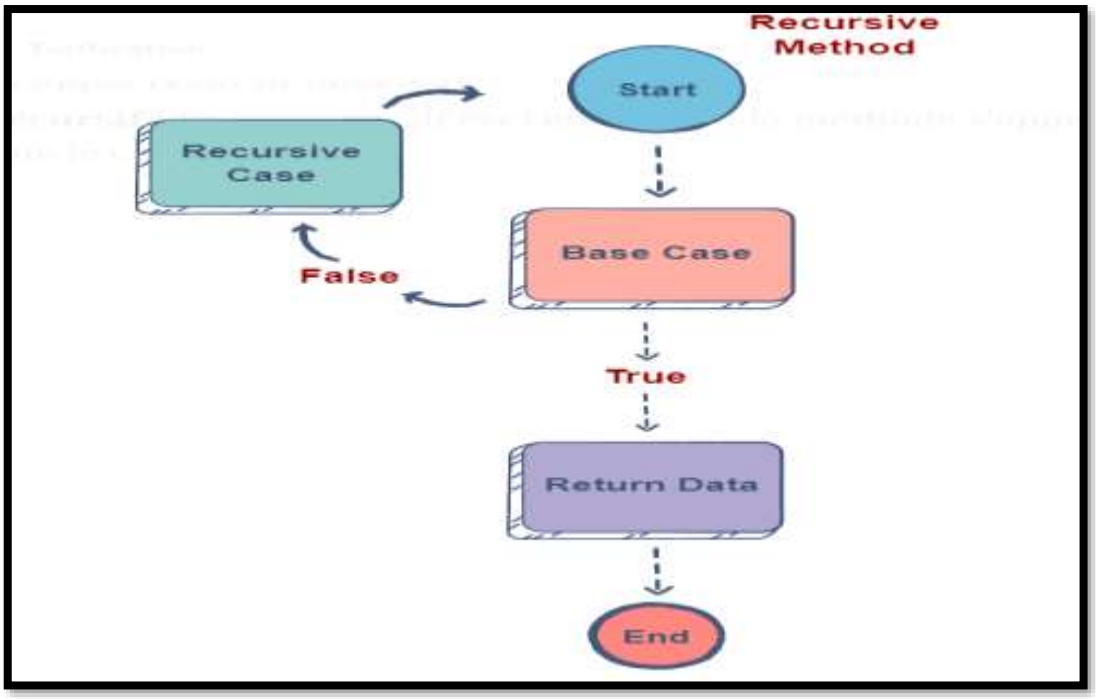

Figure 2.3 schémas détaillés de la méthode récursive.

2.5.3. Caractéristiques des méthodes récursives [28]

- ❖ Doit finir par prendre fin.
  - Une méthode récursive doit avoir au moins un cas de base ou d'arrêt.
  - Un cas de base n'exécute pas un appel récursif - il arrête la récursivité.
- ❖ Chaque appel successif à lui-même doit être une "version plus petite de lui-même" afin qu'un cas de base soit finalement atteint - un argument doit être rendu plus petit à chaque appel afin qu'éventuellement le cas de base s'exécute et arrête la récursivité.

2.5.4. Différents types de récursivité [26]

Il existe quatre types différents d'algorithmes récursifs, vous les examinerez un par un.

- ✓ **Récursivité directe :**

Une fonction est dite récursive directe si elle s'appelle elle-même dans son corps de fonction à plusieurs reprises.

- ✓ **Récursivité indirecte :**

La récursivité dans laquelle la fonction s'appelle elle-même via une autre fonction est appelée récursivité indirecte.

- ✓ **Récursion à queue :**

Une fonction récursive est dite récursive terminale si l'appel récursif est la dernière exécution effectuée par la fonction.

- ✓ **Récursivité sans queue :**

Une fonction récursive est dite récursive sans queue si l'appel de récursivité n'est pas la dernière chose faite par la fonction. Après le retour, il reste quelque chose à évaluer.

2.6. Architecture de la méthode du rejet [29]

Lors de la classification, un classifieur de formes associe normalement une classe à la forme proposée en entrée. Le principe de l'option de rejet est d'enrichir ce fonctionnement en permettant au classifieur de ne pas répondre. Dans ce cas on dit que la forme est rejetée, il y a rejet. Le principal intérêt du rejet est de pouvoir éviter une erreur de classification. Tant que les classifieurs ne seront pas parfaits, le rejet permettra de détecter et de gérer les erreurs potentielles.

2.6.1. La méthode Accepter-Rejeter [30]

C'est une méthode d'échantillonnage classique qui permet d'échantillonner à partir de distributions difficiles ou impossibles à simuler par des transformées inverses. Au lieu de cela, les tirages sont tirés d'une densité instrumentale et acceptés avec une probabilité soigneusement choisie. Le tirage résultant est un tirage de la densité cible.

2.6.2. L'algorithme d'acceptation-Rejet [30]

1. Générer une va Y distribuée comme G.

2. Générer U (indépendant de Y).

3. Si $U \leq \frac{f(Y)}{cg(Y)}$

Puis définissez X = Y (« accepter ») ; sinon revenir à 1 (« rejeter »).

Il faut noter :

- $f(Y)$ Et $g(Y)$ sont va, donc le rapport $f(Y)$ l'est aussi $cg(Y)$ et ce rapport est indépendant de U dans Étape (2).
- Le rapport est borné entre 0 et 1 ; $0 < \frac{f(Y)}{cg(Y)} < 1$.
- Le nombre de fois N que les étapes 1 et 2 doivent être appelées (par exemple, le nombre d'itérations nécessaire pour générer avec succès X) est lui-même un va et a une distribution géométrique avec probabilité de "succès" $p = P(U \leq \frac{f(Y)}{cg(Y)}$; $P(N = n) = (1-p)^{n-1}$, $n \geq 1$. Donc en moyenne le nombre d'itérations nécessaires est donné par $E(N) = 1/p$.
- Au final nous obtenons notre X comme ayant la distribution conditionnelle d'un Y étant donné que l'un événement $\{U \leq \frac{f(Y)}{cg(Y)}\}$ est passe.

Un calcul direct donne que $p = 1/c$, en conditionnant d'abord sur Y :

$$P(U \leq \frac{f(y)}{cg(y)} \mid Y = y) = f(y)/cg(y)$$

Ainsi, déconditionner et rappeler que Y a une densité $g(y)$ donne :

$$p = \int_{-\infty}^{\infty} \frac{f(y)}{cg(y)} \times g(y)dy$$

$$= \frac{1}{C} \int_{-\infty}^{\infty} f(y)dy$$

$= \frac{1}{c}$

Où la dernière égalité suit puisque f est une fonction de densité (donc par définitions intégrations à 1). Ainsi $E(N) = c$, la constante de borne, et nous pouvons maintenant voire en effet qu'il est souhaitable de choisir notre densité alternative g de manière à minimiser cette constante $c = suppx\{f(x)/g(x)\}$. De bien sûr la fonction optimale serait $g(x) = f(x)$ ce qui n'est pas ce que nous avons en tête puisque le tout l'intérêt est de choisir une alternative différente (facile à simuler) de f. Bref, c'est un peu d'un art de trouver un g. En tout cas, on résume avec :

Le nombre attendu d'itérations de l'algorithme requis jusqu'à ce qu'un X soit réussi générée est exactement la constante de délimitation $c = suppx\{f(x)/g(x)\}$

2.6.3. Applications de la méthode rejet [31]

- Distribution normale : Si nous désirons un X ∼ N (μ, σ2), alors nous pouvons l'exprimer sous la forme $X = \sigma Z + \mu$, où Z désigne une va avec la distribution
  - N (0, 1). Il suffit donc de trouver un algorithme pour générer Z ∼ N (0, 1).

De plus, si nous pouvons générer à partir de la valeur absolue, |Z|, alors par symétrie nous pouvons obtenir notre Z en générant indépendamment une va S (pour le signe) qui est de ±1 avec une probabilité de 1/2 et en définissant Z = S|Z|. En d'autres termes, nous générons un U et posons Z = |Z| si U ≤ 0,5 et poser Z = −|Z| si U > 0,5. |Z| est non négatif et a une densité

$f(x) = \frac{2}{\sqrt{2\pi}} e^{-x^2/2}$; $x \geq 0$.

Pour notre alternative nous choisirons $g(x) = e^{-x}$, x ≥ 0, la densité exponentielle avec le taux

1, quelque chose que nous savons déjà simuler facilement (méthode de transformation inverse, par Exemple). Notant que h(x) déf = $f(x)/g(x) = e^{x-x^2/2}\sqrt{2/\pi}$, nous utilisons simplement le calcul différentiel pour calculer son maximale (résoudre h´(x) = 0) ; qui doit se produire à cette valeur de x qui maximise l'exposant x = $x^2/2$; à savoir à la valeur x = 1. Ainsi

$c = \sqrt{2e/\pi} \approx 1.32$, et donc $\frac{f(y)}{cg(y)} = e^{-(y-1)^2/2}$.

- Répartition bêta : En général, une distribution bêta sur l'intervalle unitaire, x ∈ (0, 1), a une densité de la forme $f(x) = bx^n(1-x)^m$avec n et m non négatifs (entiers ou non). La constante b est la constante de normalisation,

$$b = \left[\int_0^1 x^n (1-x)^m dx\right]^{-1}.$$

(De telles distributions généralisent la distribution uniforme et sont utiles pour modéliser proportions.)

Considérons un cas particulier: $f(x) = bx^n(1-x)^n = b\big(x(1-x)\big)^n$. Comme l'uniforme distribution sur (0, 1), celle-ci a une moyenne de 1/2, mais sa masse est plus concentrée vers 1/2 que vers 0 ou 1 ; il a une plus petite variance que l'uniforme. Si nous utilisons g(x) = 1, x ∈ (0, 1) (l'uniforme densité), alors f(x)/g(x) = f(x) et comme cela se vérifie facilement,

$c = sux\ f(x)$ Est atteint à x = 1/2 ; c = $b(1/4)^n$.

2.6.4. Limitations et défis de l'échantillonnage de rejet [32]

- Sélection de la fonction de proposition appropriée et recherche de sa constante d'échelle.
- Nécessite que le PDF de la fonction cible soit connu.
- Généralement inefficace surtout dans les dimensions supérieures.

2.7. Conclusion :

Dans ce chapitre nous avons présenté comment peut-on générer les nombres aléatoires gaussiens et c'est par notre explication des quatre méthodes : la méthode de la limite centrale, la méthode Box-Muller, la méthode Récursive et la méthode du rejet.

La meilleure méthode de génération et la plus rapide à simuler c'est la méthode du rejet.

Chapitre 3 Etude de la méthode de Box-Muller à découpage d'intervalle.

2.1.Introduction :

Dans ce chapitre nous présenterons la méthode de Box Muller est l'une des méthodes de génération de nombres aléatoires gaussiens les plus utilisées. L'algorithme Box-Muller peut être utilisé pour convertir deux ensembles de nombres aléatoires avec des distributions uniformes en deux ensembles de nombres aléatoires avec des distributions gaussiennes. Cependant, un procédé optimisé pour la simulation d'un système de communication informatique ne peut pas générer de manière cohérente une utilisation efficace des ressources matérielles dans les circuits intégrés spécifiques à l'application (ASIC) et les dispositifs à matrice de portes programmables (FPGA).

Nous allons faire tout d'abord la présentation de la méthode de BM, ensuite sur son fonctionnement logicielle MATLAB et son fonctionnement logicielle FPGA.

Nous pouvons réduire la taille de mémoire requise tout en conservant la précision conventionnelle.

2.2. Principe de la méthode proposée

3.2.1. Définition de transformation de Box-Muller proposée [33]

La méthode proposée à précision fixe est basée sur une méthode de transformation exacte. Comparable à l'approche classique de Box-Muller, les échantillons GRN sont générés à partir de couples de nombres aléatoires uniformes indépendants (U1, U2), sur l'intervalle unitaire]0, 1]. Néanmoins, U1 doit être généré avec une double précision de U2. Si n-bits sont utilisés pour coder U2, 2xn-bits sont utilisés pour coder U1. L'algorithme 2 fournit le code principal de la méthode proposée. La figure12 montre le tracé associé de la nouvelle fonction de densité de probabilité gaussienne.

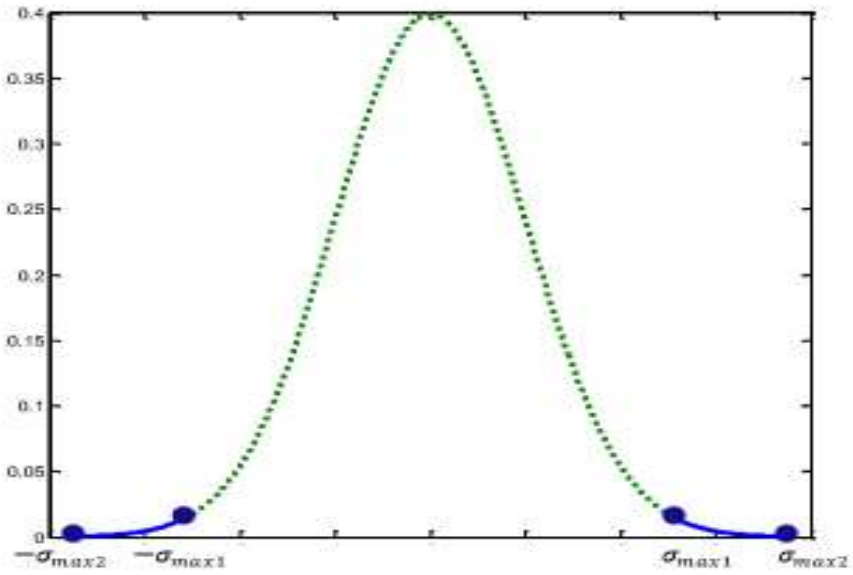

Figure 3.1La nouvelle fonction de densité de probabilité gaussienne. **[33]**

3.2.2. L'algorithme de Box Muller [34]

La méthode Box-Muller (BM) génère à partir de deux variables aléatoires uniformes et indépendantes U1, U2, deux variables aléatoires V1, V2. Les opérations réalisées sur U1 et U2 sont les suivantes :

$f(U1) = \sqrt{-2\ln U1}$ (1)

$g0(U2) = \sin(2\pi U2)$ (2)

$g1(U2) = \cos(2\pi U2)$ (3)

$V0 = f(U1) * g0(U2)$ (4)

$V1 = f(U1) * g2(U2)$ (5)

3.2.3. L'architecture basée sur FPGA Box-Muller proposée [33]

La structure GRNG proposée basée sur le FPGA Box-Muller, avec (N=2), est illustrée à la Figure2. Il comprend une unité basée sur LFSR, un sin/cos Block RAM, de $(0, \sigma_{max1})$ f fonction Block RAM, de $(\sigma_{max1}, \sigma_{max2})$, f fonction Block RAM, un multiplexeur et un multiplicateur.

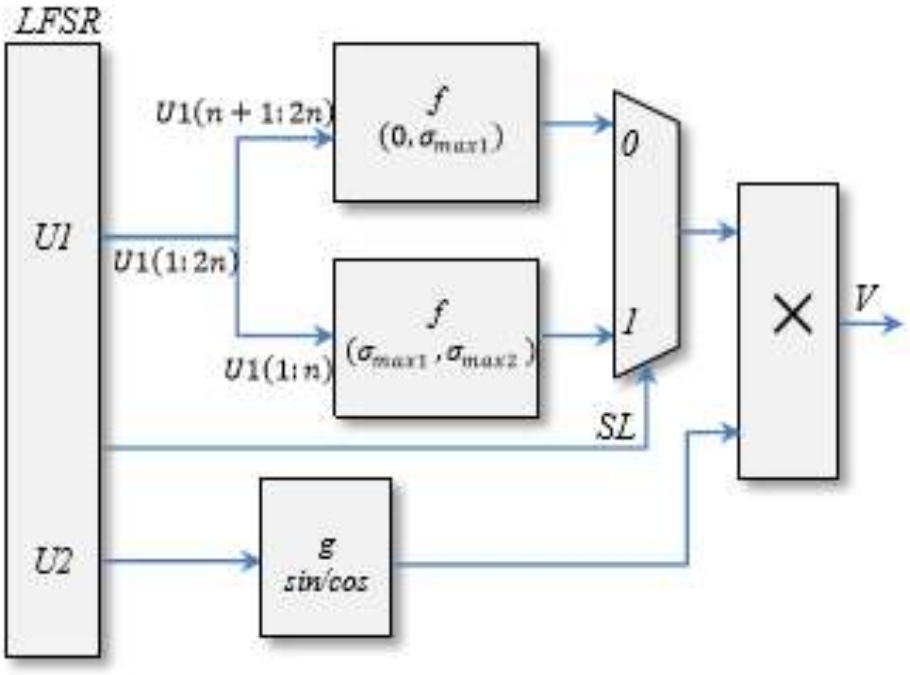

Figure 3 2La structure principale du Box-Muller proposé pour la mise en œuvre de FPGA. **[33]**

- Le générateur de paires de nombres aléatoires uniformes (U1, U2) est implémenté à l'aide d'une unité basée sur LFSR. Le sélecteur de multiplexeur SL est également contrôlé par une unité basée sur LFSR. Cette sélection est l'état de la valeur de test U1.

2.3.Fonctionnement logicielle de la méthode proposée

3.3.1Travail sur Matlab

Pour élaborer un générateur de bruit gaussien nous avons utilisé la méthode de box-Muller dans MATLAB en suivant les étapes suivantes :

- En premier, nous avons calculer la variation fixe sigma(σ), et cela se fait en soustrayant les abscisses de la fonction f (la différence des échantillons).
- En deuxième, nous avons calculer et fixé le nombre de bits de bus d'adresse, en calculant la relation suivante : $log_2(\max/\min diff) = 12\ bits$.
- En troisième, nous avons déterminer et fixé le nombre de bits de bus data (bus de donnée).
- En quatrième, nous avons rajouter deux intervalles (L2 et L3) après le premier intervalle (L1), ceci exprime que la fonction f est définiesur ces trois intervalles : L1, L2 et L3

L1= 0 → 4095

L2= L1*L1

L 3= L1*L1*L1

- En cinquième, nous avons faire la conversion des deux fonctions f et g, en premier du décimal vers → binaire, puis l'inverse binaire vers→ décimal, utilisant la virgule flottante et la virgule fixe.

- En sixième, après la transformation des résultats Matlab des deux fonctions f et g (décimal vers binaire), nous avons transférer ces résultats en fichiers textes pour l'utiliser dans VHDL.

- Il faut que toutes les valeurs binaires de $f(x1) et\ g(x2)$seront codés par des codes non nuls.

Virgule fixe et virgule flottant :

- Nous avons déterminé le maximum de $f(1,i) = sqrt(-2 * \log(x1))$, et $f1$ c'est une virgule flottante.
- Puis nous avons faire la conversion en binaire du maximum de $f(1,i)$.
- Ensuite nous avons le codé sur 12bits et le nombre (N) représente le reste des nombres binaires totales (12bits) c'est à dire N=9 bits.
- Aussi, nous avons déterminé $Fs1$ c'est le vecteur de virgule fixe $fs1 = f1 * 2^N$.
- La valeur binaire du vecteur $fs1$envirgulefixe est$fb1 = dec2bin(fs1,12)$.

Création du ficher ROM :

- Nous avons créé deux fichiers textes le premier pour les codes décimaux et les deuxièmes pour les codes binaires.
- Aussi, nous avons créé des boucles « for » pour F1, F2 et F3.
- Ensuite, nous avons faire la conversion de ces valeurs en binaire.
- Et à la fin nous avons stocké les valeurs obtenues vers des fichiers textes.

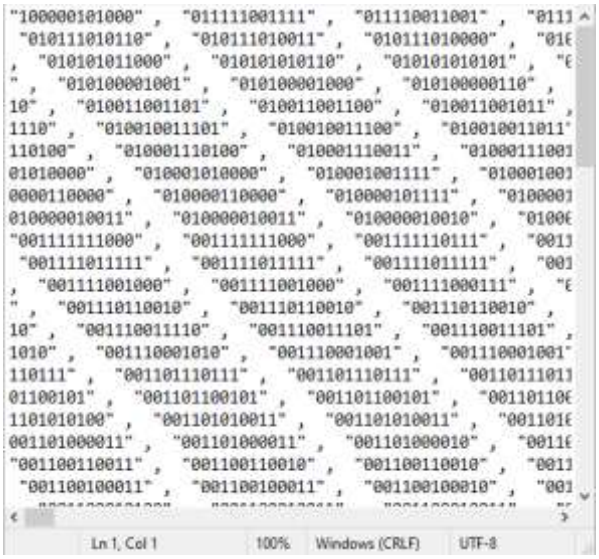

Figure 3. 3 Ficher texte en binaire.

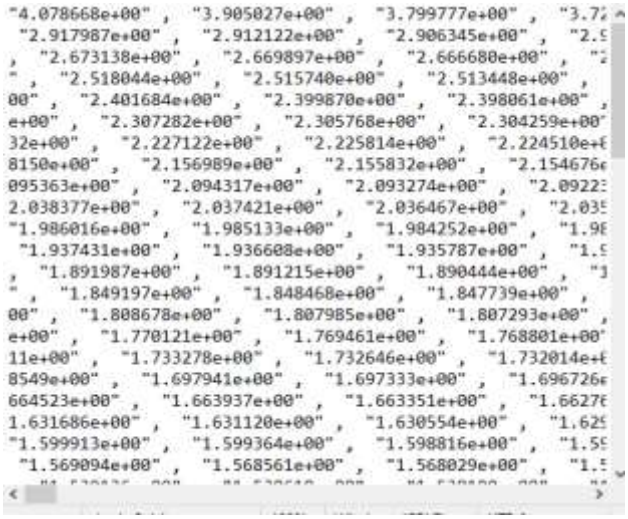

Figure 3.4Fichier texte en décimale.

2.4.Fonctionnement matérielle (FPGA) de la méthode proposée

3.4.1. Logiciel Xlilinx ISE14.7[34]

Xilinx ISE est un logiciel de conception matérielle de renommée mondiale, il fournit un outil intuitif d'amélioration de la productivité pour chaque étape du processus de conception, couvrant tous les processus de conception, de l'exploration de la conception au niveau du système, du développement logiciel et de la conception matérielle basée sur HDL à la vérification, au débogage et au PCB. Intégration de la conception.

Xilinx ISE fonctionne très rapidement et les concepteurs peuvent effectuer plusieurs itérations de conception en une journée. Cet environnement de conception amélioré fournit désormais également la technologie smartxplorer. La technologie Smartxplorer est spécifiquement développée pour relever les deux défis de taille auxquels sont confrontés les concepteurs : la convergence temporelle et la productivité. La technologie Smartxplorer prend en charge le traitement distribué sur plusieurs hôtes Linux, ce qui permet d'effectuer davantage de processus de mise en œuvre en une journée. En utilisant le traitement distribué et plusieurs stratégies de mise en œuvre, les performances peuvent être améliorées jusqu'à 38 %. La technologie Smartxplorer fournit également des outils qui permettent aux utilisateurs de surveiller chaque instance en cours d'exécution avec des rapports de synchronisation indépendants.

2.4.1.1.Fonctionnalités logicielles [34]

1. Utiliser la technologie de déclenchement d'horloge automatique pour réduire la consommation d'énergie dynamique jusqu'à 30 %.
2. Utiliser le processus de conception de reconfiguration partielle de quatrième génération pour réduire le coût du système.
3. Planahead - Nouveau processus de conception RTL vers bitstream pour les concepteurs de logique.
4. Utilisez l'interface axi4 pour réaliser la conception FPGA plug and play.

3.4.2. Les fonctions f et g sont des ROM Développées par VHDL

C. Block ROM $f1$:

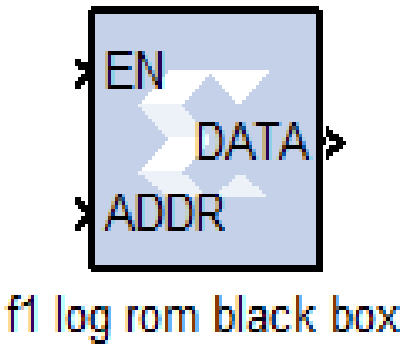

Figure 3.5 Bloc ROM $f1$.

Contient :

Synthèse VHDL

- 5 ports : (4entréet 1sortie).
- 4 entrées : Clk, EN, bused'adresse2^12bits.
- Sortie :DATA 12 bits.

Rom **f1** : = ("100000101000" , "011111001111" , "011110011001" ,…..)

Bus d'adresse K : Qui n

rdata<= ROM(conv_integer(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 ;

Syntaxe et simulation Dans le Système Generator.

D. Block ROM $f2$:

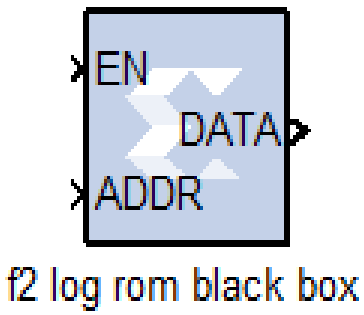

Figure 3.6Block ROM $f2$.

Contient :

Synthèse VHDL

- 5 ports : (4entréet 1sortie).
- 4 entrées : Clk, EN, bused'adresse2^12bits.
- Sortie :DATA 12 bits.

Rom $\boldsymbol{f2}$: = ("101110001001" , "101101001011" , "101100100110" ,…..) ;

Buse d'adresse K : Qui n

rdata<= ROM(conv_integer(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 ;

Syntaxe et simulation Dans le Système Generator.

E. Block ROM $f3$:

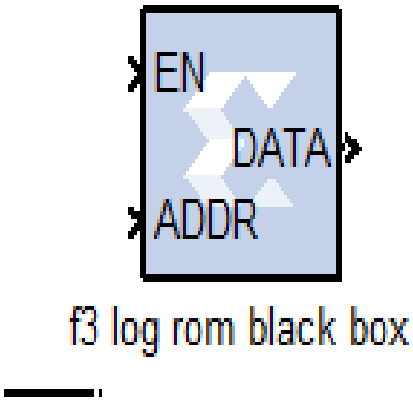

Figure 3.7Bloc ROM $f3$.

Contient :

Synthèse VHDL

- 5 ports : (4 entré et 1 sortie).
- 4 entrées : Clk, EN, buse d'adresse 2^12bits.
- Sortie : DATA 12 bits.

Rom $\boldsymbol{f3}$: = ("111000100001" , "110111101110" , "110111010000" ,…..) ;

Buse d'adresse K : Qui n

rdata<= ROM(conv_integer(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 ;

Syntaxe et simulation Dans le Système Generator.

F. Block ROM *g* :

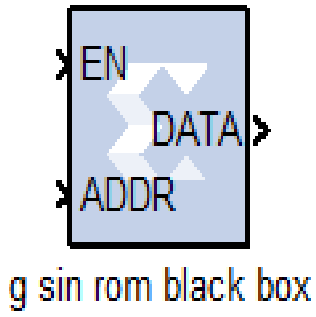

Figure 3.8Bloc ROM g.

Contient :

Synthèse VHDL

- 5 ports : (4 entré et 1 sortie).
- 4 entrées : Clk, EN, buse d'adresse 2^12bits.
- Sortie : DATA 12 bits.

Rom *g* : = ("000000000000", "000000000001", "000000000010",…..) ;

Buse d'adresse K : Qui n

rdata<= ROM(conv_integer(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 ;

Syntaxe et simulation Dans le Système Generator.

3.4.3. Définition de LFSR [35]

Le registre à décalage de rétroaction se compose de deux parties : un registre à décalage et une fonction de rétroaction (voir Fig20). Le registre à décalage est une séquence de bits. Sa longueur est déterminée en bits ; s'il a une longueur de n bits, il est appelé registre à décalage de n bits. Tous les bits du registre à décalage sont décalés d'un bit vers la droite chaque fois qu'un bit est nécessaire. Le nouveau bit le plus à gauche est calculé en fonction des autres bits du registre. La fonction de rétroaction est normalement le XOR des bits sélectionnés dans le registre ; la liste de ces bits est appelée une séquence de prises. La sortie du registre à décalage est un bit, généralement le bit le moins significatif. Période, complexité linéaire et mesures statistiques du flux de clés.

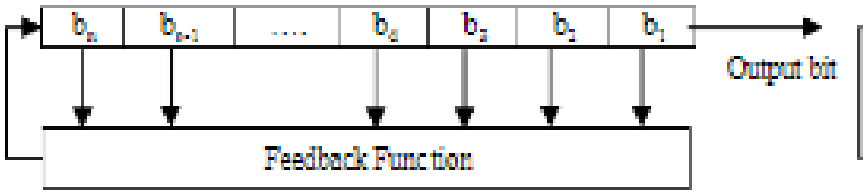

Figure 3.9Les constructions générales d'un registre à décalage à rétroaction linéaire. **[35]**

Les LFSRs sont couramment utilisés dans le cadre des générateurs de flux de clés dansles chiffrements de flux. Certains critères sont pris en compte pour les parties des générateurs de flux de clés. Ces critères incluent la période, la complexité linéaire et les mesures statistiques des flux clés.

- **Période [36]**

La période d'un registre à décalage est la longueur de la séquence de sortie avant qu'elle ne commence à se répéter. Si le polynôme de rétroaction du LFSR à n bits est primitif et que son état initial est à un état non nul, alors la séquence de sortie générée par ce LFSR a la période maximale de 2n -1. Cette séquence est appelée séquence de longueur maximale ou séquence m. Les séquences m possèdent d'excellentes propriétés de caractère aléatoire.

- **Complexité linéaire [37]**

Une métrique essentielle utilisée pour évaluer les générateurs basés sur LFSR est la complexité linéaire, ou étendue linéaire. Ceci est décrit comme la longueur (n) du LFSR le plus court qui peut imiter la sortie du générateur. La complexité linéaire est très importante, car un algorithme simple, appelé l'algorithme de Berlekamp-Massey, peut générer ce LFSR après avoir étudié seulement 2n bits du flux de clé. Une fois ce LFSR généré, le chiffrement de flux est rompu. Il est à noter qu'une grande complexité linéaire n'indique pas toujours un générateur sûr. Cependant, une petite complexité linéaire indique une insécurité.

- **Mesures statistiques [38] [39] [40]**

Des mesures appropriées sont nécessaires pour examiner le degré de caractère aléatoire des séquences binaires générées par des générateurs de nombres aléatoires. Un certain nombre de tests statistiques existent pour déterminer le comportement statistique de la séquence. Ces tests vérifient généralement la distribution aléatoire, la distribution des uns et des zéros dans une séquence, la dépendance linéaire entre les sous-chaînes de longueur fixe, le niveau de compression qui peut être effectué sur la séquence testée et si une séquence est suffisamment complexe pour être considérée comme aléatoire. Trois tests bien connus sont les tests Federal Information Processing Standard (FIPS), Diehard suite et National Institute of Standards and TechnologyStatistical Test Suite (NIST).

3.4.4. Fonctionnement de LFSR [41] [42]

Un LFSR est un dispositif dérivé du registre à décalage de type SIPO (Serial In - Parallel Out), dans lequel un ou plusieurs « étages » du registre subissent une transformation pour être réinjectés en entrée de celui-ci.
Il est dit de longueur « r » lorsqu'il est composé de r éléments appelés « étages » ou « cellules », le contenu de l'ensemble de ces éléments à un moment « t » est l'état du LFSR à ce moment. À chaque top d'horloge le contenu d'un étage est transféré au suivant et le premier est rempli par le résultat d'une fonction linéaire qui prend en compte l'état d'un ou de plusieurs étages.

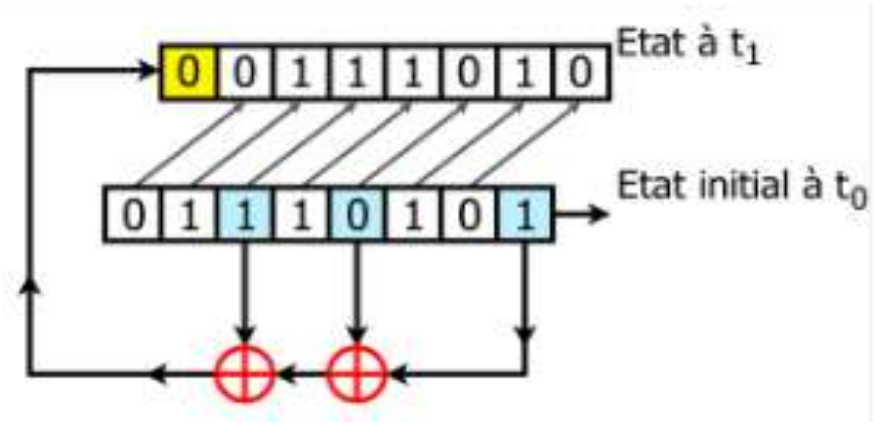

Figure 3.10 LFSR à 8bits. ***[42]***

G. Black Box de LFSR de log :

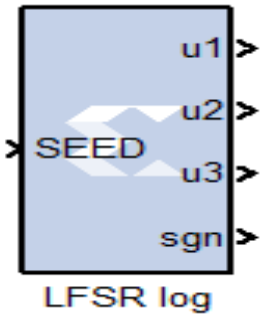

Figure 3.11 Bloc LFSR log.

Contient :

Synthèse VHDL

- 7 ports (3 entré et 4sorties).
- 3 entrées : Clk, CE, SEED.
- Sortie : u1→ 12bits, u2 → 12 bits, u3 →12 bits, Sgn→ 2bits.

```
Process (C)
begin
if (C'event and C='1') then
if SEED ='1' then
tmp<= x"5FAF5FAF5FAAF5FAF5FAF5FAAF5FAF5FAF5FAAF5FAF5AF5";
else
if CE='1' then
for i in 0 to 180 loop -- 1 to 181
tmp(i) <= tmp(i+1);
Endloop;
tmp(187) <= tmp(0); --188
tmp(186) <= tmp(187); --187
tmp(185) <= tmp(0) xortmp(186); --186
tmp(184) <= tmp(185); -- 185
(183) <= tmp(184); -- 184
tmp(182) <= tmp(0) xortmp(183); -- 183
tmp(181) <= tmp(0) xortmp(182); -- 182
End if;
```

```
End if;
End if;
End process;
Process (tmp)
begin
if  tmp(122 downto 89)>16777216 then SL <= "00";
elsiftmp(122 downto 89)>4096 then SL <="01";
else SL<="10";
end if ;
end process;
u1<= tmp(122 downto 111);
u2<= tmp(110 downto 99);
u3<= tmp(98 downto 87);
sgn<= SL;
```

H. Black Box de LFSR de sin :

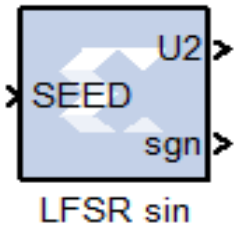

Figure 3.12Bloc LFSR sin .

Contient :

Synthèse VHDL

- 7 ports : (3 entré et 2 sorties).
- 3 entrées : Clk, CE, SEED.
- Sortie : U2→ 12 bits, Sgn →2bits.

```
Process (C)
begin
if (C'event and C='1') then
if SEED ='1' then
```

```
tmp<= x"5FAF5FAF5FAAF5FAF5FAF5FAAF5FAF5FAF5FAAF5FAF5AF5";
else
if CE='1' then
for i in 0 to 180 loop -- 1 to 181
tmp(i) <= tmp(i+1);
End loop;
tmp(187) <= tmp(0); --188
tmp(186) <= tmp(187); --187
tmp(185) <= tmp(0) xortmp(186); --186
tmp(184) <= tmp(185); -- 185
 (183) <= tmp(184); -- 184
tmp(182) <= tmp(0) xortmp(183); -- 183
tmp(181) <= tmp(0) xortmp(182); -- 182
End if;
End if;
End if;
End process;
U2 <= tmp(90 downto 79)  ; -- out: 12 bits
sgn<= tmp(80);
```

3.4. Conclusion :

Dans ce chapitre nous avons bien détaillé le fonctionnement de la méthode de Box-Muller dans le logicielle MATLAB et FPGA.

La méthode de Box-Muller proposée est plus rapide que la précédente, avec une utilisation optimisée de la logique FPGA du réseau de portes programmables sur le terrain

Chapitre 4 Implémentation et résultats

4.1. Introduction

Dans ce chapitre nous présenterons les deux implémentations sur MATLAB et FPGA. Nous allons faire tout d'abord l'implémentation sur MATLAB et faire tester les résultats obtenus de la méthode de Box-Muller. Ensuite nous allons implémenter sur FPGA et comparer entre le software et le hardware.

4.2. Implémentation et test sur MatLab

Nous avons découpé l'intervalle total sur trois intervalles : L1=4096, L2=L1*L1 et L3=L1*L1*L1.

Pour chaque intervalle une fonction spécialisée :

$F1 = \sqrt{-2\log(x1)}$Pourl'intervalle L1.

$F2 = \sqrt{-2\log(x2)}$ Pour l'intervalle L2.

$F3 = \sqrt{-2\log(x3)}$ Pour l'intervalle L3.

Ensuite chaque fonction on la met dans une boucle for.

A la fin nous avons déterminé le vecteur V (signal de sortie), $V = f * g * sgn$

Nous avons faire la multiplication fois 'sgn' pour avoir la partie négative de signal.

A. Autocorrélation :

```
Ycor=(1/N)*xcorr(V1);
figure(1)
plot(Ycor);
```

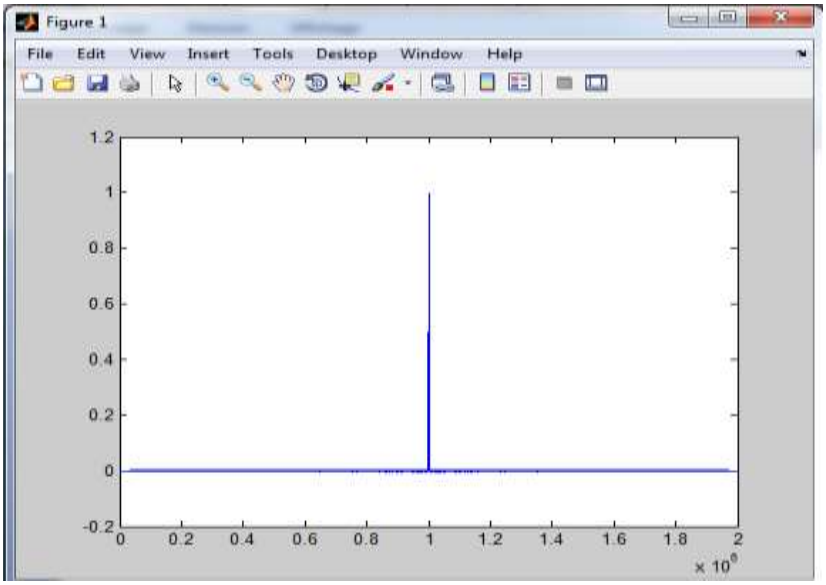

Figure 4. 1Autocorrélation de la sotie V1.

B. **Spectrale :**

```
fftSignal = fft(V1);
figure (2)
plot(abs(fftSignal));
```

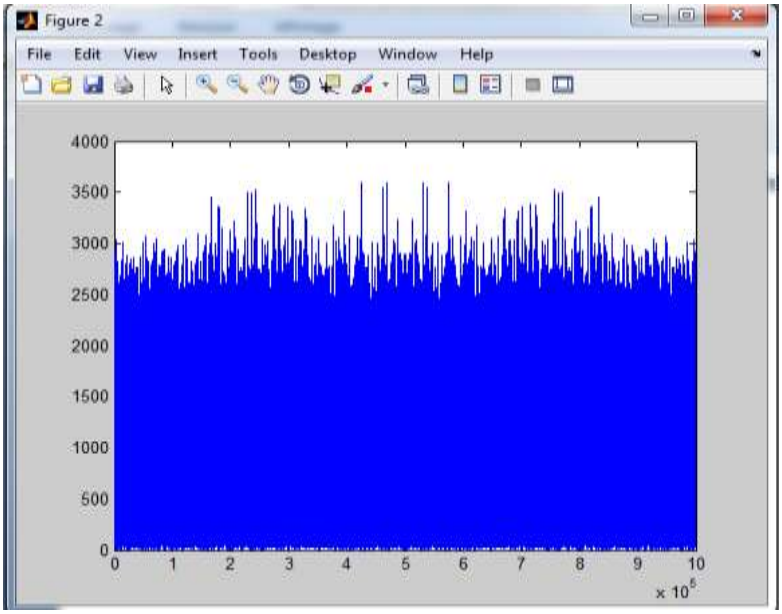

Figure 4. 2 Spectrale de la sotie V1.

C. Distribution normale

```
[f, x] = hist(V1, 100); % Create histogram from a normal distribution.
 g = 1 / sqrt(2 * pi) * exp(-0.5 * x .^ 2); % pdf of the normal distribution
figure (3)
bar (x, f / trapz(x, f)); hold on
plot (x, g, 'r'); hold off
```

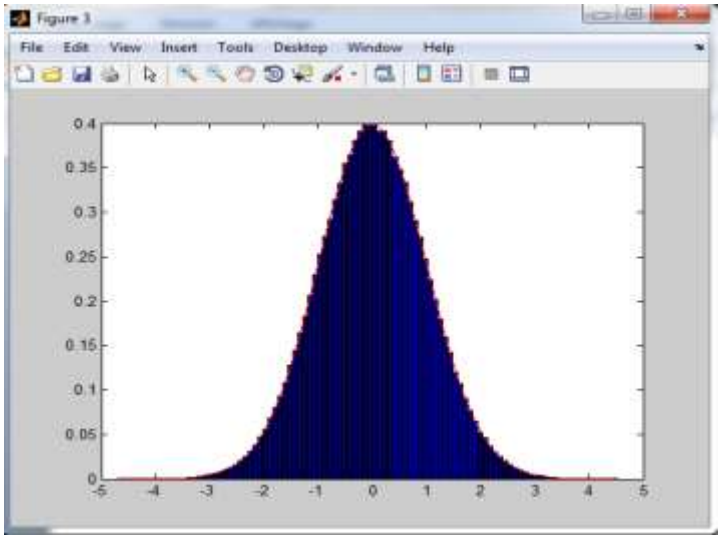

Figure 4.3La distribution normale de la sotie V1

D. Scatter plot V1

```
fori=1:(N-1)
      V1s(1,i)=V1(1,(i+1));
end
   V1s(1,N)=V1(1,1);
Figure (4)
   scatter(V1s,V1,6,'.')  %scatter plot V1
```

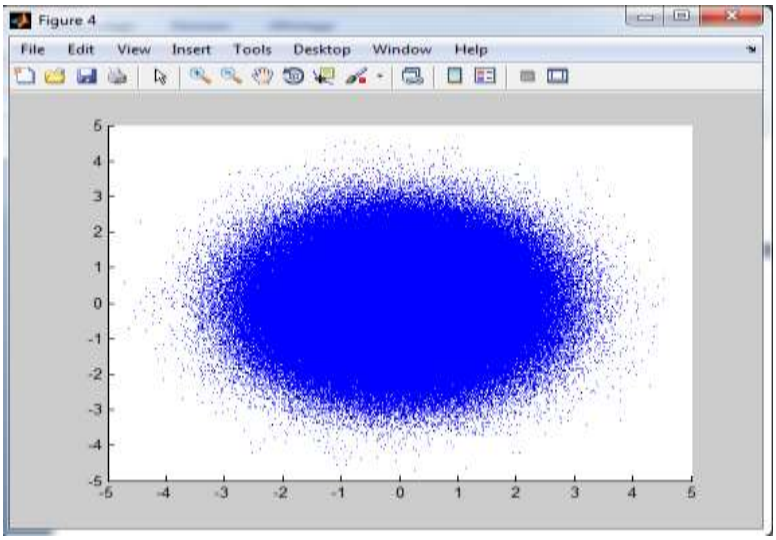

Figure 4.4scatter plot de la sotie V1.

E. Tail:

bar (x, (f/L1) / trapz(x, f)); hold on

- réduire l'intervalle de f.

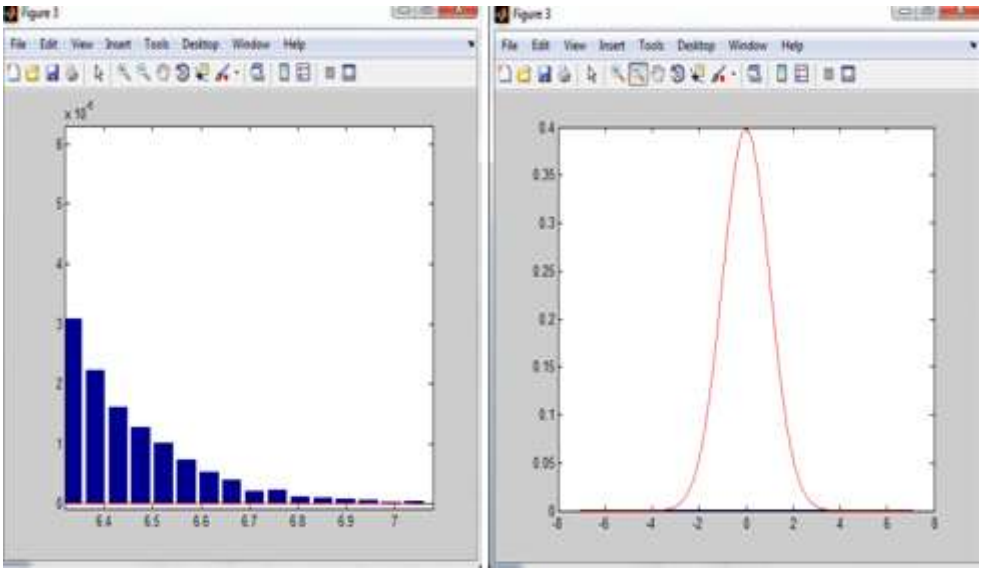

Figure 4. 5tail de la sortie V1.

4.3. Implémentation *sur FPGA*

4.3.1. Bloc Gatay In [44]

Ces blocs convertissent les types de données entier, double et à virgule fixe Simulink® en type à virgule fixe Model Composer.

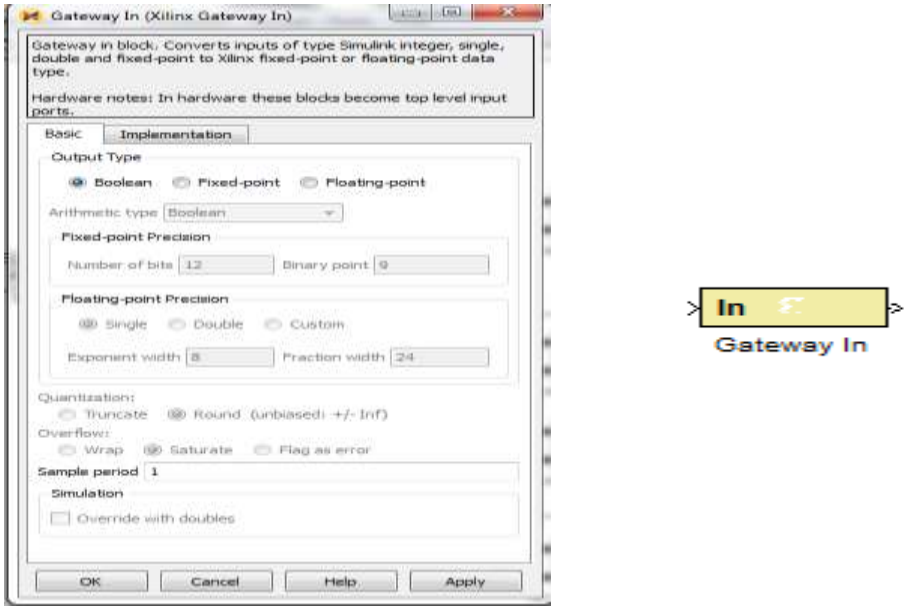

Figure 4.6 Bloc Gatay In.

4.3.2. Bloc Gatay Out [44]

Ce bloc convertit le type de données à virgule fixe ou à virgule flottante Model Composer en un type de données Simulink entier, simple, double ou à virgule fixe.

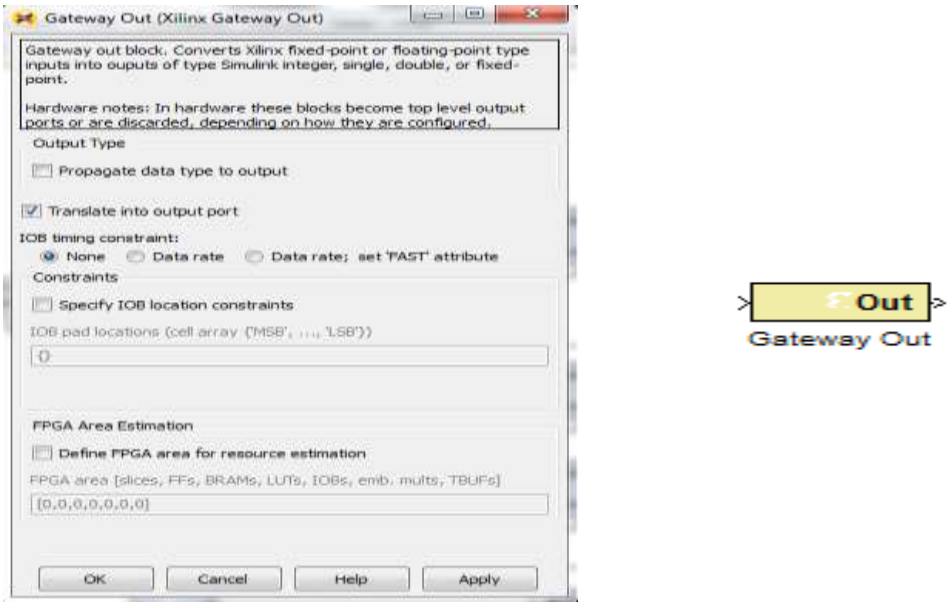

Figure 4.7 Bloc Gatay Out.

4.3.3. Bloc Reinterpret [44]

Le bloc Xilinx Reinterpret force sa sortie à un nouveau type sans tenir compte de la conservation de la valeur numérique représentée par l'entrée.

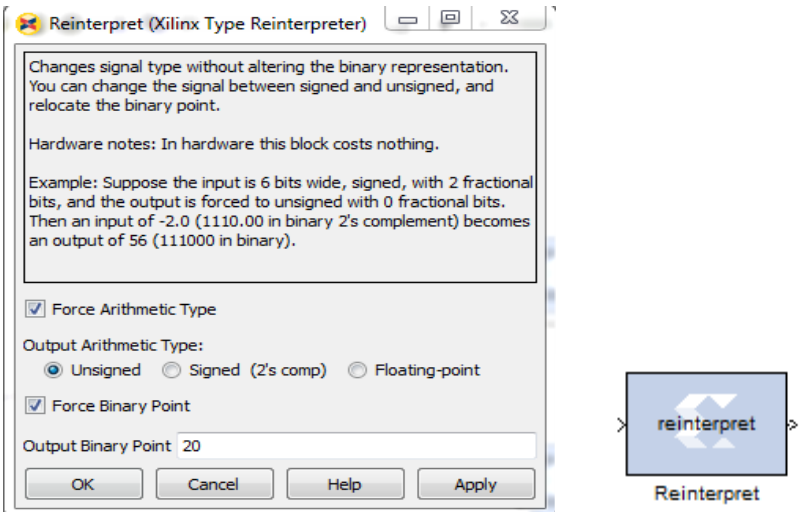

Figure 4.8Bloc Reinterpret.

4.3.4. Simulation d'un générateur de bruit Gaussienne avec SG :

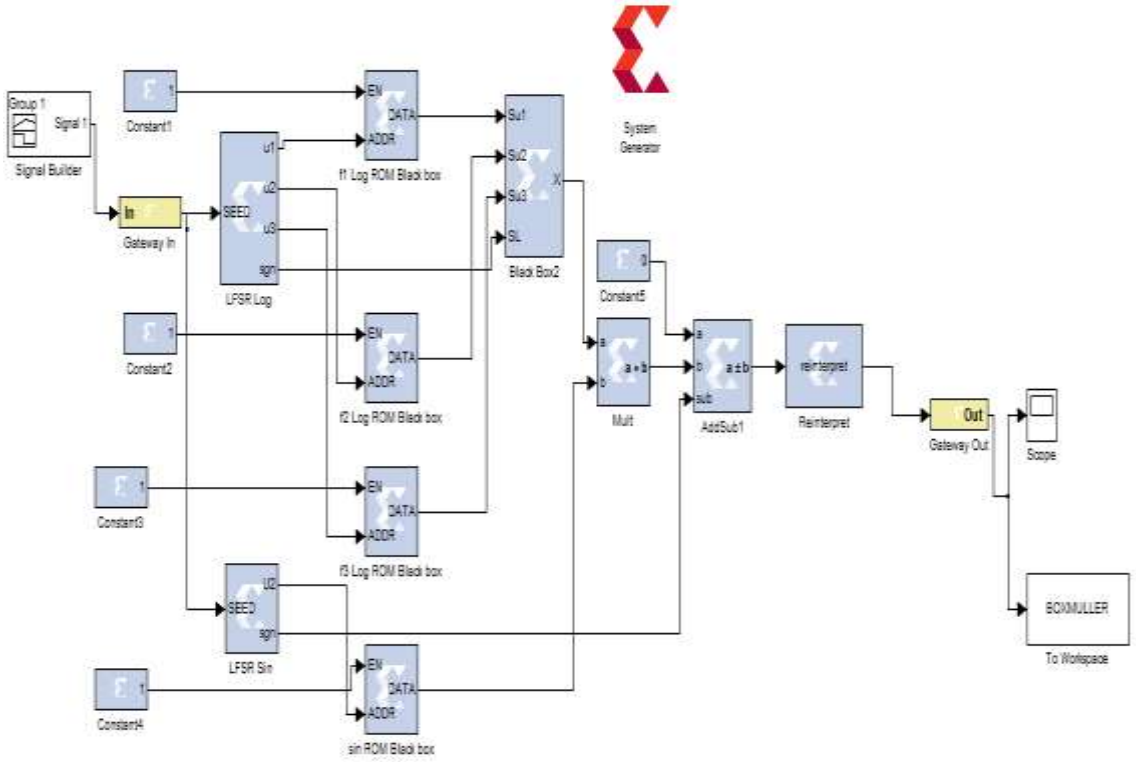

Figure 4.9Simulation d'un générateur de bruit Gaussien par la méthode de Box-Muller.

4.4. Simulations et résultats :

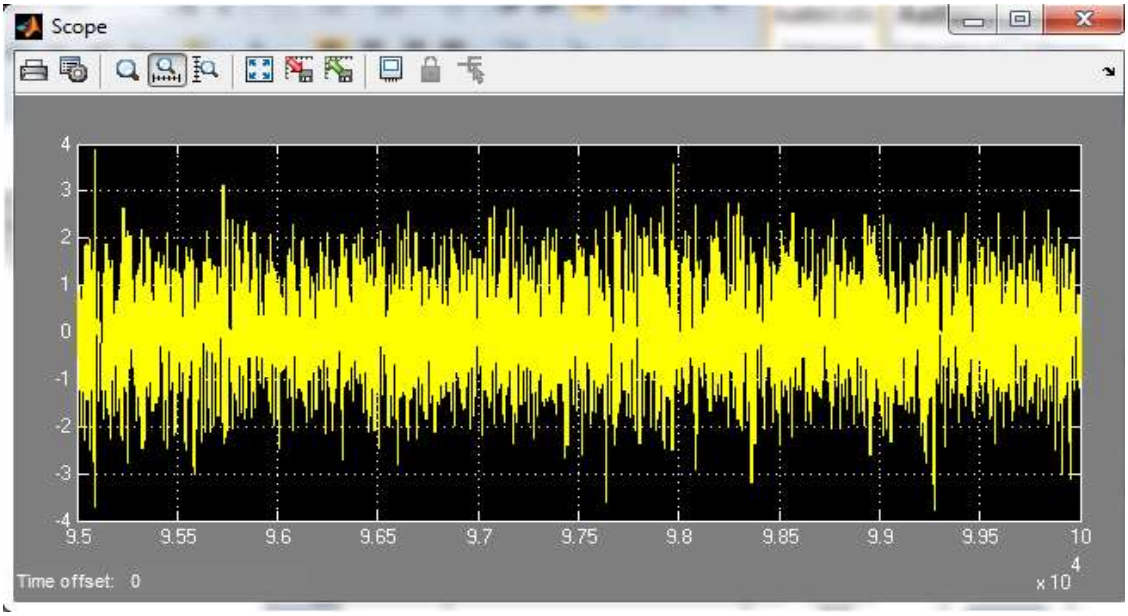

Figure 4.10 Signal de sortie de Box-Muller.

Autocorrélation :

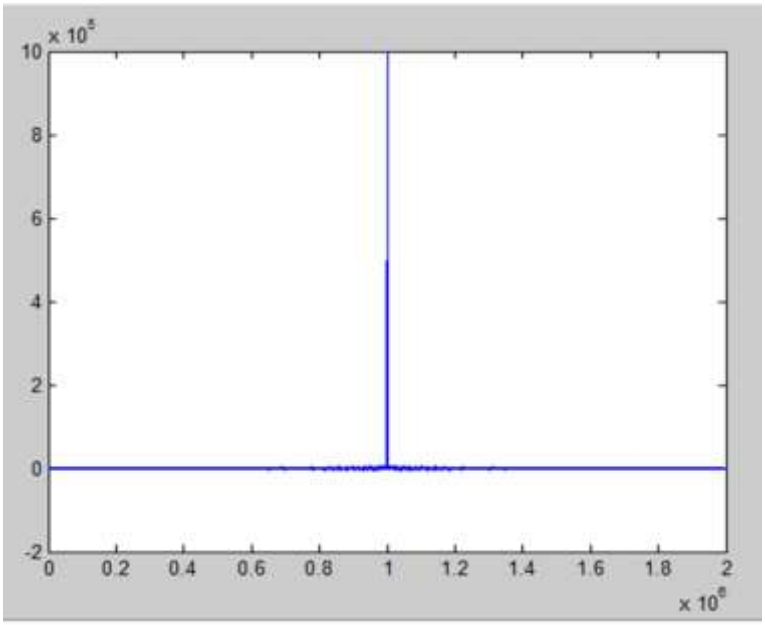

Figure 4. 11Autocorrélation de BM.

Spectrale :

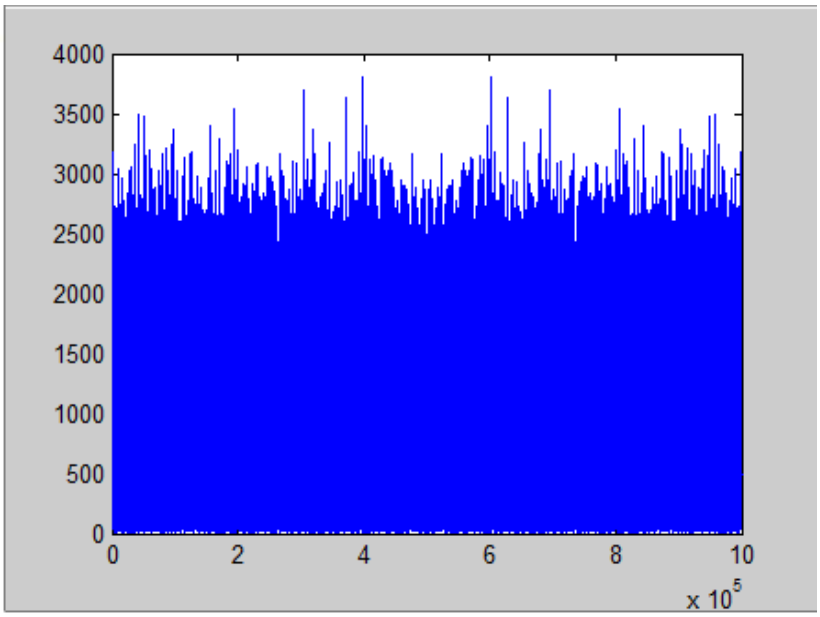

Figure 4.12Spectrale de BM.

La distribution normale

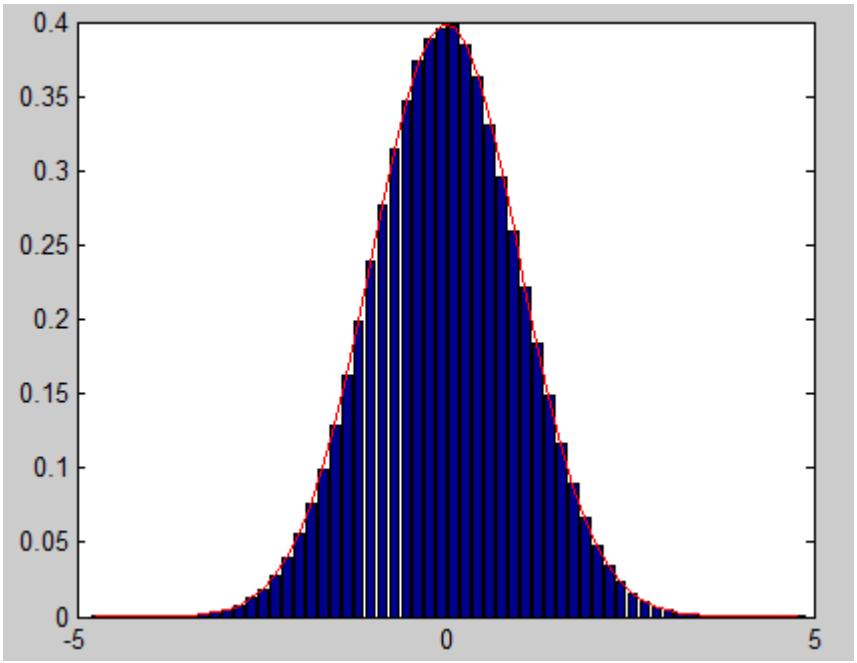

Figure 4.13Distribution normale de BM.

Nous avons remarqué que tous les graphes donnent des meilleurs résultats avec un temps de simulation dans FPGA parfait.

4.4.1. Comparaison des Tails :

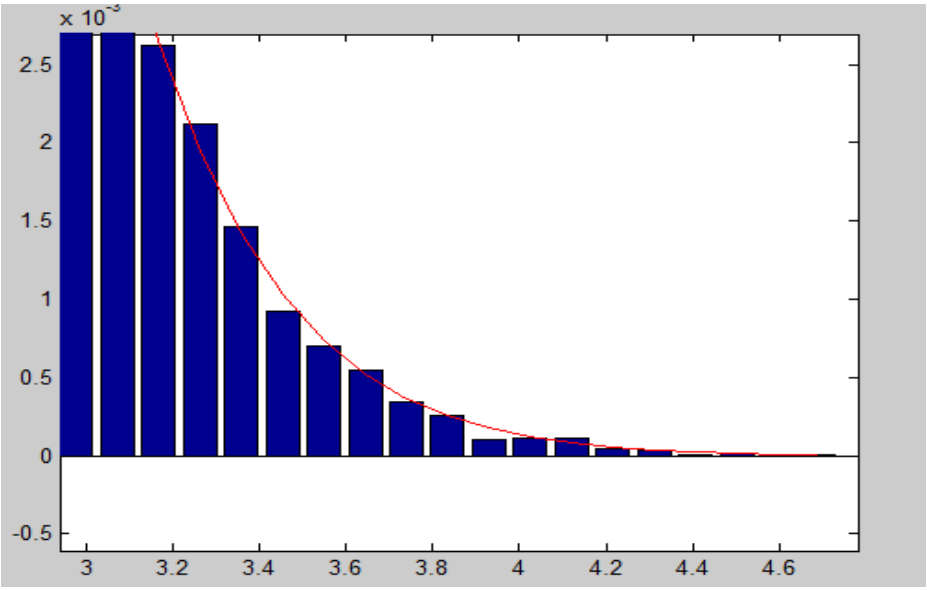

Figure 4.14Tail de signal sortie dans Matlab.

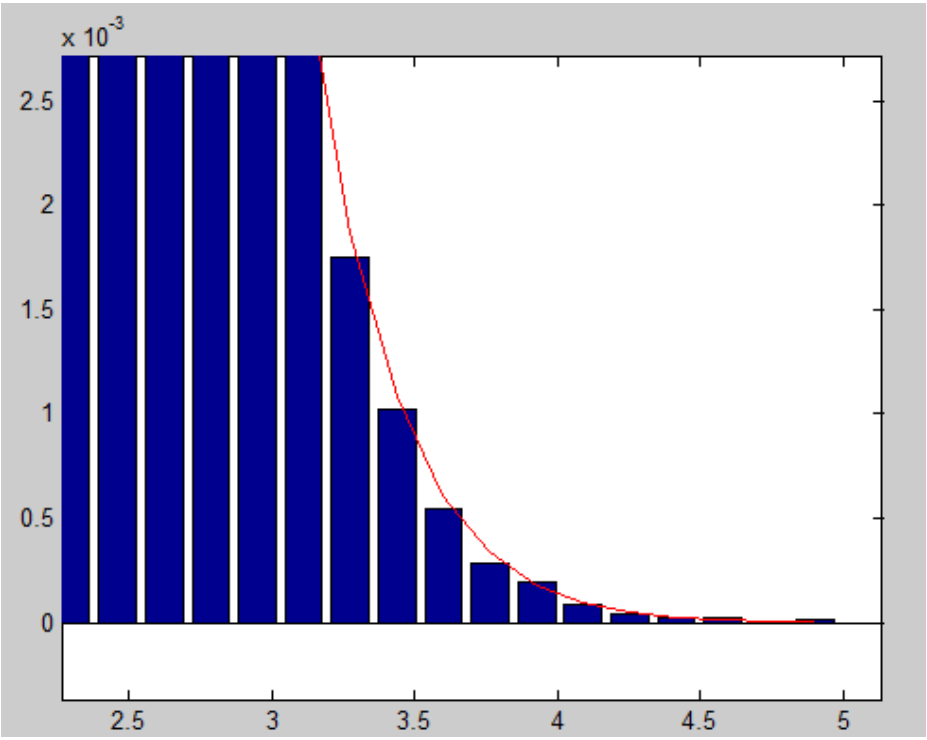

Figure 4.15Tail de signal sortie dans système generator.

Commentaire :

On remarque que le tail de la simulation FPGA est bien claire que le tail de la simulation dans Matlab.

4.4.2. Implémentation sur la carte FPGA

Bloc System Generator : Le bloc « SystemGenerator » : sert à transformer le modèle conçu en code VHDL (ou VERILOG) synthétisable.

Nous avons changé la compilation de bloc de system Generator de HDL Netlist vers Hardware Co-Simulation pour avoir cet circuit :

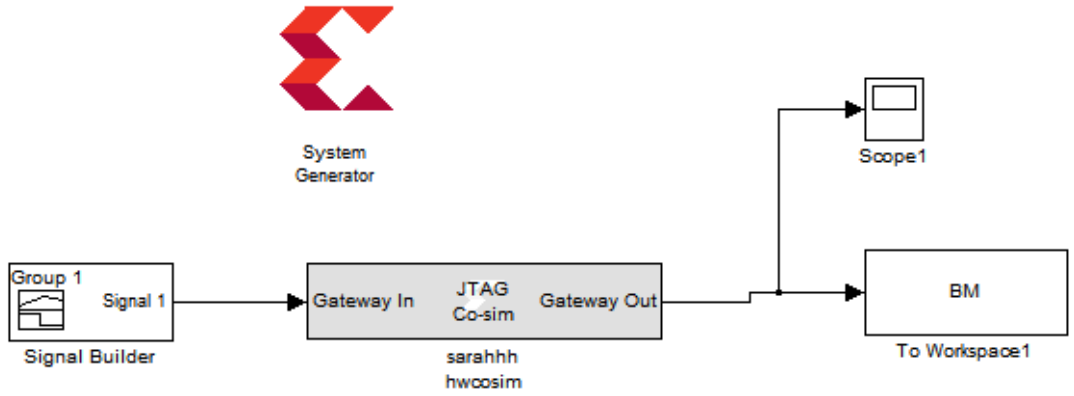

Figure 4.16 Co-Simulation hardware environnement FPGA.

Résultats :

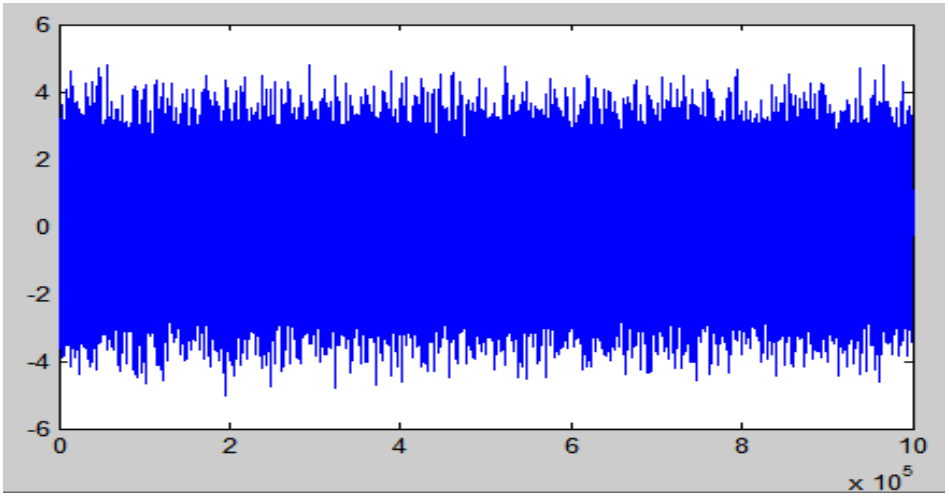

Figure 4.17 bruit blanc gaussien par FPGA.

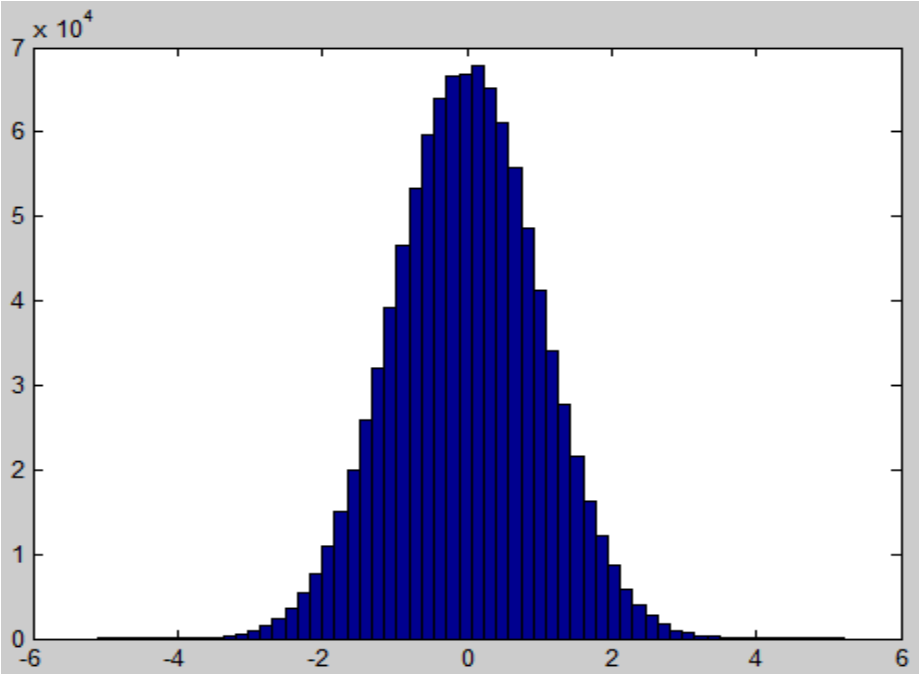

Figure 4.18 Distribution d'une v.a gaussienne réel par FPGA.

Remarque : Cette figure représente la Distribution d'une v.a gaussienne réel générée par la carteFPGA et distribution idéale normal N (0 1).

Remarque : Les résultats obtenus à partir des simulations dans Matlab et FPGA sontsemblables.

4.5. Conclusion

Dans ce chapitre nous avons présenté implémentations et résultats de simulation sur MATLAB et FPGA.

Les testes sur FPGA prouvent que l'implémentation sur FPGA est plus rapide que l'implémentation sur MATLAB.

Conclusion générale

Notre travail a pour but de générer de nombres aléatoires gaussien (GRNG) avec une utilisation optimisée de la logique FPGA du réseau de portes programmables, pour les applications communicantes. Les blocs RAM FPGA(BRAMs) sont associés à un multiplexeur pour réduire la taille de la mémoire requise pour implémenter logarithmique de Box-Muller.
Ce mémoire comporte quatre chapitres organisés dela manière qui suit :

Le premier chapitre, présente les principes des circuits logiques programmables et calcul sur FPGA. Qui nous a permis de connaitre l'architecture globale de FPGA (CLB, I /O, BRAM), et ces caractéristiques.

Les FPGA sont les plus couteux et plus compliqués à utiliser, mais ils offrent beaucoup plus de puissances et de flexibilité.

Dans le deuxième chapitre, présente comment peut-on générer les nombres aléatoires gaussiens par quatre méthodes : la méthode de la limite centrale, la méthode Box-Muller, la méthode Récursive et la méthode du rejet. Nous avons trouvé que la meilleure méthode de génération et la plus rapide à simuler c'est la méthode du rejet.

Dans le troisième chapitre, présente le fonctionnement de la méthode de Box-Muller dans le logicielle MATLAB et FPGA. La BM proposée est plus rapide lorsqu'on implémente sur FPGA.

Dans le dernier chapitre, présente les implémentations sur Matlab et FPGA. Et de puis les tests qu'on a fait sur la carte FPGA ils ont démontré la supériorité, car notre résultat se rapproche plus à la distribution idéale que les résultats du AWGNG de System Generator de Xilinx.

Annexes

Teste de la sortie de $f1$:

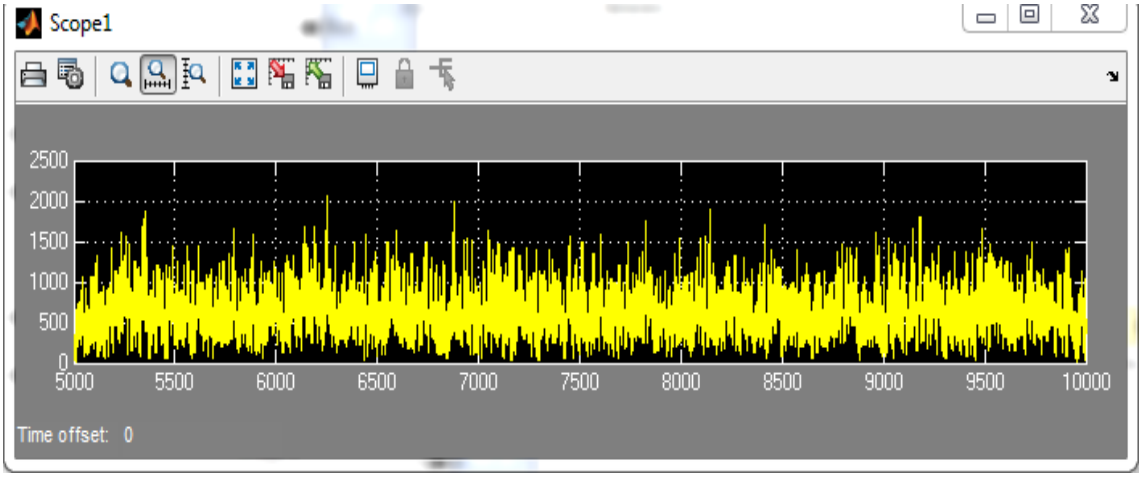

Figure 1 Rom f1

Teste de la sortie de f2 :

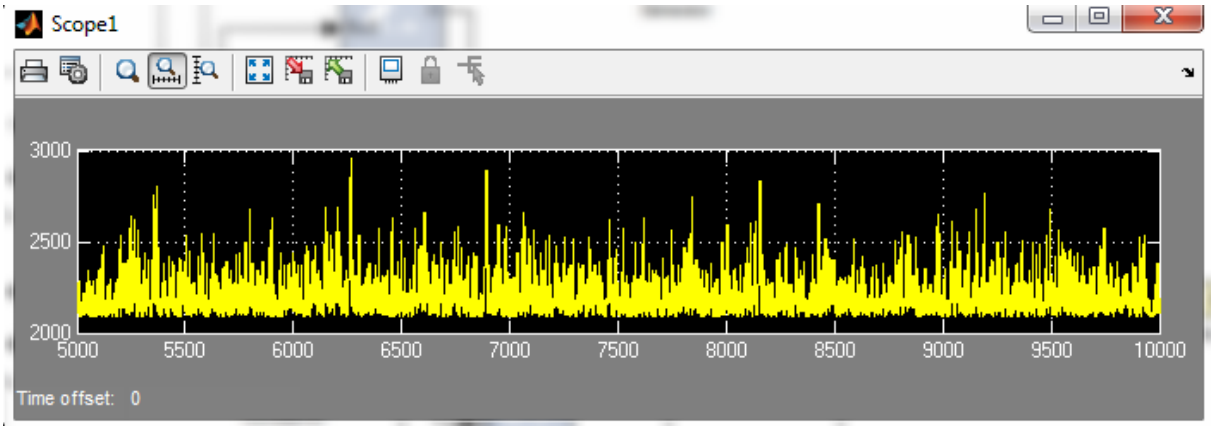

Figure 2 Rom f2

Teste de la sortie de f3 :

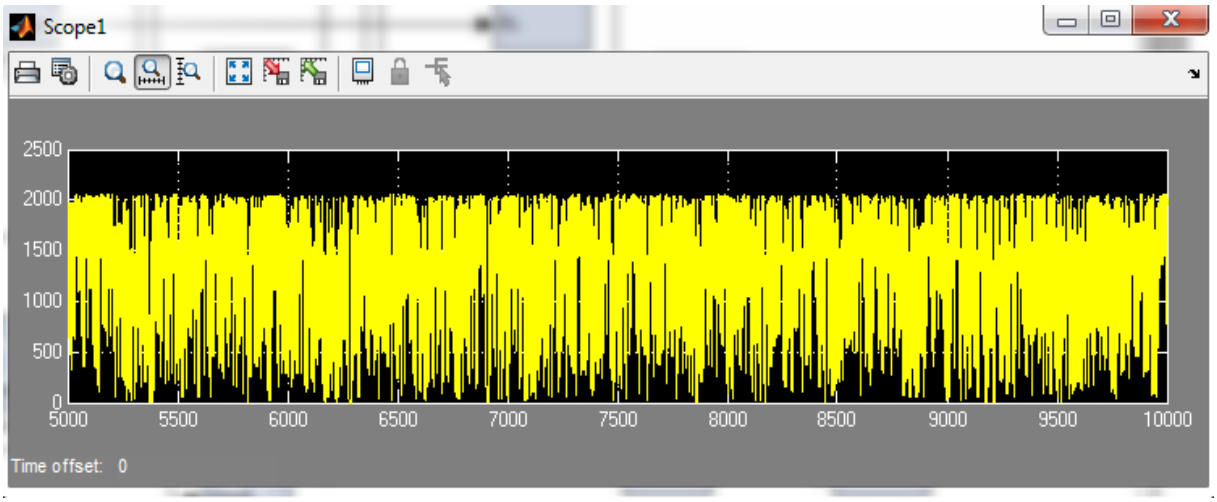

Figure 3 Rom f3

Teste de la sortie de g :

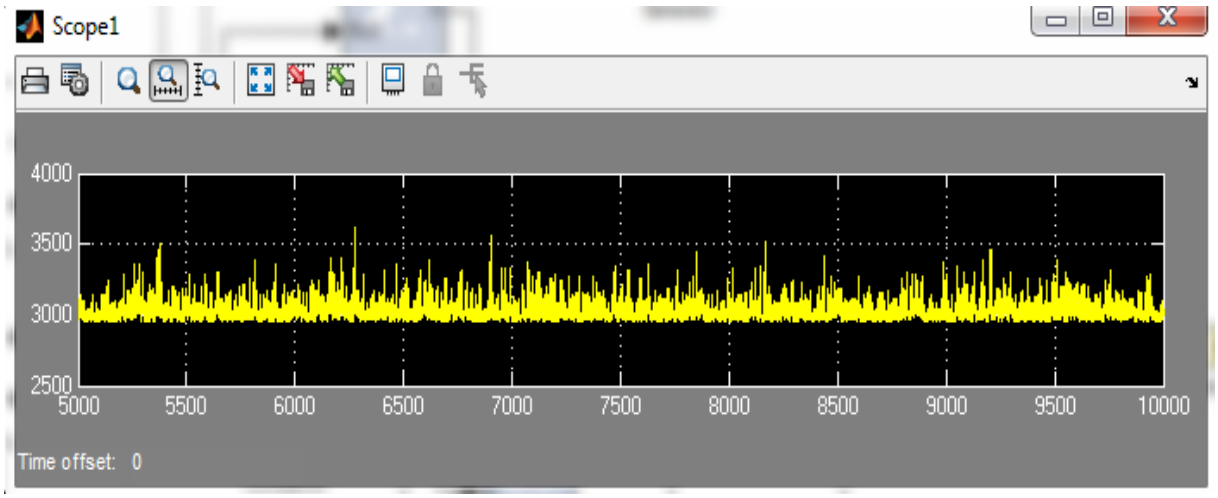

Figure 4 Rom g.

Bibliographie

[1] Futura-Sciences2022.Configuré un FPGA

[2] mindphp.FPGA (เอฟพีจีเอ) อุปกรณ์โลจิกแบบโปรแกรมได้image) 12 janvier 2018

[3]Futura-Sciences2022.Digital signal processor

[4] ADMINISTRATEURS b2BCHIEF.*Global DSP Digital Signal Processor Market, Industry, Market.*

https://www.google.com/search?q=DSP&tbm=isch&tbs=rimg:CSZSSwJ8ZBiYYbAMmx

TRhH6b8AEAsgIMCgIIABAAOgQIABAA&rlz=1C1RLNS_frDZ1007DZ1007&hl=ar

&sa=X&ved=0CAIQrnZqFwoTCLC2idjntPgCFQAAAAAdAAAAABAR&biw=1263&

bih=600#imgrc=KzPYngN2XW2stM

[5]Technologue pro12022.Caractéristiques des DSP

[6] https://fr.slideshare.net/miyamiya/digital-signal-processing-french/

[7]https://www.clicours.com/cours-electricite-circuits-logiques-programmables-fpga/

[8]Copyright 2017-2022, HardwareBee. All rights reserved.Configurable Logic Block (CLB)

[9]FPGA KEY 2022.Configurable logic module.

[10]design Methodologies for source Embedded systems.Springer-Verlag Berlin Heidelberg 2010
https://books.google.dz/books?id=T2a6oilgZ64C&pg=PA65&dq=configurable+logic+block+CLB&hl=ar&sa=X&ved=2ahUKEwjTl6-R3f_3AhWI4YUKHYqbA3A4ChDoAXoECAkQAg#v=onepage&q=configurable%20logic%20block%20CLB&f=false

[11]Digilent 2022. SLICEM et SLICE.

[12] 2022 clicours.com. logiques programmables FPGA

[13] Community.element14.Number Plate Recognition # 3: Implementing Block RAM usingVerilog 11 Jan 2022

[14]PLD Based Design withVHDL.Springer Nature Singapore Pte Ltd.2017 page 182

[15] FPGA KEY 2022.BLOCK RAM.

[16] Guanglie Zhang. Philip H.W. Leong .Dong-U Lee , John D. Villasenor.Ray C.C. Cheung. Wayne Luk 2022.ZIGGURAT-BASED HARDWARE GAUSSIAN RANDOM NUMBER GENERATOR.

[17]STATISTIQUE. Spinger-verlagFrance,Paris,2004 Page 569.

[18]BYJU'S. All rightsreserved 2022.centrale limittheorem

[19] Renaud SANTORO.Thèse soutenue à Lannion. Jeudi 17 Décembre 2009

[20]Y Baudot.Théorème central-limite

[21]Théorème central limite - Définition et Explications (techno-science.net)

[22]G.Marsaglia and T. A. Bray. 1964. A convenient method for generating normal variables. Journal of SIAM Review 6, 3 (1964), 260–264.

[23] J. S. Malik and A. Hemani ACM Computing Surveys, Vol. 49, No. 3, Article 53, Publication date: October 2016.

[24] J. L. Danger, A. Ghazel, E. Boutillon, and H. Laamari. 2000. Efficient FPGA implementation of Gaussian noise generator for communication channelemulation. In The 7th IEEE International Conference on Electronics, Circuits and Systems, 2000. Vol. 1. 366–369.

[25] Thèse soutenue à Lannion le Jeudi 17 Décembre 2009 présentée par Renaud SANTORO page 101

[26] *Solutions Simplilearn2022.*méthode récursive

[27] ADMINISTRATEURS d'éducative.*what is recursion.*2022

https://www.educative.io/courses/recursion-for-coding-interviews-in-java/m790BPVrPBr?aid=5082902844932096&utm_source=google&utm_medium=paid&utm_campaign=interview-prep&utm_content=search&utm_term=&utm_campaign=Grokking+Coding+Interview+-+USA%2B&utm_source=adwords&utm_medium=ppc&hsa_acc=5451446008&hsa_cam=1871092258&hsa_grp=138551263162&hsa_ad=599143029766&hsa_src=g&hsa_tgt=dsa-496636572923&hsa_kw=&hsa_mt=&hsa_net=adwords&hsa_ver=3&gclid=Cj0KCQjwqPGUBhDwARIsANNwjV6oPy6-oTcb7h35TFYVMrZ7k_-lYQdi9VB2QqeaiManiB979aBGEtkaAvoaEALw_wcB

[28] Dheeraj.M.ISC COMPUTER SCIENCE USING JAVA.edition de S.chand school.edition 2021.NEW DELHI IN INDE.Page 282
[29] HaroldMouchère Interface homme-machine [cs.HC]. INSA de Rennes, 2007.
[30] Robert, CP et G. Casella (2004). *Méthodes statistiques de Monte Carlo* , deuxième édition. New York : Springer.
[31]Copyright c 2007 by Karl Sigman
[32] https://ksachdeva17.medium.com/?source=post_page-----1f6aff92330d---------- Kapilsachdeva

[33] Maamoun.M, Ait Saadi.H,Dahmani.S,Zerari.Gh,Chabini.N,Beguenane.R.Définition de la méthode de Box Muller.AnOptimized FPGA Based Box-Muller GaussianRandomNumberGenerator Architecture for Communication Applications.

[34] 2022 Develop Paper.xilinx14.7
[35]https://www.researchgate.net/publication/283226531_Dynamic_linear_feedback_shift_registers_A_review

[36] C. S. Lamba, "Design and analysis of Stream Cipher for Network security," in Communication Software and Networks, 2010. ICCSN'10. Second International Conference on, 2010, pp. 562-567.

[37]B.Schneier, Applied cryptography: protocols, algorithms, and source code in C, 1996.

[38] P. FIPS, "140-2: Security requirements for cryptographic modules," National Institute of Standards and Technology, 2001.

[39] G. Marsaglia, "DIEHARD statistical tests," Florida state university,(http://www. stat. fsu. edu/pub/diehard/), 1995.

[40] A. Rukhin, J. Soto, J. Nechvatal, E. Barker, S. Leigh, M. Levenson, et al., "Statistical test suite for random and pseudorandom number generators for cryptographic applications, NIST special publication," 2010.

[41] Klein 2013, p. 17
[42] Cagigal 1986, p. 191