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

Mention Électronique
Spécialité Systèmes de télécommunication

présenté par

Abbas Hichem

&

Choumane Mohamed

Étude et implémentation sur FPGA d'un générateur de bruit Gaussien avec la méthode Box-Muller

Proposé par : Mr. Maamoun

Année Universitaire 2017-2018

Remerciements

Nous remercions Dieu de nous avoir donné patience et courage pour accomplir ce travail.

Nous tenons à remercier notre encadreur Mr. Maamoun qui s'est toujours montré à l'écoute et très disponible tout au long du travail, ainsi que son aide et le temps qu'il nous a consacré afin que notre travail soit bien fait et bien présenté.

Nous remercions également tous les membres du jury pour nous avoir honorés par leur présence et pour avoir acceptés d'évaluer ce travail de mémoire.

Nous tenons à exprimer notre profonde gratitude à tous les enseignants de la spécialité System de Télécommunications.

Enfin, nous n'oublions pas à remercier tous le staff du département d'électronique de l'université de Blida.

ملخص:

تم تطوير مولد ضوضاء أبيض "غاوسي" جيد الجودة على هدف " AGPF " كجزء من محاكاة قنوات الاتصالات. يتم الحصول على دقة عالية وتكلفة منخفضة للمولد من خلال استخدام مناسب و بطريقة ملائمة لطريقة "BoxMuller". يتم تقديم نموذج معلمات مكتوب مع "aLatMb" ، مولد الضجيج الغوسي. تظهر النتائج في التعقيد والأداء التي تم الحصول عليها من خلال نموذج baLatM وHVDL اهتمام النموذج المقترح

كلمات المفاتيح : مولد ضوضاء "غوسي" بيضاء , طريقة Box-Muller ,FPGA.

Résumé : Un générateur de bruit blanc gaussien(GBBG) de "bonne qualité" est développé sur une cible FPGA, dans le cadre d'une émulation de canal de communications numériques. Une grande précision et un faible coût matériel du générateur sont obtenus par une utilisation convenablement adaptée de la méthode de Box-Muller. Un modèle paramétrable écrit avec MATLAB, du générateur de bruit gaussien est présenté. Les résultats en complexité et performance obtenus grâce aux modèles MATLAB et VHDL montrent l'intérêt du modèle proposé.

Mots clés : GBBG ; Box-Muller ; FPGA.

Abstract : A hardware White Gaussian Noise Generator (WGNG) is developed for communication channel emulation in FPGA circuit.

High accuracy, fast and low-cost hardware are reached by a smart using of the Box-Muller method. The performance of the designed model is investigated using MATLAB and VHDL models. The complexity and the performance level are given for some configurations and show the interest of the proposed model.

Keywords : WGNG ; Box-Muller ; FPGA.

Dédicace

Je dédie ce travail à ma famille qui m'a doté d'une éducation digne, son amour a fait de moi ce que je suis aujourd'hui, particulièrement les parents, mes grands-mères, frères et ma sœur

A tous mes amis : bilal, nounou, hichem et mes amis d'étude Zahra, meriem, noura, imane, fereil qui m'avez toujours soutenu et encouragé durant ces années d'études

Abbas Hichem

Dédicace

Je dédie ce modeste travail à mes très chers parents pour l'éducation qu'ils m'ont prodigué ainsi que tous leurs sacrifices.

A toute la famille qui m’ont soutenu durant les moments difficiles.

Au groupe Master 2 ST et tous les étudiants avec qui j’ai parcouru mon cursus universitaire et a tous mes amis.

Choumane Mohamed

Listes des acronymes et abréviations

TTL : Transistor Transistor Logic

CMOS : Complementary Metal Oxide Semiconductor

EPLD : Erasable Programmable Logic Device

ASIC : Application Specific Integrated Circuit.

EEPROM : Electrically Erasable Programmable Read-Only Memory

SRAM : Static Random Access Memory.

FPGA : Field Programmable Gate Array.

PLD : Programmable Logic Device

CPLD : Complex Programmable Logic Device

RAM : Random Access Memory

LUT :Luck Up Table.

DSP : Digital Signal Processor.

IOB : Input output block.

VHDL : Very High Density Logic

DCM : Digital Clock Manager

EPLD : Erasable Programmable Logic Device

CLB : Configurable Logique Bloc

ROM : Read Only Memory.

ISE : Integrated Software Environment.

HDL : Hardware Description Langage.

GAL : Generic Array Logic

v.a : variable aléatoire

σ^2 : la variance

LFSR : Linear Feedback Shift Register

SG : System Generator

Table des matières

Chapitre 2 : Architecture et calcul sur FPGA

Chapitre 3 : Synthèse du générateur de bruit gaussien

Chapitre 4 : Architecture et implémentation sur FPGA

Liste des figures

Chapitre1

Chapitre 2

Chapitre 3

Chapitre 4

Liste des tableaux

Introduction générale

La conception d'un circuit intégré pour une application de communication numérique (décodeur correcteur d'erreur, démodulateur, …) requiert souvent des compromis entre complexité matérielle et performances. Les méthodes formelles de calcul de performances sont limitées, en général, à des problèmes simples et linéaires et elles ne peuvent pas tenir compte de l'influence des non linéarités introduites par les opérations de saturation et d'arrondi. Des simulations logicielles par la méthode de Monte-Carlo sont utilisées afin d'estimer l'influence de tel ou tel paramètre sur les performances. Malheureusement, cette méthode devient rapidement coûteuse en ressources de calcul et en temps.

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 l'algorithme Box-Muller pour obtenir des échantillons tirée d'une gaussienne moyenne nulle.

Dans la majorité des cas, cela se produit dans des environnements tels que la simulation par ordinateur, où des fonctions telles que sinus, cosinus et racines carrées sont faciles à réaliser, et où il y a suffisamment de précision pour que les effets des mots de longueur finis soient négligeables.

Néanmoins, une méthode optimisée pour simulation d'un système de communication sur ordinateur ne peut pas générer systématiquement 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.

Le but de ce travail est d'étudier le principe du générateur de la variable aléatoire gaussienne, appliquée dans la communication numérique, existant et d'implémenter une approche moins complexe et implémentable sur FPGA.

Ce mémoire est réparti en quatre chapitres :

Le premier chapitre est consacré à la présentation le principe et l'architecture du générateur de la variable aléatoire gaussienne, appliquée dans la communication numérique. Les notions et les calculs, accomplis par la méthode présentée par Box et Muller, sont examinés.

Le deuxième chapitre expose l'architecture et les calculs sur les circuits FPGA. L'objectif est de comprendre la méthodologie de conception et de validation sur circuit FPGA.

Dans le troisième chapitre, une approche simplifiée de synthèse du générateur de bruit gaussien, pour une implémentation sur FPGA, est traitée avec une nouvelle méthode et perspective.

Dans le quatrième chapitre, nous présentons l'architecture adoptée sur FPGA, les résultats d'implémentation et les comparaisons entre les résultats théoriques, les résultats du « System Generator », de MatLab et nos résultats

La thèse se termine par une conclusion générale où nous commentons le travail effectué et nous proposons des améliorations et perspectives.

Chapitre 1

Les générateurs de bruit blanc

Chapitre 1 Les générateurs de bruit gaussien

1.1 Introduction :

Les variables aléatoires sont très utilisées dans les applications qui dépendent de la théorie des probabilités et statistiques.
Les variables aléatoires sont utilisées pour modéliser le résultat d'un mécanisme non-déterministe ou encore comme le résultat d'une expérience non-déterministe qui génère un résultat aléatoire.

Le signal bruit généré dans les appareils électroniques varie aléatoirement, pour cela il est considéré comme un résultat d'un mécanisme non-déterministe ou encore comme le résultat d'une expérience aléatoire.

1.2 Les signaux aléatoires :

Les signaux aléatoires sont des signaux qui varient aléatoirement, c'est-à-dire des signaux qui ne peuvent pas être décrits sous un terme mathématique, qui traduit le fonctionnement d'un mécanisme non déterministe ou aléatoire.
Les signaux aléatoires sont des signaux dont l'évolution temporelle est imprévisible et dont on ne peut pas prédire la valeur à un temps t. La description est basée sur les propriétés statistiques des signaux (moyenne, variance, loi de probabilité, …)
En électronique tous les appareils fonctionner avec un bruit qui est un signal aléatoire et qui varie aléatoirement. Pour cette raison, nous utilisons des caractéristiques statistiques des signaux pour modéliser les différents signaux bruités.

1.3 Les processus aléatoires :

Les processus aléatoires décrivent l'évolution d'une grandeur aléatoire en fonction du temps (ou de l'espace probabiliste). Il existe de nombreuses applications des processus aléatoires notamment en physique statistique (par exemple le ferromagnétisme, les transitions de phases, etc.), en biologie (évolution, génétique et génétique des populations), médecine (croissance de tumeurs, épidémie), et bien entendu les sciences de l'ingénieur, dans ce dernier domaine, les applications principales sont pour l'administration des réseaux, de l'internet, des télécommunications et bien entendu dans les domaines économique et financier

Dans le domaine de l'électronique général les signaux bruités qui varient aléatoirement sont considérés comme un processus aléatoire. On définit un processus aléatoire sous la forme suivant :

Un processus aléatoire (scalaire ou vectoriel) est défini comme une famille de variables ou vecteurs aléatoires indexée par un ensemble de paramètres t Є T. La notation est {xt(ω) | t Є T}. T peut être un ensemble discret ou continu.

C'est donc une fonction de deux paramètres : t le temps et ω paramètre aléatoire (résultat d'une expérience).

- Pour chaque t, xt(•) est une variable aléatoire égale à l'état du processus considéré a l'instant t.
- Pour ω fixé, x•(ω) est réalisation du processus qui est une fonction du temps
- Pour t et ω fixés, xt(ω) est un nombre [1].

1.4 Densité spectral de puissance :

En théorie du signal, les spectres sont associés à la transformée de Fourier. Dans le cas de signaux déterministes, c'est une façon de représenter les signaux comme une superposition d'exponentiel les .Pour les signaux aléatoires, il s'agit principalement d'étudier la transformée des moyennes.

La densité spectrale de puissance Ψx(ω) du processus aléatoire

x(t) est définie comme la transformation de Fourier, si elle existe, de la

Matrice de corrélation$Rx(\tau)$.[1]

$$\psi x(\omega)=\int_{+\infty}^{-\infty} e^{-j\omega\tau} Rx(\tau) d\tau \qquad (1.1)$$

Réciproquement,

$$Rx(\tau)=\frac{1}{2\pi}\int_{+\infty}^{-\infty} e^{j\omega\tau} \psi x(\omega) d\,\omega \qquad (1.\,2)$$

1.5 Le bruit blanc :

Le bruit blanc provient de l'agitation thermique des électrons dans les conducteurs et les matériaux. Mais, il est également le résultat de toute agitation de transporteur de matière du fait de l'environnement extérieur. Il est donc omniprésent dans tout système ou un transfert d'information a lieu. On l'appelle "bruit blanc" car sa densité spectrale de puissance est constante, ce qui fait qu'il "possède" toutes les fréquences comme un signal optique de couleur blanche.

En traitement du signal le bruit blanc est un signal aléatoire qui n'est pas corrèle, c'est-a-dire, que la valeur du bruit a un instant donne, n'est pas corrélée (n'a pas de relation) 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, car une DSP constante sur toutes les fréquences implique une puissance infinie. Cependant, un bruit peut avoir une DSP constante sur une bande de fréquence limitée (bande du signal utile par exemple) et par abus de langage, on l'appelle bruit blanc aussi. Un bruit blanc est alors un signal aléatoire de moyenne nulle ayant la fonction Dirac comme fonction d'autocorrélation.

Bruit blanc fort et bruit blanc faible :

En statistique spatiale, on distingue les deux définitions suivantes :

Bruit blanc fort

Un processus stochastique X sur un espace S est un **bruit blanc fort** (BBF) si les variables $X(s)$, $s \in S$ sont centrées, indépendantes et identiquement distribuées.

Bruit blanc faible

Un processus stochastique X sur un espace S est un **bruit blanc faible** (BBf) si les variables $X(s)$, $s \in S$ sont centrées, décorrélées, et de variances finies constantes.

1.6 Processus gaussien :

Très fréquemment, on rencontre des signaux modélisable par une loi de Gauss.

1.6.1 Variable et densité de probabilité gaussienne :

Le processus est Gaussien si pour chaque ensemble fini d'instants les variables aléatoires ont une densité de probabilité Gaussienne. Cette densité dépend des instants auxquels on observe le processus.

1.6.2 Propriétés des processus Gaussien :

La densité d'ordre n ne dépend que des moments d'ordre 1 et 2. Par conséquent, la stationnarité d'ordre 2 entraîne la stationnarité au sens strict. Dans le cas des processus Gaussien les deux notions de stationnarité sont équivalentes.

Un processus Gaussien conserve son caractère Gaussien pour toute transformation linéaire. Cette propriété est intéressante pour le filtrage linéaire de tels signaux.

Pour qu'un processus Gaussien stationnaire soit ergodique, il existe une condition suffisante. Son spectre de puissance doit être continu. Il faut donc que sa fonction d'autocorrélation soit intégrable.

1.7 Générateurs des variables aléatoires uniformes :

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). Le registre a décalage est une série de bascules reliées entre elles de façon

qu'à chaque coup d'horloge, tous les bits sont décales. Le LFSR est un registre à décalage dont le bit d'entrée est une fonction linéaire de l'état précédant. Ce bit d'entrée est régit par des opérations XOR (ou exclusif) sur certains bits du mot stocke dans le registre (voir Figure 3). Au début, le registre a décalage possède un état initial (obligatoirement différent de 0) appelé en anglais speed et à chaque coup d'horloge, un bit est génère et est injecte à l'entrée du LFSR qui passe à un nouvel état. Si un registre est compose de *n* bascules (n bits), et si la fonction de rétroaction est bien choisie, on aura 2n -1 états possible, âpres avoir parcouru l'ensemble des états, le système revient à son état initial et répète son cycle. [3]

1.7.1 Loi uniforme continue :

les lois uniformes continues forment une famille de lois de probabilité à densité caractérisées par la propriété suivante : tous les intervalles de même longueur inclus dans le support de la loi ont la même probabilité. Cela se traduit par le fait que la densité de probabilités de ces lois est constante sur leur support.

La loi uniforme continue est une généralisation de la fonction rectangle à cause de la forme de sa fonction densité de probabilité. Elle est paramétrée par les plus petites et plus grandes valeurs *a* et *b* que la variable aléatoire uniforme peut prendre. Cette loi continue est souvent notée *U* (*a*, *b*)[4].

Caractérisation :

Densité

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

$$f(x) = \begin{cases} \frac{1}{b-a} & pour\ a \leq x \leq b \\ 0 & sinon \end{cases}$$

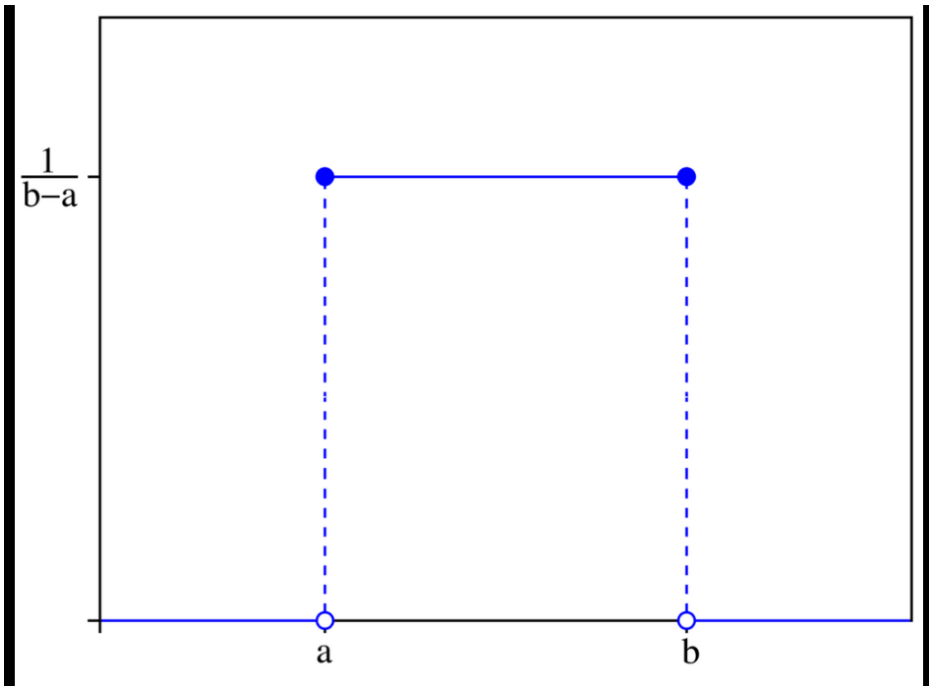


Figure 1.1 : La densité de probabilité de la loi uniforme continue

Fonction de réparation :

La fonction de répartition est donnée par

$$F(x) = \begin{cases} 0 & pour\ x < a \\ \dfrac{x-a}{b-a} & pour\ a \leq x < b \\ 1 & pour\ \ x \geq b \end{cases}$$

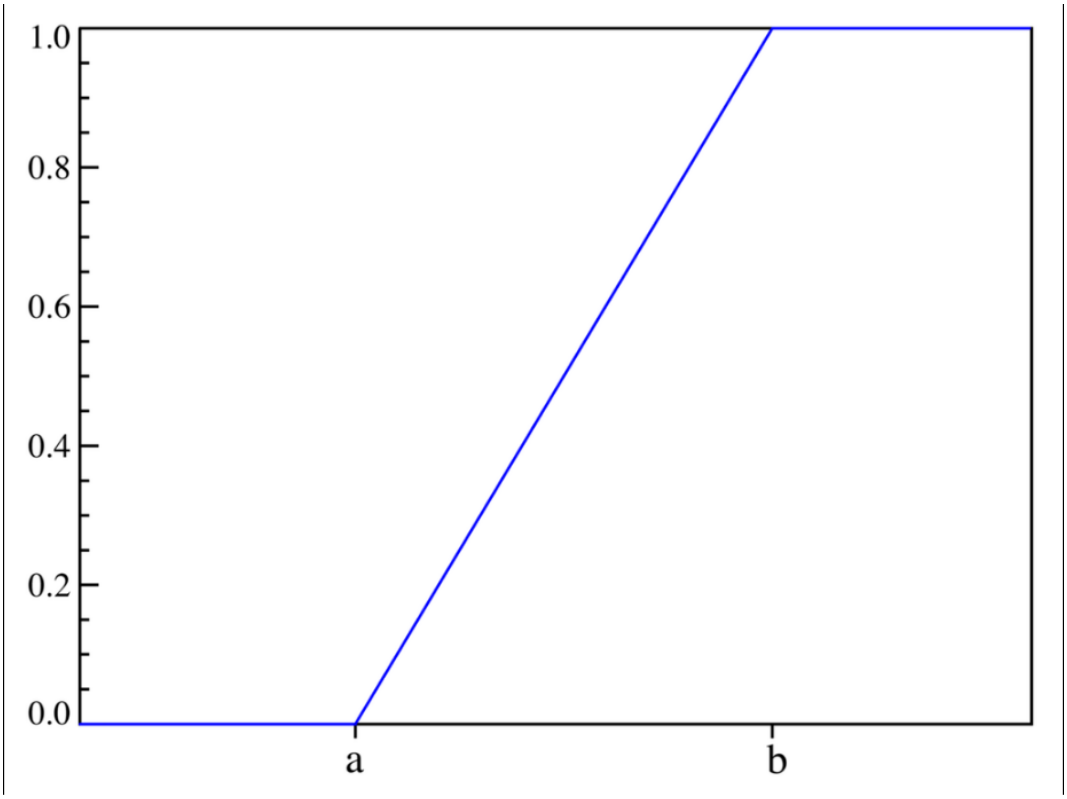


Figure 1.2 : La fonction de répartition

1.7.2 Loi uniforme standard :

Le cas particulier *a* = 0 et *b* = 1 donne naissance à la loi uniforme standard, aussi notée $U(0,1)$.

1.7.3 Le registre à décalage à rétroaction linéaire LFSR :

Définition :

Un registre à décalage à rétroaction linéaire (LFSR : Linear Feedback Shift Register) binaire de longueur L est composé d'un registre à décalage contenant une suite de L bits (si, . . ., si+L-1) et d'une fonction de rétroaction linéaire

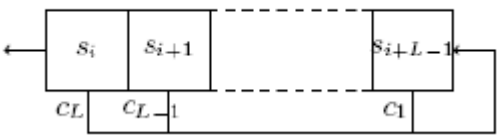


Figure 1.3 : Un LFSR de longueur L à l'instant i

Le fonctionnement d'un LFSR binaire de longueur L est le suivant : à chaque top d'horloge, le bit de « gauche » si constitue la sortie du registre, et les autres sont décalés vers la gauche ; le nouveau bit si+L placé dans la cellule de « droite » du registre est donné par une fonction linéaire :

$$s_{i+L}=c_1 s_{i+L-1}+c_2 s_{i+L-2}+\cdots+c_{L-1} s_{i+1}+c_1 s_i$$

où les coefficients C_i sont binaires.

- la méthode de Box-Muller est plus précise et plus rapide pour générer des variables aléatoires normales.

Modes de connexion

Le mode le plus naturel pour représenter la connexion entre les différents étages du registre est le mode dit de « Fibonacci ». Il est appelé ainsi car la suite de Fibonacci se représente dans ce mode. En 2002 Andy Klapper et Mark Goresky mettent en place le mode dit de « Galois »

Fibonacci

Un registre en mode Fibonacci applique strictement la définition d'un LFSR : les contenus des différents étages sont ajoutés ou non les uns aux autres, le résultat de cette addition est ensuite placé dans l'étage d'entrée du registre et tous les étages subissent un décalage vers la sortie

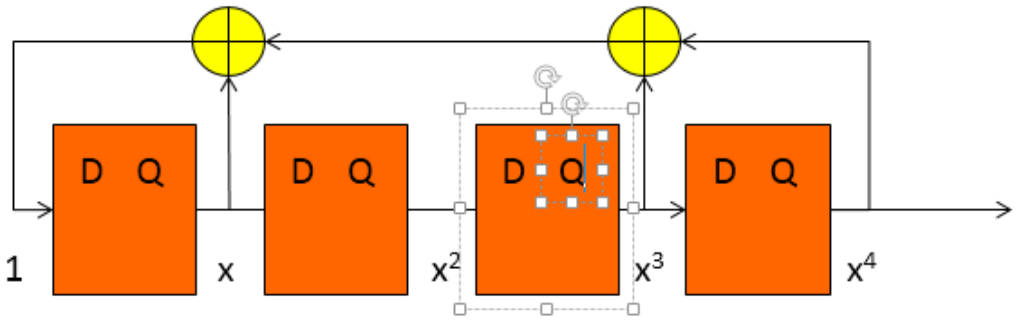


Figure 1.4 : Un registre en mode Fibonacci

Galois

Dans le mode dit de Galois le contenu de l'étage de sortie est ajouté ou non au contenu des étages du registre puis tous les étages subissent un décalage vers la sortie et le contenu de l'étage sortant est réinjecté dans l'étage d'entrée[17].

Au niveau matériel les LFSRs sont souvent mis en œuvre en utilisant ce mode car celui-ci est plus rapide et présente moins de latence que le mode Fibonacci puisque les étages sont mis à jour simultanément

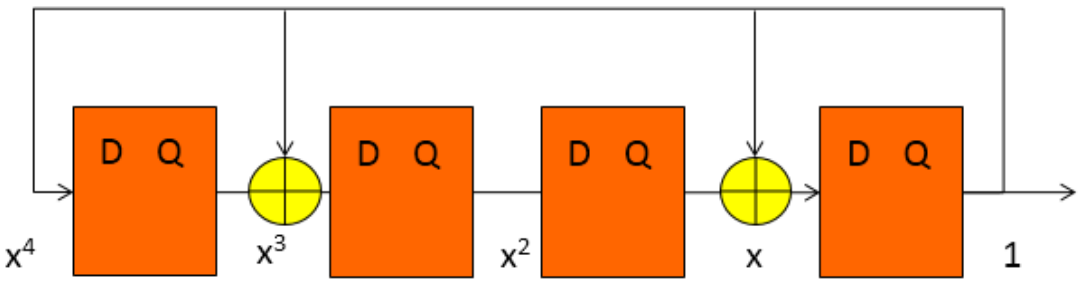


Figure 1.5 : Un registre en mode Galois

1.8 Générateurs des variables aléatoires normales :

La manière la plus simple de générer une variable aléatoire normale est l'utilisation méthode de Box-Muller

1.8.1 Méthode de Box-Muller :

La méthode de Box-Muller (George E. P. Box et Mervin E. Muller, 1958) consiste à générer des paires de nombres aléatoires à distribution normale centrée réduite, à partir d'une source de nombres aléatoires de loi uniforme.

La transformation prend communément deux formes.

- La forme « simple » transforme des coordonnées polaires uniformément distribuées en des coordonnées cartésiennes normalement distribuées.
- La forme « polaire » transforme des coordonnées cartésiennes uniformément distribuées dans le cercle unité (obtenues par rejet) en des coordonnées normalement distribuées.

1.8.2 Algorithme de Box-Muller :

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

Soient :

X1 =R$\cos\Theta$ =$\sqrt{-2lnU1}\cos(2\pi U2)$

X2 =R$\sin\Theta$=$\sqrt{-2lnU1}\sin(2\pi U2)$

. $\Theta = (2\pi U2)$

R=$\sqrt{-2lnU1}$

Alors X1 et X2 sont indépendantes et de même loi N (0, 1).[5]

1.8.3 La densité de probabilité de la loi normale :

La loi normale est donnée par :

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1(x-\mu)^2}{2\sigma^2}} \qquad (1.3)$$

La courbe de cette densité est appelée *courbe de Gauss* ou *courbe en cloche*, entre autres. C'est la représentation la plus connue de cette loi. La loi normale de moyenne

nulle et d'écart type unitaire est appelée loi normale centrée réduite ou loi normale standard.

Lorsqu'une variable aléatoire *X* suit la loi normale, elle est dite *gaussienne* ou *normale* et il est habituel d'utiliser la notation avec la variance σ^2 :

$$x \sim N(\mu, \sigma^2)$$

Cas particulier

$$\mu = 0 \ \ et \ \ \sigma = 1$$

Donc loi normale centre/ réduite.

1.8.4 La loi normale centrée :

La loi normale centrée réduite est la loi de probabilité absolument continue dont la

La densité de probabilité est donnée par la fonction $\varphi: R \to R +$ défini par

$$\varphi(t) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1t^2}{2}} \qquad (1.4)$$

Cette loi est dite centrée puisque son moment d'ordre 1 (espérance) vaut 0 et réduite puisque son moment d'ordre 2 (variance) vaut 1, tout comme son écart type. Le graphe de la densité φ est appelé fonction gaussienne, courbe de Gauss ou courbe en cloche. Cette loi est notée grâce à la première lettre de « normal », une variable aléatoire *X* qui suit la loi normale centrée réduite est notée : [6]

$$x \sim N(0,1)$$

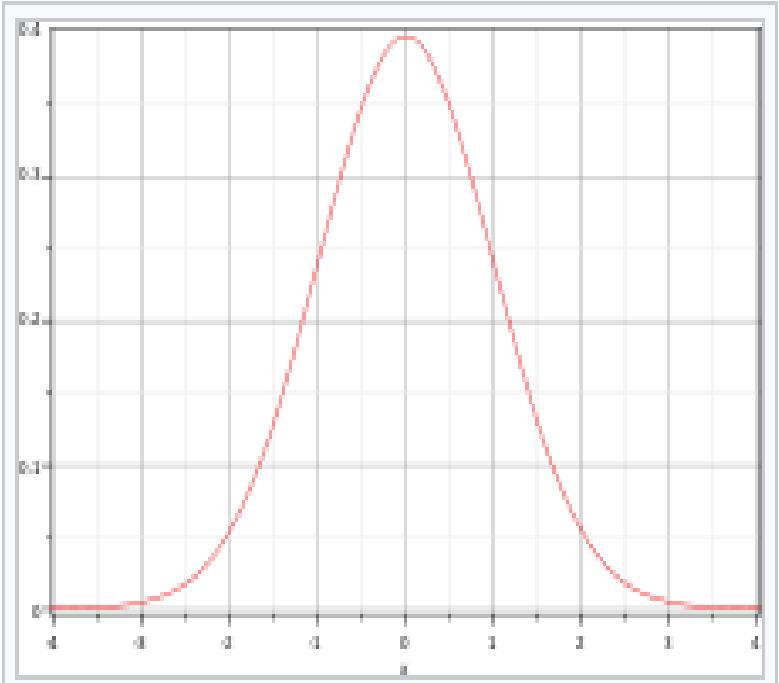


Figure 1.6 : Fonction de densité de la loi normale centrée réduite

1.8.5 La fonction de répartition centrée :

La fonction de répartition de la loi normale centrée réduite $F: R \to R +$ défini par :

Pour tout $x \in R$

$$F(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1t^2}{2}} dt \qquad (1.5)$$

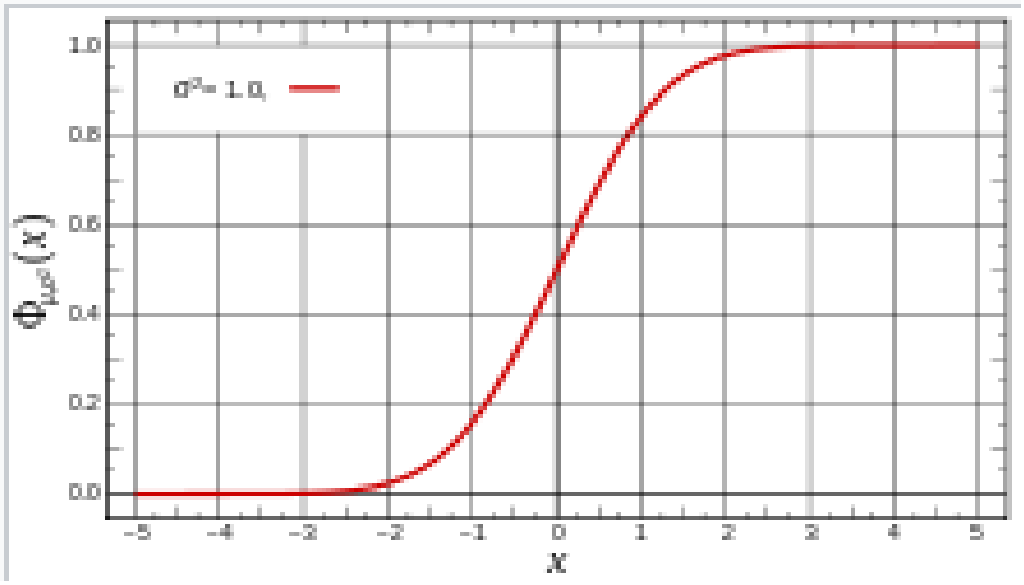


Figure 1.7 : Fonction de répartition de la loi normale centrée réduite

1.8.6 La fonction caractéristique :

La caractérisation de la loi normale par sa fonction caractéristique présente un intérêt pour démontrer certaines propriétés, comme la stabilité par addition ou le théorème central limite. Cette fonction caractéristique $\Phi: R \to R +:$ est donnée par

$$\Phi(t) = x^{-\frac{x^2}{2}} \quad (1.6)$$

Pour tout $t \in R$

1.9 Conclusion :

Dans ce chapitre nous avons présenté les déférentes définitions des opérations (Densité spectral de puissance , Algorithme de Box-Muller , LFSR,..... ect) qui touchent les processus aléatoire et les variables aléatoire. Nous avons particulièrement élucidé la théorie de probabilité et statistique pour le processus de gauss. Nous avons exposé

les étapes de la méthode Box-Muller. La méthode consiste à générer des variables aléatoires, uniformes et indépendantes, comme première étape. La variable aléatoire Gaussienne est le résultat de la dernière étape, de l'approche Box-Muller.

Chapitre 2

Architecture et calcul sur FPGA

Chapitre 2 Architecture et calcule sur FPGA

2.1 Introduction

Les circuits numériques peuvent etre classés dans quatre catégorie. La premier catégorie regroupe les circuits logique standards, nous pouvons citer les technologies TTL et les technologies CMOS. La deuxième catégorie regroupe les circuits à fonctionnement programmables, comme les microprocesseurs, les microcontrôleurs et les DSP. La troisième catégorie regroupe les circuits intégrés spécifique (ASIC), nous pouvons invoquer les circuits pré-diffusés qui intègre un ensemble de cellules logiques à base de transistor implanté, mais non connecté. Nous mentionnons aussi les circuits pré-caractérisé est les circuits sur-mesure qui représentent l'avantage de garantir une performance maximale, mais un temps de développement important et un cout de développement élevé. La dernière catégorie des circuits numérique regroupe les circuits à logique programmable.

Un circuit à logique programmable est constitué d'un ensemble de portes logiques ou cellules logique relie par des interconnexions reconfigurable.

Ils existent des circuits programmables une seule fois avec les technologies fusible, et des circuits reprogrammables plusieurs fois avec les technologies mémoire flash EEPROM et SRAM. Ces termes désignent les technologies interconnexions.

Dans les circuits PAL, la technologie des interconnexions est à base de la technologie fusible. Dans les circuits FPGA, la technologie des interconnections est basée sur la technologie SRAM [7].

2.2 Circuits à logiques Programmables

Actuellement les objets techniques utilisent de plus en plus la logique programmée. Ces structures ont besoin de s'interfacer entre elles. Elles sont utilisées généralement pour réaliser des interfaces et des fonctions à base de fonctions logiques élémentaires, compteurs, registres. Le nombre de circuits nécessaires pour remplir ces fonctions peut devenir très vite important. Pour diminuer les couts de fabrication, de développement et de maintenance, les fabricants de circuits intégrés ont donné naissance aux Circuits à Logique Programmable ou encore P.L.D (Programmable Logic Device). Ces circuits sont capables pour un objet technique de réaliser plusieurs fonctions logiques dans un seul circuit. Si ces fonctions étaient réalisées à base de circuits logiques classique, il faudrait plusieurs circuits. Un autre avantage, l'évolution des fonctions d'un Objet Technique s'effectue par programmation comparée à une solution classique où il faut refaire un circuit imprimé si on veut modifier le fonctionnement. [8]

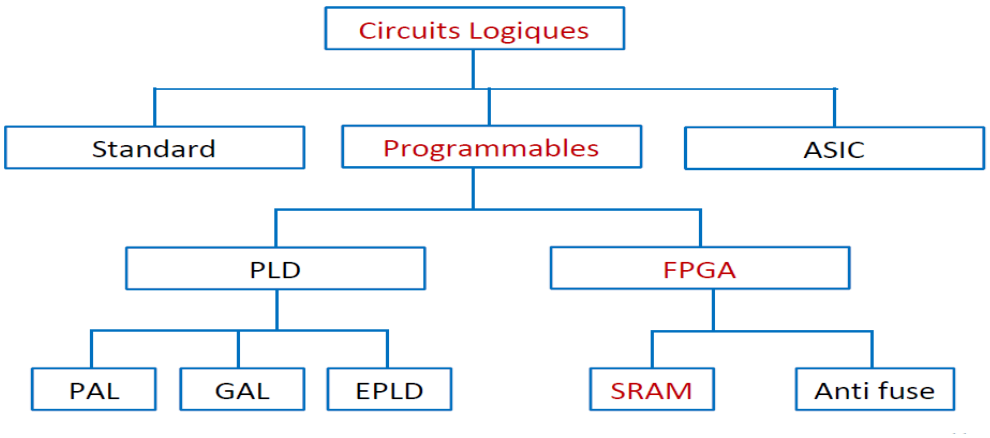


Figure 2.1 : Classification des circuits logiques

Nous allons nous concentrer sur l'étude la partie programmable des circuits FPGA .

2.3 FPGA

Les circuits logique programmable de types FPGA sont de plus en plus utilisés dans la conception des circuits numériques, ils sont constitués de plusieurs portes logiques, de mémoires, de multiplexeurs et un ensemble d'interconnexions, permettant de reliée des déférentes composants par programmation. Les déférents composants et les blocs logiques programmables du circuit FPGA sont disposés de manière ordonnée. La synthèse d'une fonction logique avec un circuit FPGA reviens a interconnecté un certain nombre de blocs logique à travers les interconnexions programmables.

Les circuits FPGA actuels permettent la synthèse et l'analyse de déférents algorithmes avec un temps très court. Il présent l'avantage de la grande souplesse architecturale qui permettent des évolutions rapides à moindre cout. L'évolution et aussi en terme d'application [9].

2.3.1 Quelque familles de FPGA

2.3.1.1 Virtex II

Solution FPGA de la première plate-forme de l'industrie de Densités des portes à système de 40K à 8M avec un Vitesse d'horloge interne de 420 MHz (données avancées).

2.3.1.2 Virtex 4

Sont disponibles en vitesse -12, -11 et -10 notes, avec -12 ayant la plus haute performance**.** Les familles sont considérées comme des périphériques hérités et ne sont pas recommandées pour les nouvelles conceptions, bien qu'elles soient toujours produites par Xilinx pour les conceptions existantes.

La famille Virtex-4 a été lancée en juin 2004 sur la technologie des procédés 90 nm. Les FPGA Virtex-4 ont été utilisés pour le projet ALICE (Une grande expérience sur les

collisionneurs d'ions) au laboratoire européen du CERN à la frontière franco-suisse pour cartographier et démêler les trajectoires de milliers de particules subatomiques.

2.3.1.3 Virtex 5

A été lancée en mai 2006 sur la technologie 65 nm. Le Virtex-5 LX et le LXT sont destinés à des applications à forte intensité logique et le Virtex-5 SXT est destiné aux applications DSP. Avec le Virtex-5, Xilinx a changé la structure logique de LUT à quatre entrées en LUT à six entrées. Avec la complexité croissante des fonctions de logique combinatoire requises par les conceptions de SoC, le pourcentage de chemins combinatoires nécessitant plusieurs LUT à quatre entrées était devenu un passage d'étranglement de performance et de routage. La nouvelle LUT à six entrées représentait un compromis entre une meilleure gestion des fonctions combinatoires de plus en plus complexes, au prix d'une réduction du nombre absolu de LUT par appareil. La série Virtex-5 est une conception de 65 nm fabriquée en technologie de traitement triple oxyde 1,0 V.

2.3.1.4 Virtex 6

A été lancée en février 2009 sur une technologie 40 nm pour les systèmes électroniques à forte intensité de calcul, et la société affirme consommer 15% moins d'énergie et améliorer de 15% ses performances par rapport aux FPGA 40 nm concurrents.

2.3.1.5 Virtex 7

Sont disponibles en -3, -2, -1 et -2 degrés de vitesse, avec -3 ayant la plus haute performance. Les appareils -2L fonctionnent à VCCINT = 1.0V et sont filtrés pour une puissance statique maximale inférieure. La spécification de vitesse de un appareil de -2L est le même que le niveau de vitesse -2 et -2G

La vitesse est disponible dans les appareils utilisant du silicium empilé Technologie d'interconnexion (SSI). La cote de vitesse -2G prend en charge les émetteurs-récepteurs GTH 12,5 Gb / s ou 13,1 Gb / s ainsi que les spécifications de qualité standard -2.[10]

2.4 Architecture générale de FPGA

L'architecture des réseaux de portes programmables (Field Programmable Gate Arrays) peut être vue comme une matrice de blocs logiques programmables entourée par une maille de connexions configurables.

Avec des bloque de configuration et mémoire RAM sur certains circuits est des blocs entré sortie programmables avec aussi générateur d'horloge programmable sont distribuées dans l'ensemble du FPGA par des circuits de routage spécifique.

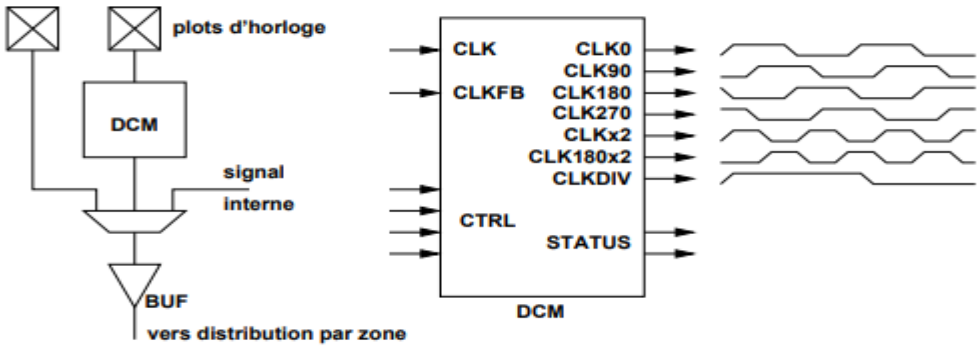


Figure 2.2 : Architecture de générateur d'horloge

2.4.1 Architecture du bloc logique

Les blocs logiques sont distingués par leur granularité, qui peut être définie comme le nombre de portes équivalentes (NANDs avec 2 entrées).

Le bloc logique de base peut varier d'un inverseur simple à une fonction logique à plusieurs entrées. Un élément logique contient plusieurs bloc logique connecté entre aux avec des interconnexions programmables.

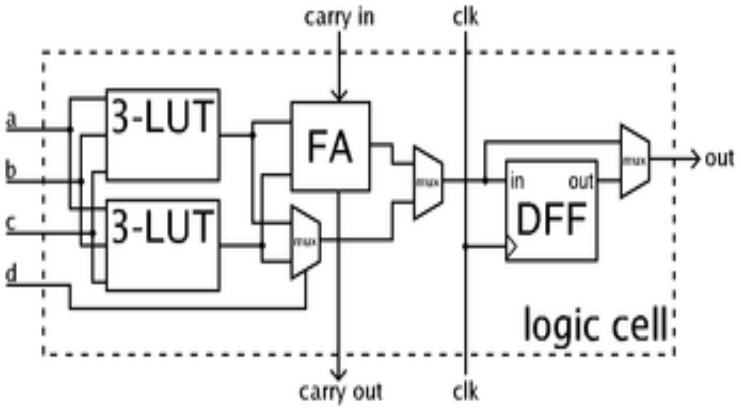


Figure2.3 : Architecture de bloc logique

2.4.2 Architecture des connexions

Les éléments logique regroupe par bloc logique programmable chaque bloc logique a 16 élément logique , dispose 4 PLL de plusieurs multiplexeurs, interface entrée/sortie , plusieurs liens de connections locaux entre les blocs logique , la configuration rapide peut se faire à moindre de 5 ms .

Dispose aussi 4 entrées connecté à la LUT qui permet de représenter n'importe quelle fonction logique a 4 entrés , de registre programmable représenté par des bascules D ,d'une entrée retenue connecté a élément logique précédent d'une sortie retenue connecté a élément logique suivant Un réseau local permet la communication entre les élément logique de même bloc logique

Les liaisons direct permet l'interconnections des blocs logique adjacente ,ils existes aussi en bloc mémoire une signal d'horloge une multiplieur , des entrés sortie entre les blocs adjacente est des réseaux interconnections ligne et colonne entre les bloc logique

L la capacité d’une circuit FPGA n'est pas limité par la complexité des calculs ,mais plus tous par le nombres de bloc logique programmable [11]

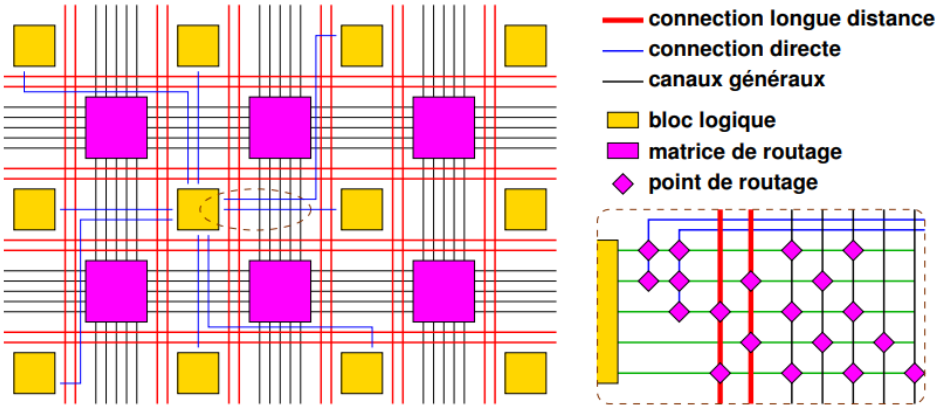


Figure2.4 : Architecture des connexions sur FPGA

2.4.3 LUT

Une table de correspondance (LUT), qui signifie, est en gros une table qui détermine la sortie pour une ou plusieurs entrées. Dans le contexte de la logique combinatoire, c'est la table de vérité. Cette table de vérité définit efficacement le comportement de votre logique combinatoire.

En d'autres termes, tout comportement obtenu en interconnectant un nombre quelconque de portes (comme AND, NOR, etc.), sans chemin de retour (pour s'assurer qu'il est sans état), peut être implémenté par une table de correspondance.

Une cellule logique ou un bloc logique est constitué d’une table correspondance, une table de vérité pour implémenté les équations logiques avec des multiplexeurs, des bascules D est des mémoires.

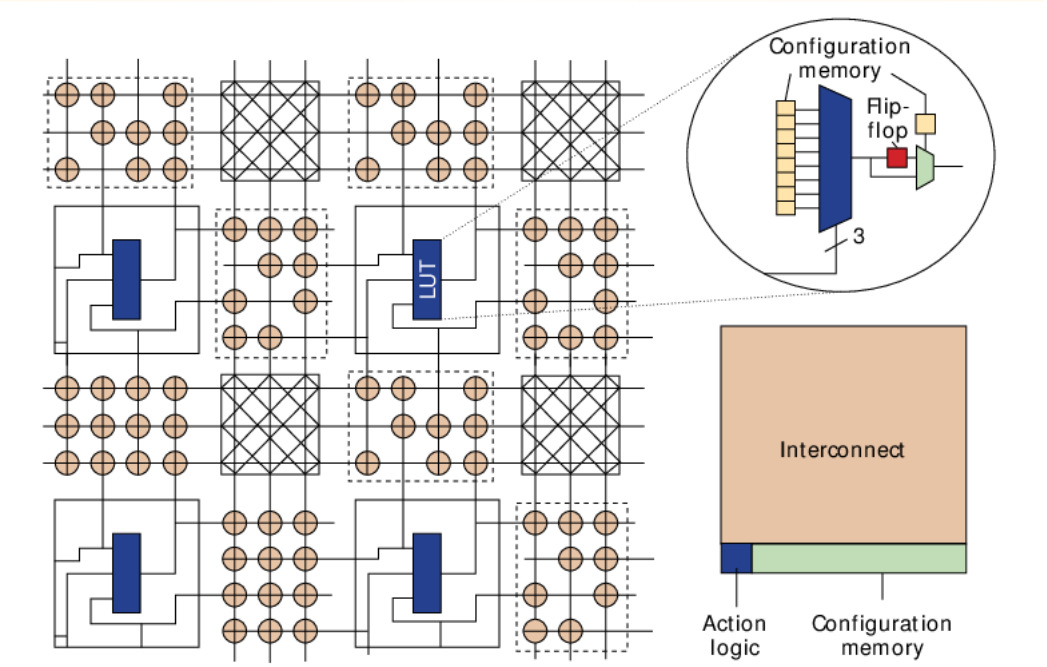


Figure 2.5 : Architecture de LUT sur FPGA

2.4.4 Blocs d'entrée / sortie

Un bloc d'entrée/sortie est l'interface entre les composants extérieurs et la logique interne du FPGA appelé (cœur logique) ('logic core') qui désigne l'ensemble des blocs logiques et du réseau d'interconnexions). Ainsi, lorsqu'il est configuré en entrée, s'il détecte un 0 ou un 1 logique en entrée, il doit véhiculer un 0 ou un 1 logique vers la logique interne du FPGA. Lorsqu'il est configuré en sortie, il doit véhiculer le signal du FPGA vers les composants extérieurs.

Un bloc d'entrée/sortie doit donc adapter les niveaux logiques et les tensions d'alimentation des circuits à l'intérieur et à l'extérieur du composant.

L'idéal serait que tous les blocs d'entrée/sortie soient équivalents et que l'on puisse configurer chaque bloc dans n'importe quel standard indépendamment des autres blocs d'entrée/sortie. Mais la surface consommée serait exorbitante d'autant plus qu'il y a de plus en plus d'entrées/sorties dans les FPGAs et il y a un grand nombre de standards.

les FPGAs modernes possèdent des banques d'entrées/sorties. Chaque banque possède sa propre tension d'alimentation et sa propre tension de référence qui sont partagées

par ses blocs d'entrée/sortie. Ainsi, une seule banque ne peut pas supporter plusieurs standards simultanément. Mais différentes banques peuvent implémenter des standards différents car elles ont des tensions d'alimentation et de référence indépendantes. On peut même mettre hors tension les banques inutilisées pour diminuer la consommation.[11]

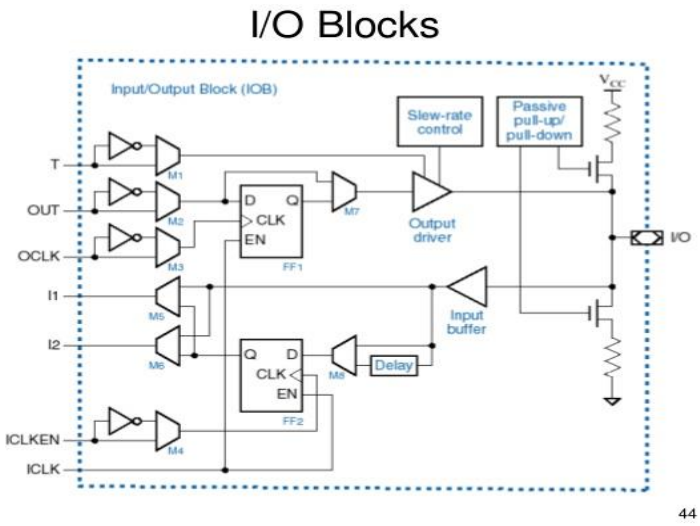


Figure2.6 : Architecture de bloc entrée/sortie sur FPGA

2.4.5 DSP

Les générations les plus récentes de FPGA proposent même des blocs DSP (pour Digital Signal Processing) dédiés, comme leur nom l'indique, au traitement du signal. Ces blocs sont généralement composés d'un multiplieur et d'un accumulateur (MAC, pour Multiplier and Accumulator) et permettent ainsi de réaliser des filtres de manière très efficace.

Dans un système DSP contenant plusieurs étapes de traitement chaque étape peut nécessiter un plage de traitement différente en fonction des exigences de traitement

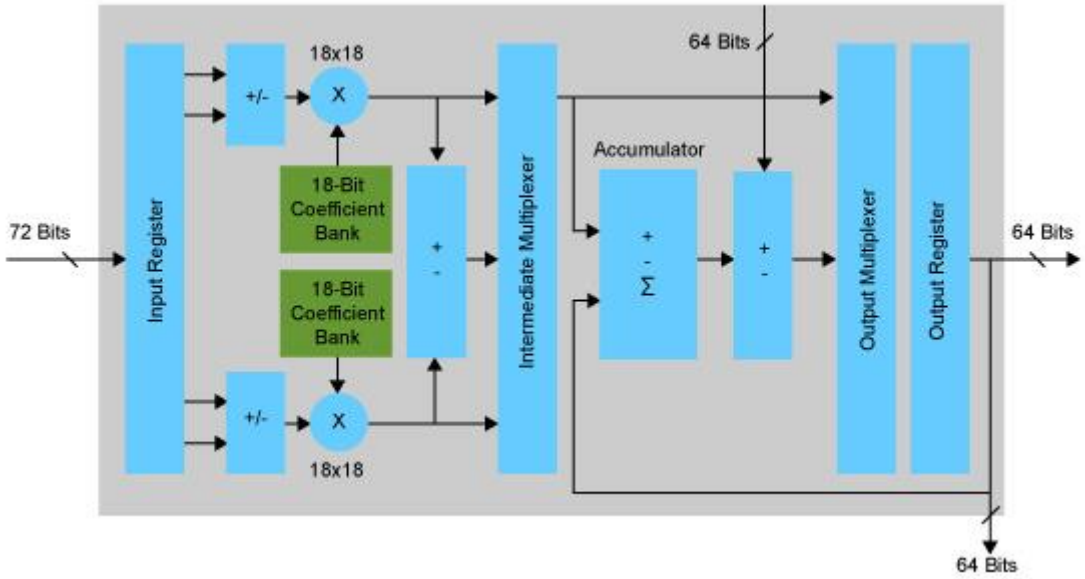


Figure2.7 : Bloc DSP sur FPGA

2.4.6 Générateur d'horloge digital (DCM)

La primitive DCM (Digital Clock Manager) dans Xilinx les parties FPGA sont utilisées pour implémenter la boucle à verrouillage de délai, synthétiseur de fréquence numérique, déphaseur numérique ou un spectre d'étalement numérique. Le gestionnaire d'horloge numérique module est un wrapper autour de la primitive DCM qui permet de l'utiliser dans la suite d'outil EDK.

Le DCM (Digital Clock Manager) permet de Diviser ou multiplier la fréquence de l'horloge , Déphaser l'horloge d'un angle multiple de 90°.

Elimine le « clock skew » = l'horloge arrive en même temps à tous les composants ,Corrige le rapport cyclique

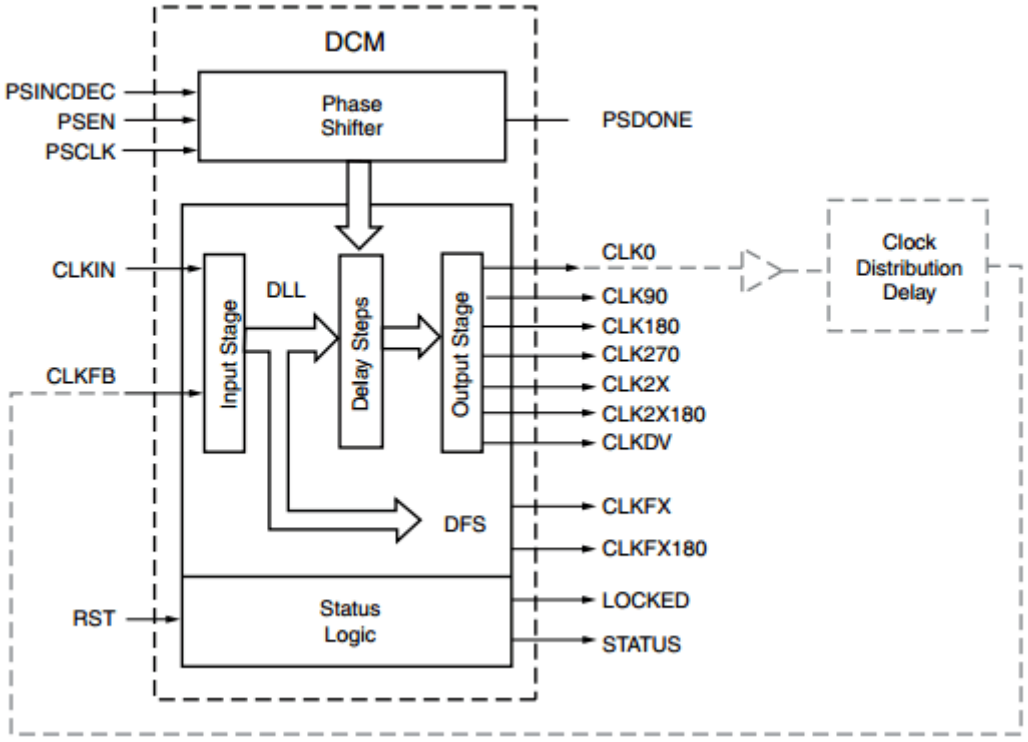


Figure2.8 : Architecture de DCM sur FPGA

2.4.7 RAM

Un bloc RAM est une mémoire dédiée (ne peut pas être utilisée pour implémenter d'autres fonctions comme la logique numérique) à deux ports contenant plusieurs kilobits de RAM. Selon l'avancement de votre FPGA, il peut en exister plusieurs. Par exemple, Spartan 3 a une RAM totale allant de 72 kbits à 1872 kbits. Bien que les périphériques Spartan 6 aient une taille de bloc de 4824 Kbits

C'est une partie discrète du FPGA , ces blocs RAM disponible et limité sur carte FPGA , stockage de grandes quantités de données , la plupart peuvent être initialisés à une valeur différente de zéro lors de l'initialisation du FPGA

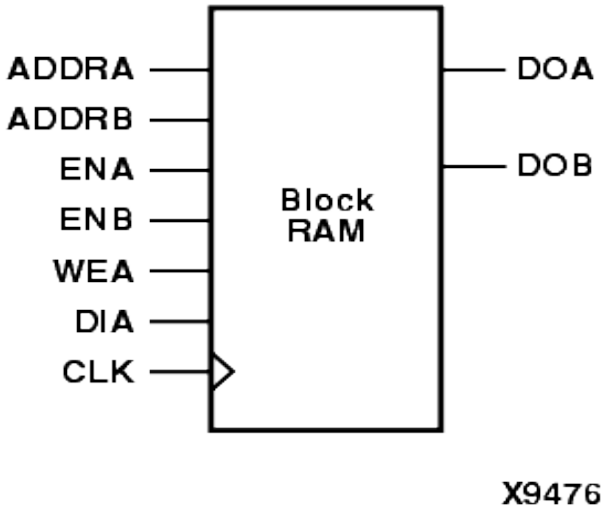


Figure2.9 : Bloc RAM X9476

RAM X9476 par exemple contient des entrés CLK qui représente Horloge à bord positif et ADDRA et ADDRB Adresse d'écriture et Adresse de lecture double dans l'ordre avec DOA Port de sortie primaire DOB Double port de sortie

2.4.8 ROM

Les ROM (Read Only Memory): Mémoires figées par le concepteur a lecture seule et non modifiables.

ROM fournit un décodage complet de la variable

chaque intersection (point de croisement dans la ROM est souvent implémenté avec un fusible

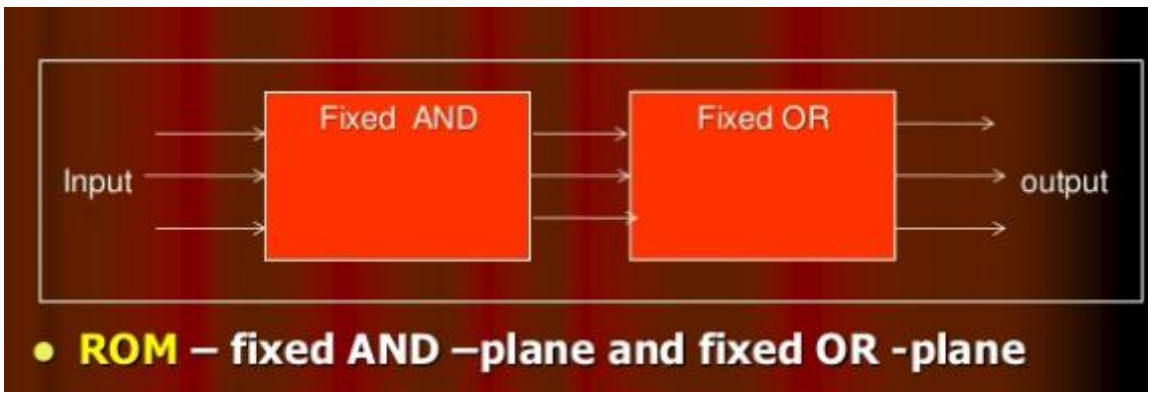


figure2.10 : Architecture de ROM sur FPGA

2.5 Description de FPGA

Plusieurs niveaux de description peut être utilisé dans la conception des circuits FPGA , nous avons

- **le niveau de la description comportementale** ou le modèle d'écrit par fonction , son algorithme, s'agit de décrire comment cela fonctionne .
- **le niveau description RTL**(Register Transfert logic) le modèle et d'écrit sous fourme d'élément séquentiels prend en compte la notion d'horloge de cycle
- **le niveau de description structurel** ,le modèle décrit par sa technologie de port logique permettent de réalise les circuits

nécessite de faire la décomposition du cahier des charges en fonction simple permettent utilisation des fonctions combinatoires avec les instructions concurrent de VHDL est des fonction séquentielles avec les process. [12]

2.6 Méthodologie de Conception de circuit FPGA

2.6.1 La spécification de circuit :

ils s'agit de la saisie du circuit logique peut se faire de manière syntaxique avec

Le VHDL ou verilog ou mode graphique avec le niveau description structurelles

2.6.2 La validation fonctionnelle

qui consiste à la vérification des syntaxes, la vérification des schémas est simulation fonctionnelle

2.6.3 la validation temporelle des simulation temporelle

qui compte des temps de propagation est de problème de recouvrement des signaux .

2.7 Implémentation Sur FPGA

Implémentation sur cible FPGA préside assignation des pins qui permette d'associer les ports composants de circuit numérique à synthétiser ou pins de la carte cible .

Le synthèse de circuit numérique avec les cartes FPGAs ça pue sur les deux principale de développement, nous avons Altera Quatus 11 et Xlinx ISE

Ces deux logicielles développé par les deux constructeurs (Altera, xilinx) permettre la conception ,la simulation ,la synthèse et aussi l'implémentation des circuits numérique

Ils permettes la spécification a mode graphique est par description HDL

Les simulations ce fond sera avec modèle SIM qui un environnement de simulation et débogage pour les circuits logique programmable, il support plusieurs langage Dans VHDL .(noir)

2.8 Conclusion

Dans ce chapitre nous avons présenté les principes des circuits à logiques programmables de type FPGA. Nous avons étalé leurs approches de configurations, de programmation et d'implémentation. Les méthodes et les architecteurs de calcules arithmétiques et logiques, sur FPGA, sont étalés avec leurs composants reliés.

Les méthodes de développements des projets sur circuits FPGA sont aussi introduites en visons les étapes de synthèse, de vérification et de simulation. Les circuits FPGA de Xilinx sont principalement visés.

Chapitre 3 :

Synthèse générateurs de bruit gaussien

Chapitre 3 Synthèse du générateur de bruit Gaussien

3.1 Introduction

La première étape de synthèse du générateur de bruit gaussien, réclame utilisation d'un outil logiciel de calcul très performant. L'outil de calcul doit produire les vecteurs à virgule flottante pour les deux fonctions f et g de la méthode Box-Muller, doit générer les versions à virgule fixe et doit avoir la possibilité d'analyse des résultats.

Il faut noter que les versions à virgule fixe des deux fonctions f et g servent aussi dans la création des fichiers VHDL ROM pour implémentation sur de FPGA.

Le MatLab et le System Genrator de Xilinx sont les outils incontournables pour notre synthèse.

3.2 Matlab

MATLAB (« *matrix laboratory* ») est un langage de programmation de quatrième génération émulé par un environnement de développement du même nom ; il est utilisé à des fins de calcul numérique. Développé par la société The MathWorks, MATLAB permet de manipuler des matrices, d'afficher des courbes et des données, de mettre en œuvre des algorithmes, de créer des interfaces utilisateurs, et peut s'interfacer avec d'autres langages comme le C, C++, Java, et Fortran. Les utilisateurs de MATLAB (environ un million en 2004) sont de milieux très différents comme l'ingénierie, les sciences et l'économie dans un contexte aussi bien industriel que pour la recherche. Matlab peut s'utiliser seul ou bien avec des *toolboxes* (« boîte à outils »).

3.2.1 L'interface de Matlab

- **Command Window** : Permet d'exécuter des commandes en dehors de programme et affiche les résultats
- **Current Directory** : Contenu du répertoire courant où doit se situer vos programmes.
- **Workspace** : Affiche l'ensemble des variables utilisées.
- **Commande History** : Permet de visualiser les dernières commandes exécutées. Il est possible de faire glisser ces commandes vers la fenêtre de commande. On peut aussi y accéder en appuyant sur flèche haut ou pour des commandes plus anciennes en tapant la première lettre de l'expression puis flèche haut
- **Choix du répertoire courant** : c'est le dossier où doit se situer vos programmes (fichiers *.m). Vous pouvez mettre vos programmes dans un autre dossier mais dans ce cas il faut l'inclure dans File >> Set Path [13]

3.2.2 Simulation dans MatLab

Simulink est l'extension graphique de MATLAB permettant de représenter les fonctions mathématiques et les systèmes sous forme de diagramme en blocs, et de simuler le fonctionnement de ces systèmes.

3.3 La fonction F

C'est une fonction logarithmique avec un variable aléatoire *x1* de valeur entre [0 1], le résultat de fonction f c'est des valeurs aléatoire aussi parce que les valeurs de x1 aléatoire

$$f(x1) = \sqrt{-\ln(x1)}$$

En peut faire un petit programme matlab qui fait calcul la fonction f (x1)

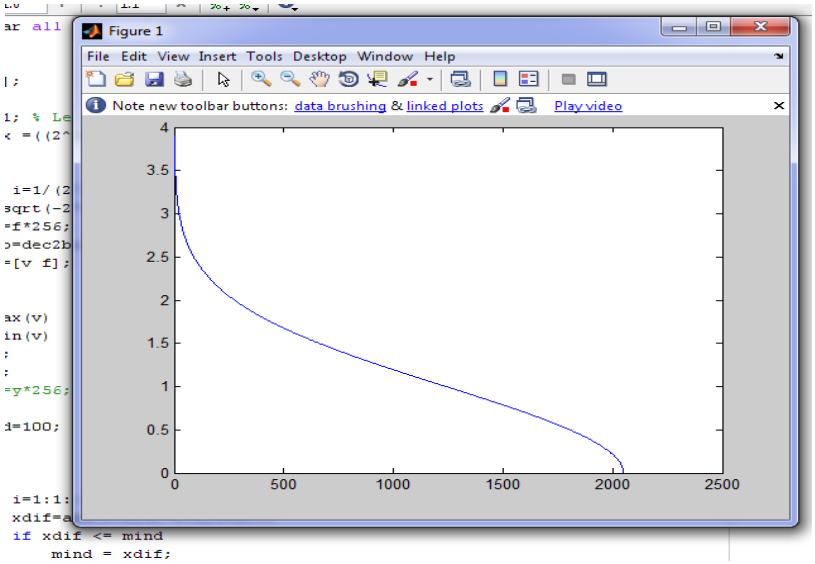


Figure3.1 : La fonction f

3.4 La fonction g

C'est une fonction cosinus ou bien sinus dans notre travail nous allons utiliser la fonction cosinus avec une variable aléatoire x2 de valeur entre [0 1]

$$g(x2) = \sqrt{2}\cos(2\pi x2)$$

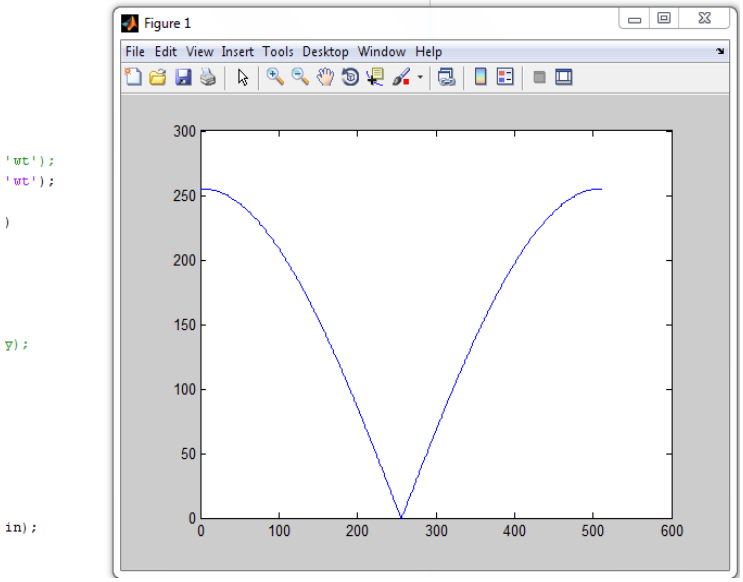


Figure3.2 : La fonction g

Remarque : le cosinus contient deux symétries, positive et négative.

Dans notre travail nous avons utilisé le résultat d'une demi-période multiplié avec un bit signe aléatoire.

3.5La méthode de Box-Muller

Simulations logicielles. Elle consiste à générer la v.a. N(0,1) à partir de deux v.a. réelles indépendantes x1 et x2 uniformément distribuées sur [0, 1] en utilisant deux fonction F et G qui nous donnent des valeurs aléatoire normales n, avec :

$$n = f(x1) * g(x2)$$

Les valeurs obtenues de N c'est des valeurs aléatoires.

3.6 Numérisation de fonction F et G

Dans notre travail nous avons utilisé des fonctions f et g pour élaborer un générateur de bruit gaussien avec la méthode de Box-Muller dans MatLab avec les étapes suivantes :

- Fixé σ
- Fixé le nombre de bit de bus d'adresse
- Fixé le nombre de bit de bus data
- La conversation de fonction f et g décimal / binaire est l'inverse (virgule flottent et virgule fixe)

Dans la numérisation des fonctions f et g, il faut que tous les valeurs de f(x1) et g(x2) seront codés par des codes non nuls. Au moins un (01) bit est utilisé pour le codage de la plus petite valeur. C'est pour cette logique qu'il faut fixer le nombre de bits du bus d'adresse et du bus data et pour les utiliser dans la phase de génération des fichiers VHDL dans la partie de création de fichier ROM

Dans notre travail nous nous intéressons plus à la fonction f.

3.6.1 Valeur max de segma σ

Pour calculer le nombre de bit K de bus d'adresse mathématiquement il faut fixer la valeur de **δ.** Par exemple dans une population de variables aléatoires Gausiennes, plus de 99% des chiffres soit inferieurs a 4. Ainsi nous fixons **l**a valeur max de σ à la valeur 4.

3-6-2 Nombre de bit k du bus d'adresse

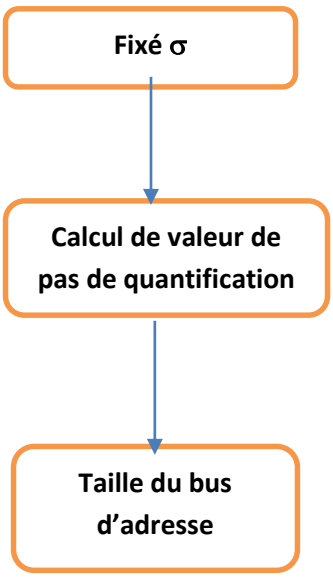
Avec la fonction F, nous calculons de nombre d'échantillon ou bien le pas de quantification. De ce fait, nous fixons la valeur de et nous calculons la valeur de x1 avec :

- $\boldsymbol{f} = \sqrt{-\ln(x1)}$
- $\sigma = \sqrt{-\ln(x1)}$
- $\sigma^2 = -2\boldsymbol{ln}(\boldsymbol{x1})$
- $\boldsymbol{x1} = \boldsymbol{e}^{(\sigma^2)/2}$

Apres, nous calculons la taille de bus d'adresse de K bits avec

$k = \log_2(1 / x_1)$

Dans notre cas, nous avons k= 11 bis



3.6.3 Nombre de bit du bus data

Après le choix de bits k, du bus d'adresse, on cherche le nombre de bit de bus data à partir de la fonction f.

Le nombre de bit de bus data utilisé pour générer le fichier VHDL dans la partie de création de fichier ROM d'une fonction f et g

Pour calculer le nombre de bits data il faut :

- Fixer le nombre de bits du bus d'adresse K= 11 bits et le pas de quantification pas = ½^k

3.6.3.1 Création des vecteurs avec virgule flottante et fixe

Dans notre projet nous utilisons les virgules fixe, alors il faut d'abord crée un vecteur en virgule flottante

V= [v f], V le vecteur de virgule flottant qui s'appelle y [N] = f(J) avec un boucle for dans un programme MatLab et avec un compteur de pas

Avec J=½^k

Y[N] : le vecteur de virgule flottante

- la condition pour que la boucle faire d'autres itération N<Nmax

Nmax la valeur max de pas de quantification

Avec N=N+1 et J=J+1

Sinon, si N≥ Nmax la boucle arrête

après les itération et la création de vecteur y[N] , il faut calculer la valeur max et la valeur min de vecteur y pour utiliser la petite valeur ymin. La condition pour que la boucle while faire d'autre itération :

$$ymin * 2^n <= 1$$

Ymin : la valeur minimale de vecteur y

Avec n=n+1

Si non il y a d'autre condition :

$$ymin * 2^n > 1$$

$$2^m < ymax * 2^n$$

Ymax= la valeur max de vecteur y

Avec m=m+1

Dans ce cas

$$y1 = y * 2^n()$$

Et

$$ybin = dec2bin(y1) \quad ()$$

avec ybin de la valeur binaire de vecteur y1 en virgule fixe

La déférence entre y et y1 le vecteur y c'est un vecteur en virgule flottant, et y1 c'est un vecteur en virgule fixe

remarque : Tout ce que nous avons fait précédemment s'applique parfaitement a la fonction g (même programme mais deux fonction déférence).

3.7 Création de ficher ROM

Pour crée un fichier ROM il faut utiliser le vecteur y1 en virgule fixe et utilisé un programme Matlab avec les étapes suivantes :

- création de boucle for
- convertir les valeurs en binaire
- stocké les valeurs obtenue dans un fichier texte

✓ **<u>la 1er étape</u>**

- création de boucle for pour le vecteur y1 avec un

pas= 1/ 2^K et { j=j+1;
x(j)=j

✓ **la 2eme étape**

- après les calculs du vecteur y1 dans la 1er étape il faut convertir les valeurs décimales en virgule fixe avec dec2bin(y1). La petite valeur, que nous avons calculée, ymin est au moins codée sur un (01) bit, pour que toutes les valeurs possibles de y1 seront des codes binaires non nuls.

✓ **la 3eme étape**

- stocké les valeurs obtenue dans un fichier texte (avec un programme MatLab).

Le fichier ROM obtenue en décimale pour f

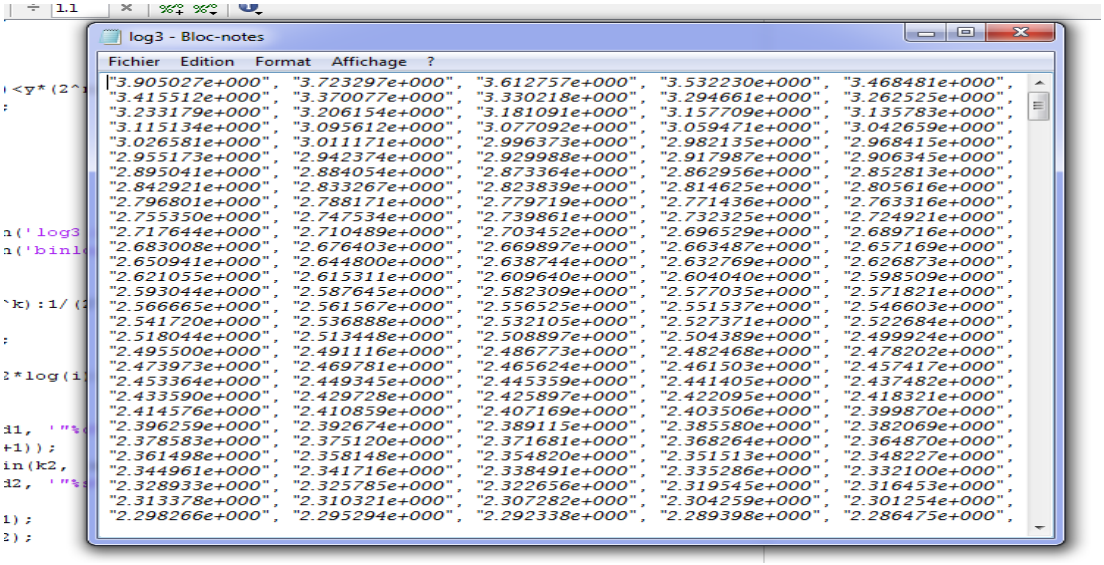


Figure3.3 : fichier ROM décimale

Le fichier ROM obtenue en forme binaire de fonction f

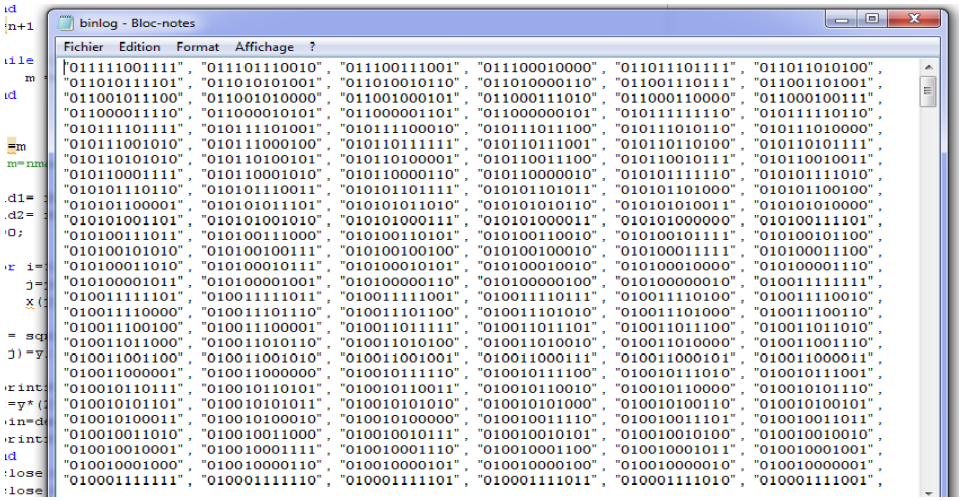


Figure3.4 : fichier ROM binaire

- **Utilisation des fichiers ROM (Matlab)**

Après la création des fichiers ROM sur Matlab, c'est des fichiers textes. On va utiliser ces fichiers dans des descriptions VHDL pour la partie implémentation sur FPGA

3.8 Architecture de LFSR

Un registre à décalage à rétroaction linéaire, ou LFSR (acronyme de l'anglais linear feedback shift register), est un dispositif électronique ou logiciel qui produit une suite de bits qui peut être vue comme une suite récurrente linéaire sur le corps fini F_2 à 2 éléments (0 et 1).

Dans le cas particulier d'une suite de 0 et de 1, c'est un registre à décalage avec rétroaction linéaire, ce qui signifie que le bit entrant est le résultat d'un OU exclusif (ou XOR) entre plusieurs bits du registre.

3.8.1Fonctionnement

Un LFSR est un dispositif dérivé du registre à décalage de type SIPO, Serial In - Parallèle Out, dans lequel un ou plusieurs « étages » du registre subissent une transformation pour être réinjectés en entrée. 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. Voir de figure ci-dessus

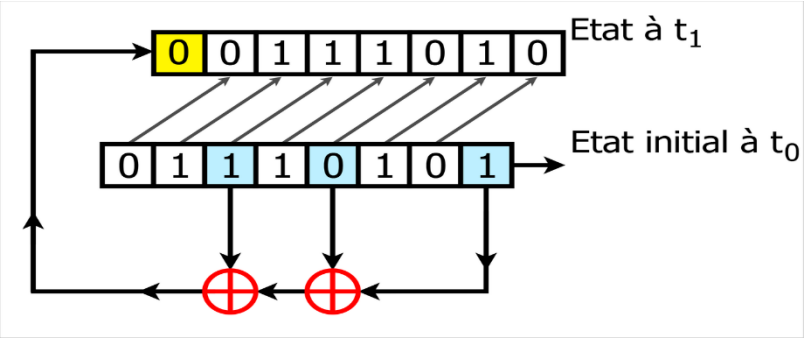


Figure3.5 : LFSR à 8 bit a t=0 et t=1

Horloge	Etat DE LFSR	Sortie
t=0	11001010	1
t=1	11100101	0
t=2	01110010	1
t=3	10111001	1
t=4	11011100	1

Tableau 3.1 : représente un exemple de LFSR de 8 bits à des instant t et le résultat en sortie de chaque état

Le polynôme de LFSR à 8 bits

3.8.2 Architecture

Le registre à décalage à rétroaction linéaire est mis en œuvre sous la forme d'une série de bascules électroniques à l'intérieur d'un FPGA qui sont câblées ensemble sous la forme d'un registre à décalage. 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)

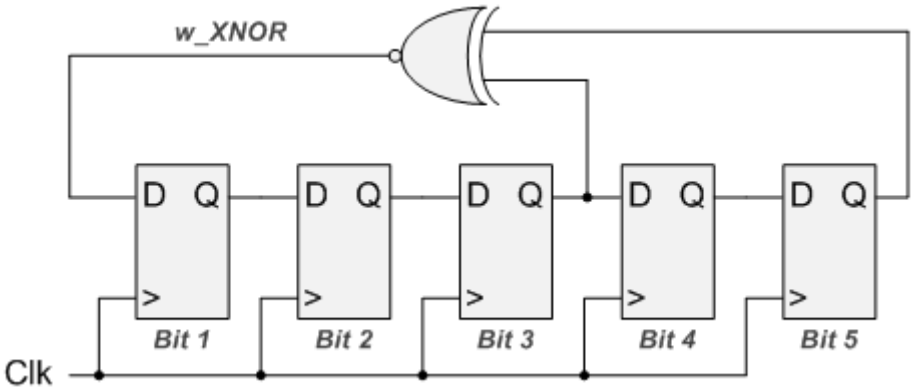


Figure3.6 : architecture de LFSR à 5 bits

Remarque : Nous utilisons LFSR dans Simulink de matlab et en FPGA avec une description VHDL, les fichiers ROM que nous avons créés dans la partie de création de fichier ROM [14]

3.9 Conclusion

Dans ce chapitre nous avons utilisé le logiciel Matlab pour numérisé les fonctions f et g de virgule flottant a virgule fixe est créé à partir de ces fonction des fichiers ROM textes pour les utiliser dans les descriptions VHDL des ROM implémentables sur FPGA.

Chapitre 4 :

Architecture et implémentation sur FPGA

Chapitre 4 Architecture et implémentation sur FPGA

4.1 Introduction :

La dernière étape pour compléter notre travail consiste à utiliser le Langage de description de matériel VHDL et un logiciel Xilinx ISE. La description VHDL est utilisée pour synthétiser l'architecture globale de notre générateur de bruit Guaussien. Le System Generator et le ISE de Xilinx sont utilisés pour la valiation et le test.

Nous comparons les résultats de notre générateur avec les résultats du Matlab, à virgule flottante, et les résultats du System Generator, à virgule fixe. Nous concluons le chapitre par une conclusion.

4.2 Langage de description de matériel VHDL :

4.2.1 Définition :

Le VHDL est un langage de description matériel destine à représenter le comportement ainsi que l'architecture d'un système électronique numérique .son nom complet est : VHSIC (VHSIC : very high speed integrated circuit). L'intérêt d'une telle description réside dans son caractère exécutable : une spécification décrite en VHDL peut être vérifiée par simulation, avant que la conception détaillée ne soit terminée. En outre, les outils de conception assistée par ordinateur permettant de passer directement d'une description fonctionnelle en VHDL à un schéma en porte logique ont révolution les méthodes de conception des circuits numériques comme les FPGA.

4.2.2 Logiciel Xlilinx ISE 14.7 :

L'offre logicielle dans le domaine de conception des circuits numérique est très varié et l'un de ces environnement que nous allons exploiter au cours de travail est le Xilinx ISE qui est un logiciel de création et de gestion de projet pour les circuits FPGA. C'est un logiciel multi taches qui possède dans son soft de différents outils permettant la création de système sous circuits numériques, l'introduction de projets de manière textuelle ou graphique en vue d'une intégration dans un circuit logique programmable.

Ce logiciel ISE de Xilinx permet la simulation de la description et la synthèse du circuit logique équivalent puis le placement et le routage du circuit sur un prototype correspondant à une technologie FPGA bien précise et enfin lorsque toutes les vérification sont faites vient l'implantation sur un FPGA réel ce qui correspond à générer le fichier de configuration du circuit cible choisi afin d'établir les interconnections des cellules logiques correspondantes au circuit logique conçus avec optimisation de ressources disponibles au niveau de circuit programmable FPGA

D'une manière générale, le ISE de Xilinx permet de réaliser toutes les étapes de conception et de programmation des circuits FPGA de Xilinx et même pour d'autres circuits programmables tel que les CPLD. La conception de circuits sur Xilinx ISE met en œuvre 4 outils :

Un éditeur de texte ou entrée graphique : permet la description dans les logiciels CAO c'est-à-dire de dessiner ou de décrire le circuit avec une interface graphique ou textuelle.

Un simulateur : la simulation du système est faite pour vérifier la validité du code avant-synthèse, après synthèse et placement – routage.

Un synthétiseur : l'étape de synthèse et routage succèderons par la suite ou la synthèse consiste à faire la transcription de la de description d'une forme texte vers une graphique RTL à base de portes logiques.

Un placeur routeur : cette étape est une adaptation du circuit logique synthétisé sur les ressources disponibles dans le circuit FPGA cible. [15]

4.2.3 Utilité du vhdl :

Le vhdl est un langage de spécification, de simulation et également de conception :

- Il est tout à fait possible de décrire un circuit en un VHDL standard pour qu'il soit lisible de tous ;
- Il est également un langage de simulation. La possibilité de simuler avec des programmes vhdl devrait considérablement faciliter l'écriture de tests avant la programmation du circuit et éviter ainsi de nombreux essais sur un prototype qui sont beaucoup plus couteux et dont les erreurs sont plus difficiles à trouver ;
- Il permet la conception de circuits avec une grande quantité de portes. Il est compatible avec les circuits actuels qui comprennent, pour les FPGA par exemple, entre 500 et 100000 portes et ce nombre augmente rapidement.[16]

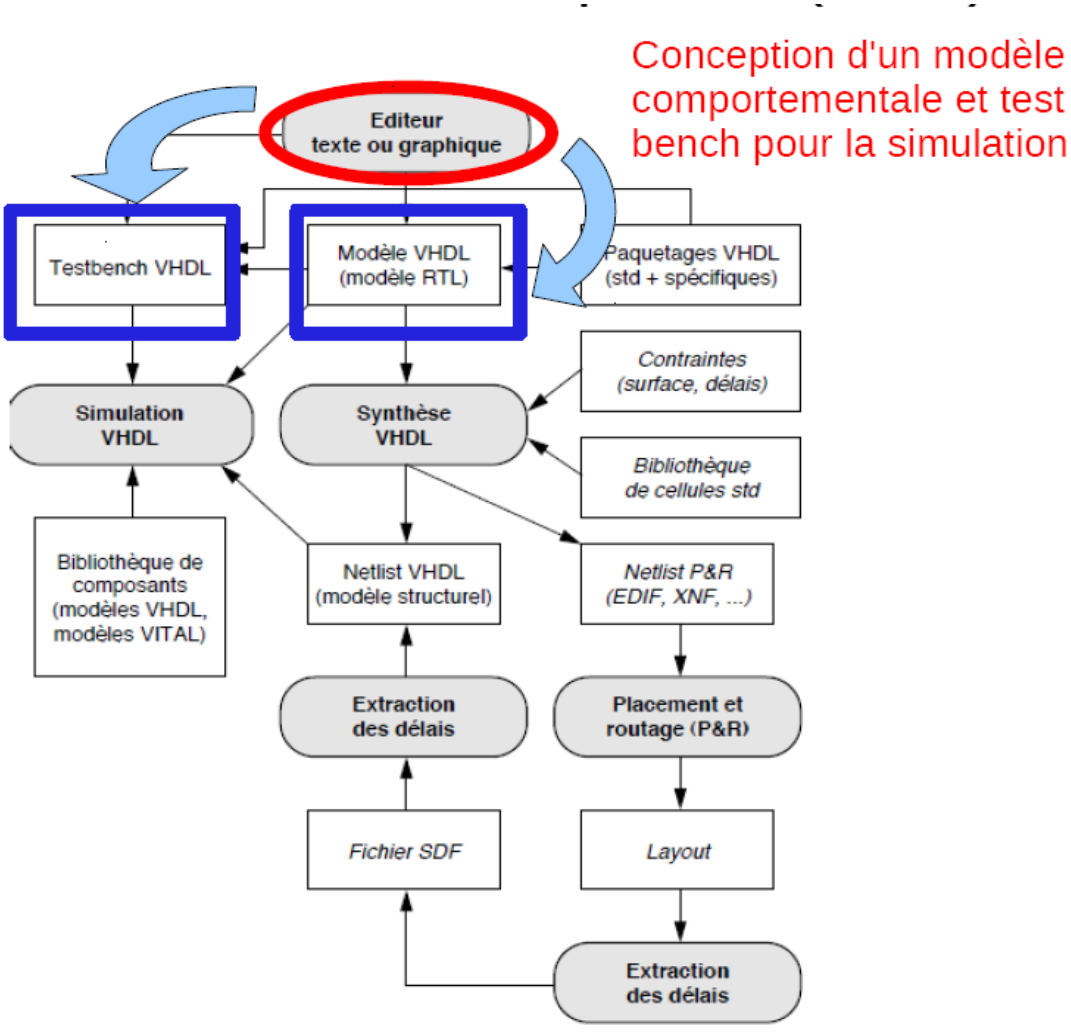


Figure 4.1 : Flote de conception VHDL

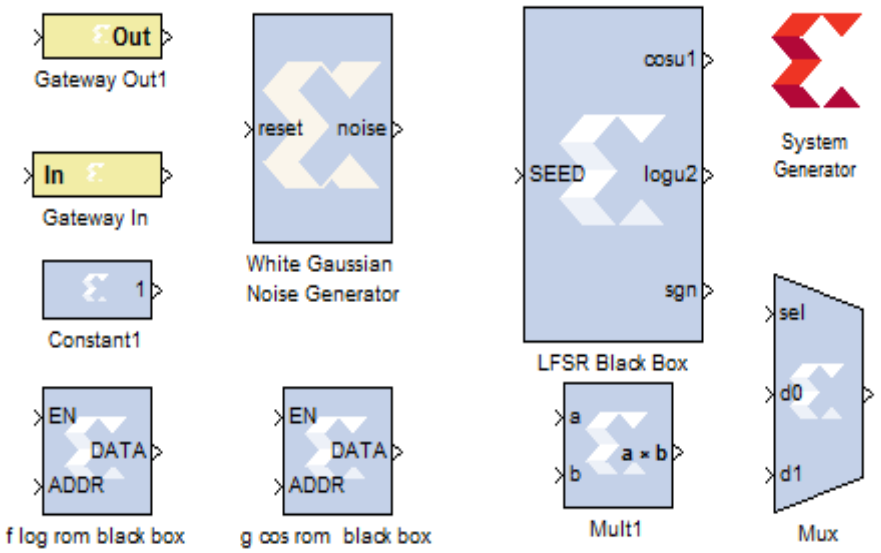


Figure 4.2 : Blocs de xilinx

4.3 Architecteur de générateur Box-Muller proposé :

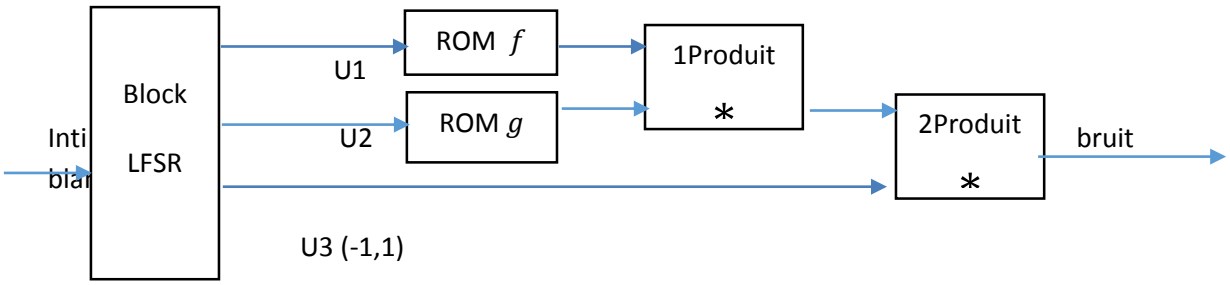


Figure4.3 : Architecteur de générateur Box-Muller

Le générateur Box-Muller proposé dans notre travail comprend les éléments suivants :

- un registre à décalage à rétroaction linéaire (LFSR) qui générer des variable aléatoire uniforme

-U1 et U2 deux variables aléatoires indépendantes uniformément distribuées dans]0,1].

-U3 bit sign

-Rom de fonction f

-Rom de fonction g

Ensuite, nous multiplions les fonctions f et g pour obtenir le bruit blanc.

4.4 Synthèse des blocks des fonctions f et g

Les fonctions f et g sont des ROM Développées par VHDL

4.4.1 Block ROM f :

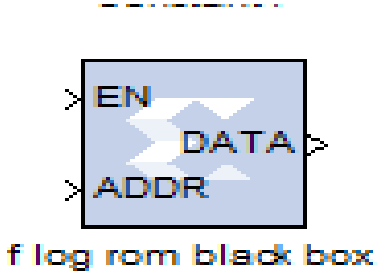


Figure4.4 : bloc ROM f

Contient :

Synthèse VHDL

- 5 ports (4 entré et 1 sorti)
- 4 entré : Clk, EN, buse d'adresse 2^11bits
- Sorti : DATA 9 bits

Rom**f** := («111110011" , "111011100" , "111001110" ……) buse d'adresse f

Buse d'adresse K : Qui nous avons calculé dans le chapitre 3

Ensuite

rdata <= ROM(conv_integer(2047-ADDR));

process (C)

begin

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

if (CE = '1') then

if (EN = '1') then

DATA <= rdata;

Syntexe et sumulation avec s.g .

la taille de ROM F est 2kbits

4.4.2 Block ROM g :

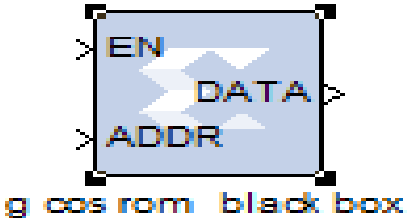


Figure4.5 : bloc ROM G

Contient :

Synthèse VHDL

- 5 ports (4 entré et 1 sorti)

- 4 entré : Clk , EN , buse d'adresse 2^9bits

- Sorti : DATA 9 bits

Rom**g** := («111110011" , "111011100" , "111001110" ……) buse d'adresse g

Buse d'adresse g : Qui nous avons calculé dans le chapitre 3

Ensuite

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;
```

la taille de ROM F est 0.5k bits

4.5 Synthèse et simulation avec system generator :

4.5.1 System Generator (SG) de Xilinx

Le System Generator de Xilinx permet de contrôler les paramètres du système et la simulation, et est utilisé pour appeler le générateur de code. Tous les modelés Simulink contenant les blocs de xilinx doivent contenir au moins un bloc SG. Une fois ce dernier ajouté à un modèle, il est possible de spécifier la manière dont la génération de code et de simulation doivent être manipulée.[17]

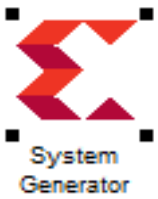


Figure4.6 bloc system generator

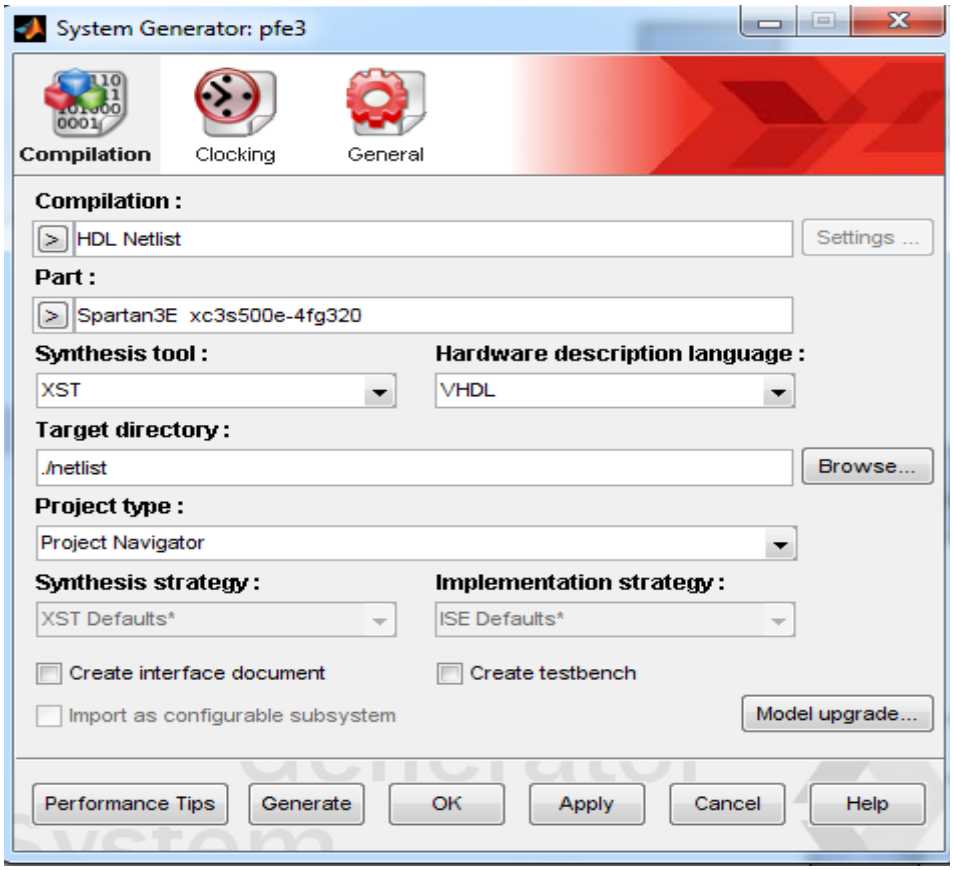


Figure 4.7 : boite de dialogue de bloc System Generator

4.5.2 La précision et les passerelles xilinx :

SG fonctionne avec les modèles Simulink standard. Deux bloc appelés'' Gateway In'' et ''Gateway Out'' définissent la limite du FPGA du modèle simule en Simulink .la passerelle dans le bloc convertit l'entrée à un nombre a virgule fixe .[34]

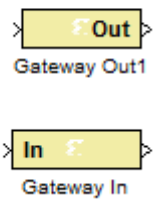


Figure4.8 : Bloc Gateway In et Out

4.5.3 Simulation d'un générateur de bruit Gaussienne avec SG :

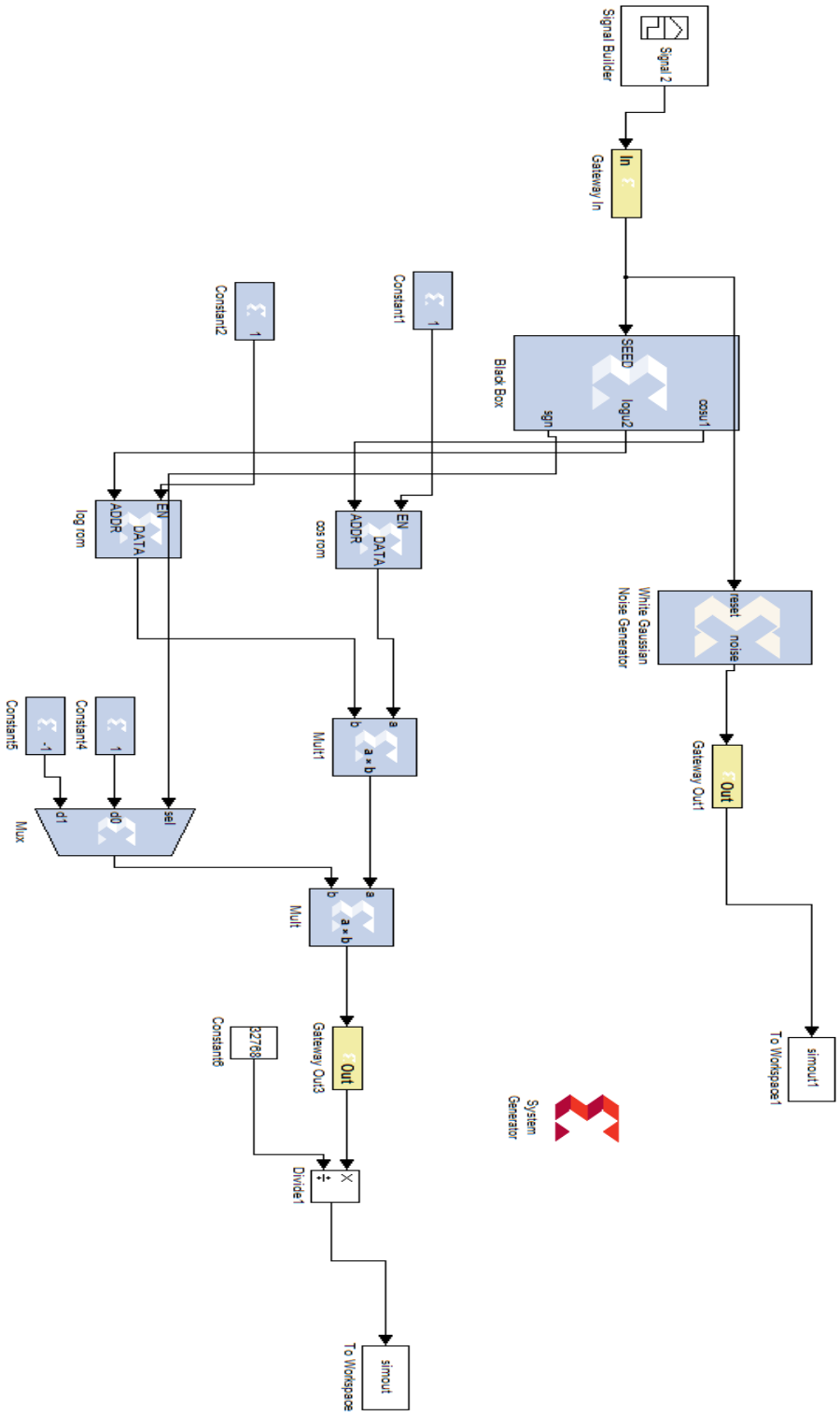


Figure 4.9 : Simulation d'un générateur de bruit Gaussienne par la méthode de Box-Muller

Le bloc LFSR : 52 bits

4.5.4 Les résultat obtenu de simulation :

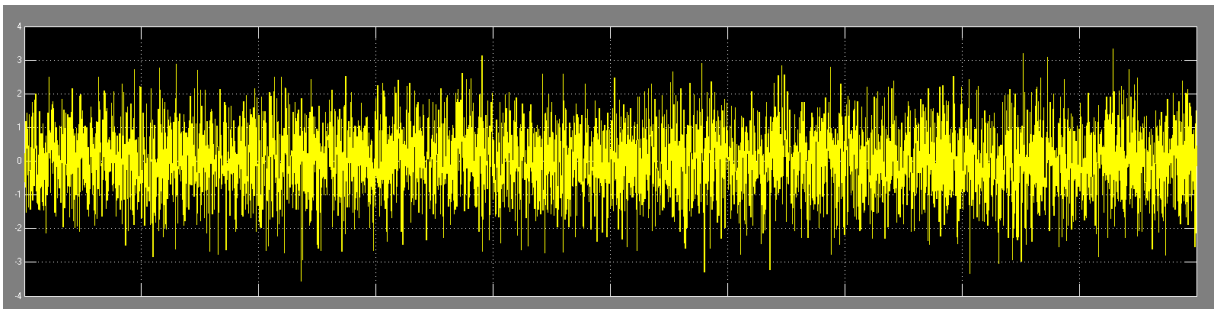


Figure4.10 : bruit blanc gaussien de SG

Remarque : cette figure représente un bruit blanc de SG .

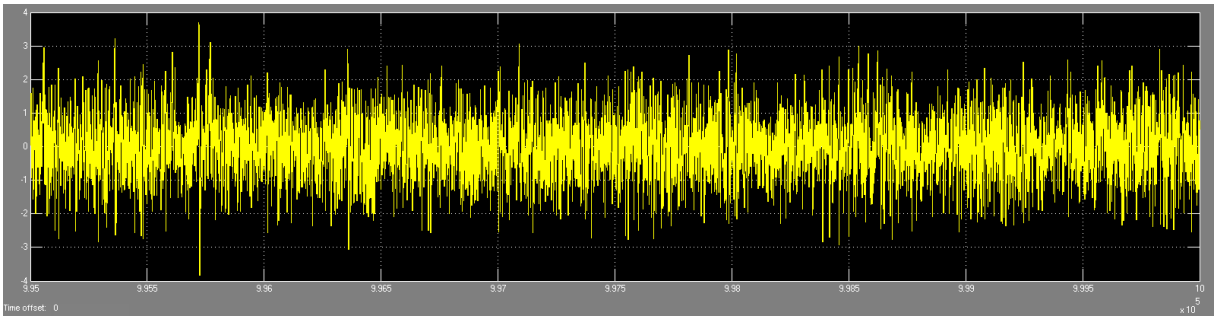


Figure4.11 : bruit blanc gaussien de BOX-Muller

Remarque : cette figure représente un bruit blanc de Box-Muller .

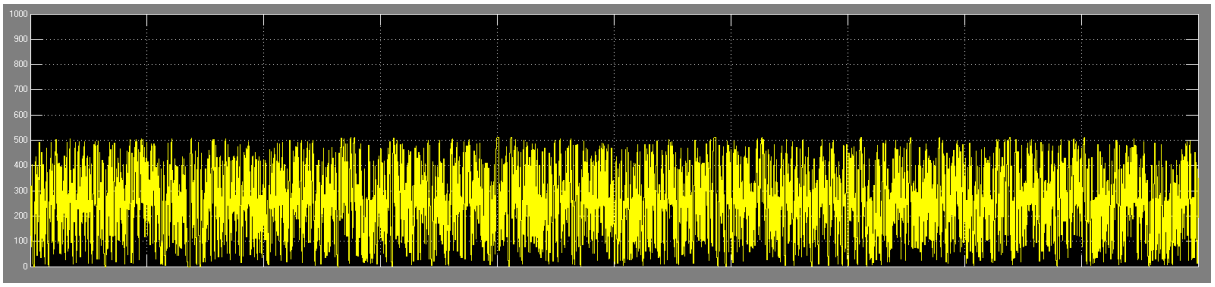


Figure4.12 : les variables aléatoires de LFSR g

Remarque : cette figure représente les variables aléatoires uniforme de LFSR g

[0 ….512] valeurs

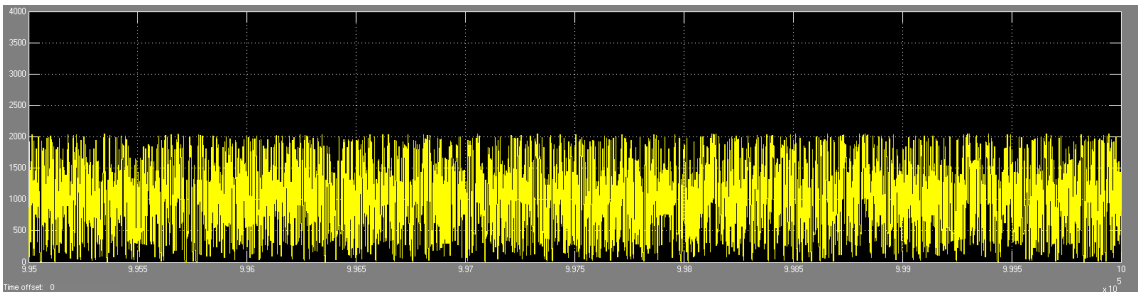


Figure4.13 : les variables aléatoires de LFSR f

Remarque : cette figure représente les variables aléatoires uniforme de LFSR f [0 ….2048] valeurs

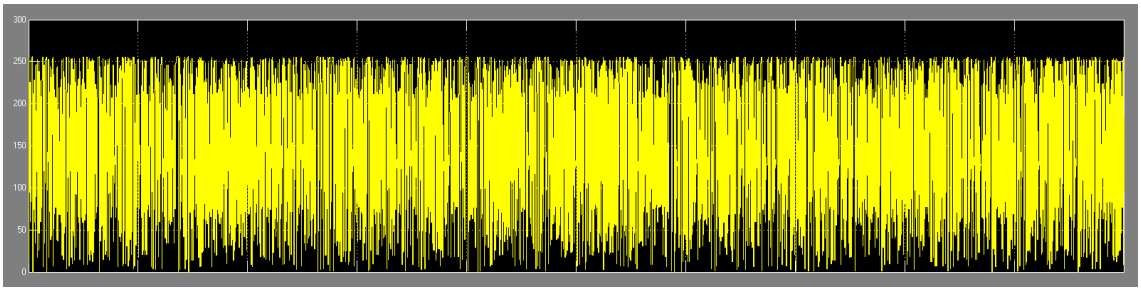


Figure 4.14 : ROM g

Remarque : cette figure représente la sortie de ROM g

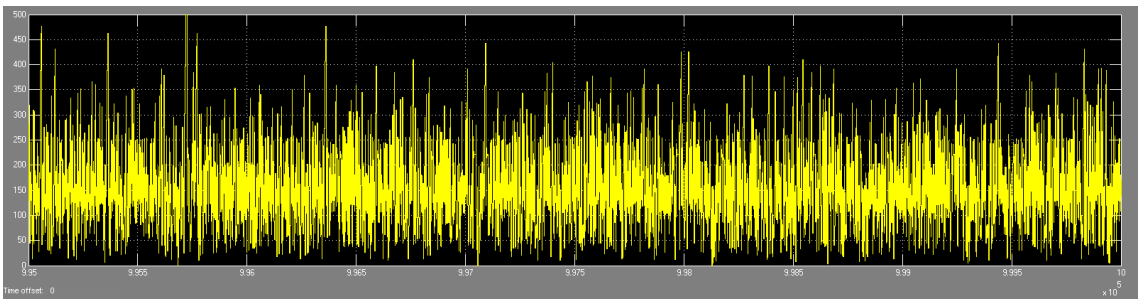


Figure 4.15 : ROM f

Remarque : cette figure représente la sortie de ROM f

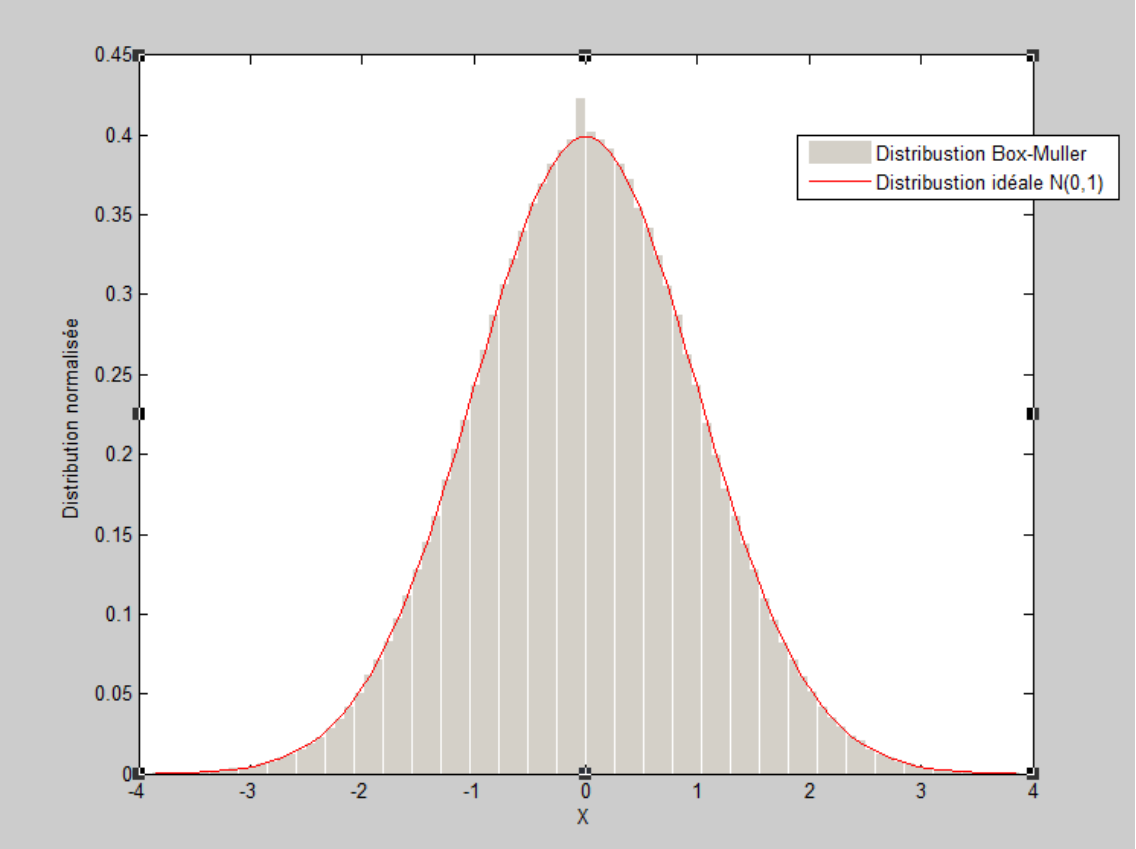


Figure4.16 : Distribution d'une v.a gaussienne de Box-Muller

Remarque : cette figure représente la distribution de 1 million échantillon d'une v.a gaussienne générée par notre approche avec la méthode Box-Muller et une distribution idéale normal N (0 1).

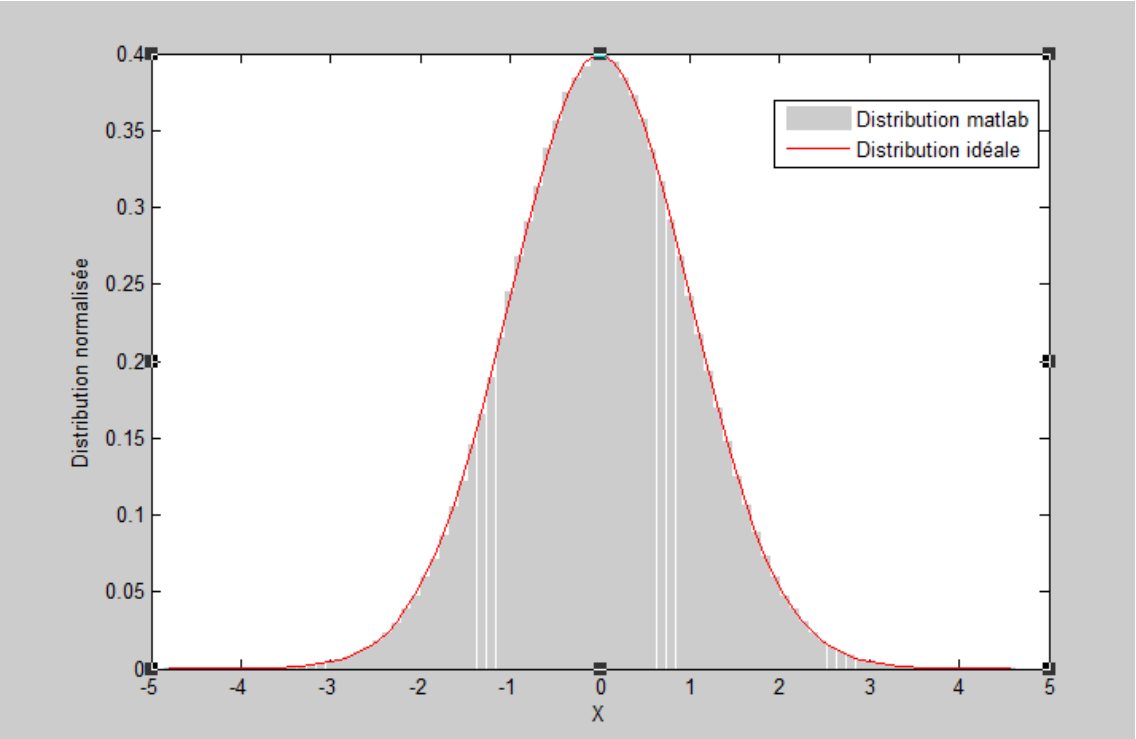


Figure4.17 : Distribution d'une v.a gaussienne de Matlab

Remarque : cette figure représente la distribution de 1 million échantillon d'une v.a gaussienne générée par matlab et la distribution idéale normal N (0 1).

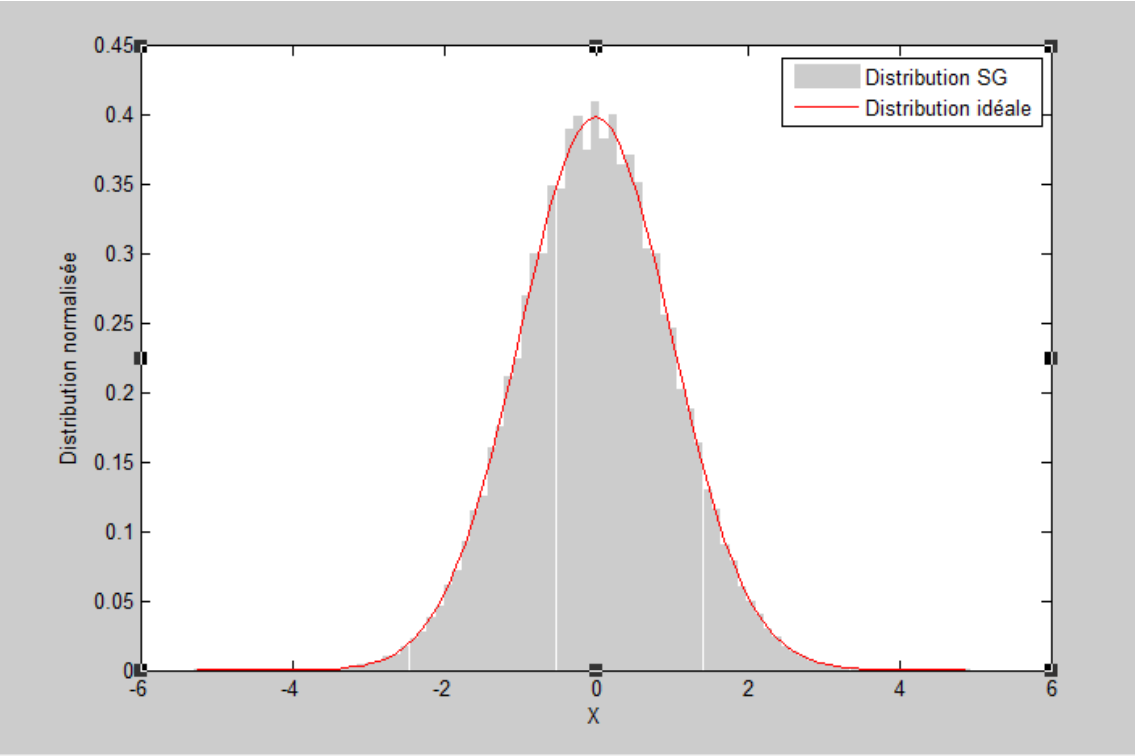


Figure4.18 : Distribution d'une v.a gaussienne de SG

Remarque : cette figure représente la Distribution d'une v.a gaussienne générée par SG de 1 million échantillon et distribution idéale normal N (0 1).

Comparaison :

La distribution gaussienne v.a normale de la méthode Box-Muller est proche de la distribution matlab par apport la distribution de SG.

La forme de Box-Muller est proche de la forme idéale N(0,1)

Donc :

Nous avons obtenu une bonne qualité de bruit blanc gaussien par la méthode de

Box-Muller, par rapport à la méthode du SG de Xilinx.

4.6 Implémentation sur la carte FPGA et test :

Nous avons implémenté le générateur du bruit blanc sur la carte FPGA de type

Spartan -3E en utilisant system generator et matlab.

D'après les résultats obtenus on remarque que le bruit blanc gaussien obtenue est de bonne qualité.

Preuve du succès de notre travail. Voir figure 4.19, 4.20, 4.21, 4.22.

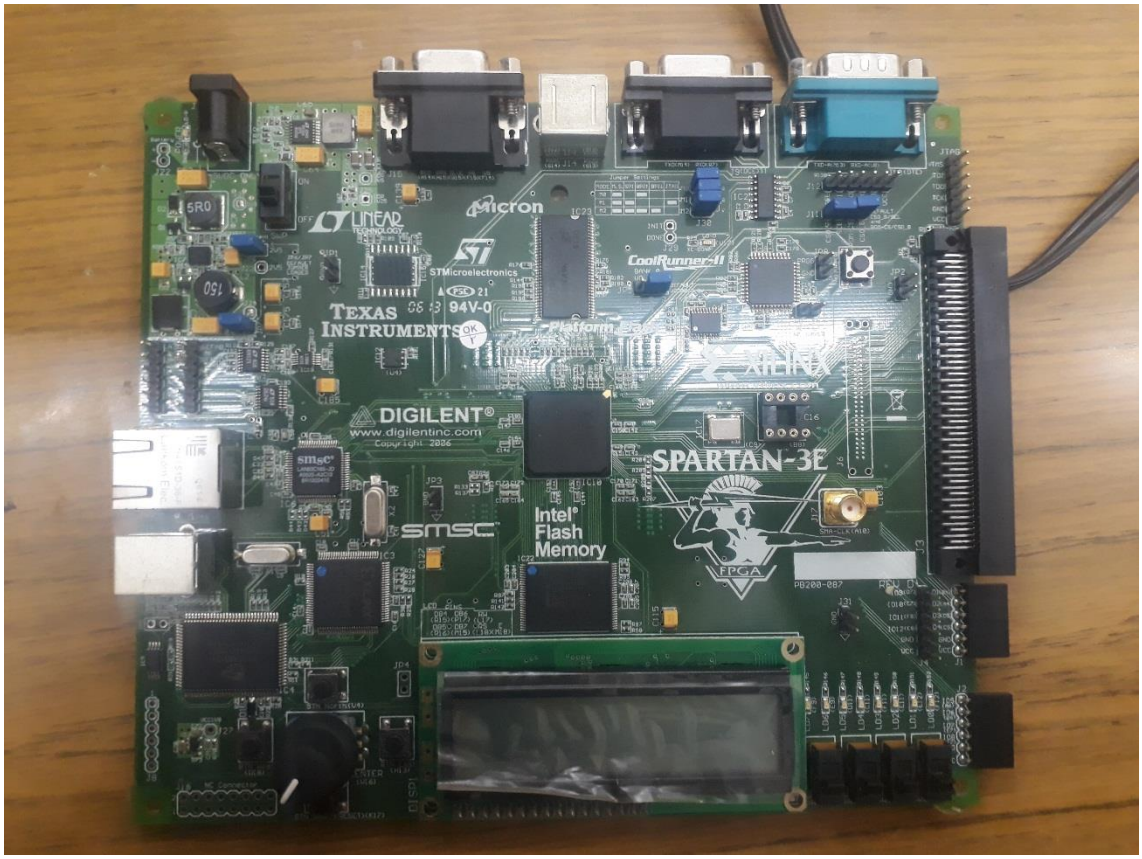


Figure4.19: la carte FPGA « spartan-3^{e} » utilisée dans l'implémentation

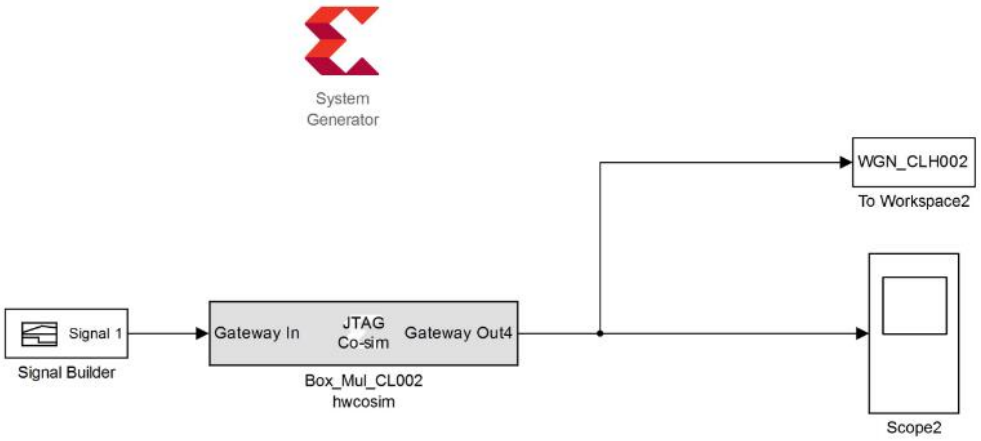


Figure4.20 : Co simulation Box-Muller « hardware »

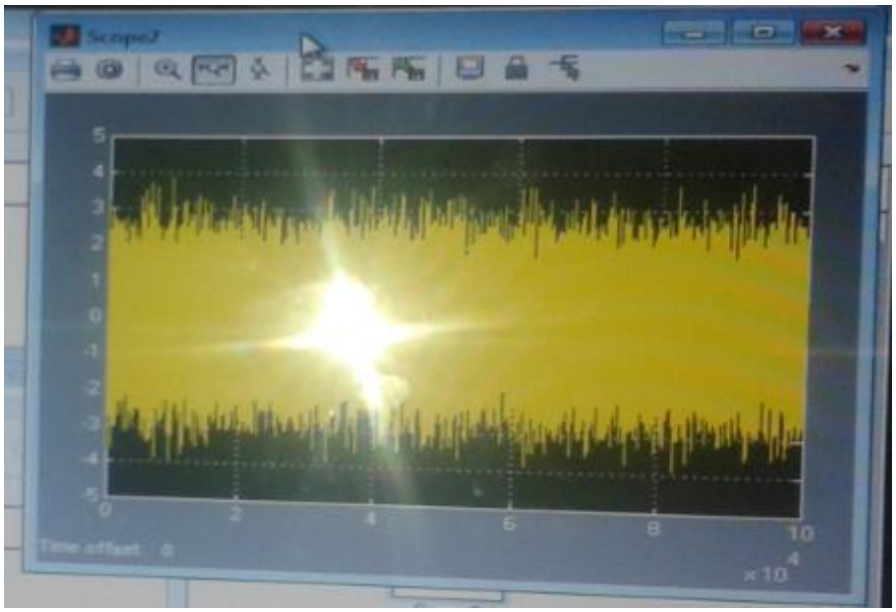


Figure4.21 : bruit blanc gaussien par FPGA

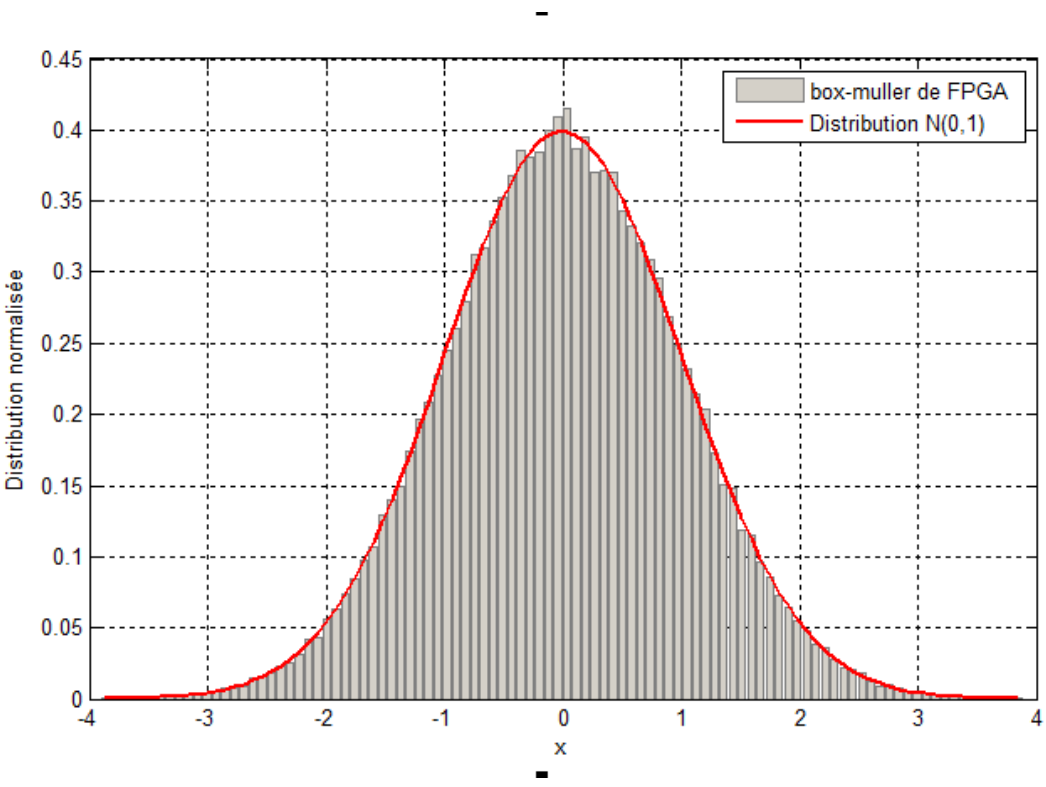


Figure4.22 : Distribution d'une v.a gaussienne réel par FPGA

Cette figure représente la Distribution d'une v.a gaussienne réel générée par la carte FPGA de 100K échantillon et distribution idéale normal N (0 1).

Remarque :

- Les résultats obtenus à partir des simulations dans Matlab et FPGA sont semblables.

4.7 Conclusion :

Dans ce chapitre, nous avons utilisé le Matlab, le ISE et le System Generator de Xilinx pour simuler, implémenter et tester notre générateur de bruit blanc.

Les résultats obtenus à partir des simulations du modèle à virgule flottante de Matlab, des simulations du modèle à virgule fixe du System Generator et des simulations de notre modèle à virgule fixe utilisant la méthode de Box-Muller, démontre la supériorité de notre approche par rapport à l'approche System Generator.

Les tests sur carte FPGA confirme parfaitement cette supériorité, car notre résultat se rapproche plus à la distribution idéale que les résultats du AWGNG de System Generator de Xilinx.

Conclusion générale

Dans ce travail nous avons étudié les principes des générateurs de la variable aléatoire gaussienne, appliqués dans les systèmes de communication numérique, existant. Nous avons exposé l'architecture, les calculs et la méthodologie de conception et de validation sur les circuits FPGA.

Dans la deuxième partie de notre travail nous avons synthétisé, développé et testé une nouvelle approche à virgule fixe moins complexe et implémentable sur FPGA.

Les résultats obtenus à partir des simulations du modèle à virgule flottante de Matlab, des simulations du modèle à virgule fixe du System Generator et des simulations de notre modèle à virgule fixe utilisant la méthode de Box-Muller, démontre la supériorité de notre approche par rapport à l'approche System Generator.

Les tests sur carte FPGA confirme parfaitement cette supériorité, car notre résultat se rapproche plus à la distribution idéale que les résultats du AWGNG de System Generator de Xilinx. .

Bibliographie

[1] D.Arzelier, cours signaux alétoires

[2] Hoel, P. G. (1984). *Mathematical Statistics* (Fifth ed.). New York.

[3]Andreas Klein, *Linear Feedback Shift Registers*, 20 avril 2013

[4] .Luc Devroye, *Non-Uniform Random Variate Generation*, New York, Springer-Verlag, 1986, 1re éd.

[5] J.L Danger, A. Ghazel, E. Boutillon H. Laamari,"Efficient
FPGA Implementation of Gaussian Noise Generator for
Communication Channel Emulation", The 7th IEEE Int.
Conf. on Electronicsm Circuits & Systemes (ICECS'2K),

Kaslik, Lebanon, Dec 2000.

[6]Philippe Tassi et Sylvia Legait, *Théorie des probabilités en vue des applications statistiques*, Technip, 1990.

[7] (Dr.Mamadou Lamine NDIAYE))fr.coursera.org/lecture/enseignes-et-afficheurs-led/5-5-circuits-logiques-programmables-fpga-i7RRe .

[8] (TAYARI LASSAAD) systemesembarques.e-monsite.com/medias/files/cours-fpga-tayari-vf-1.pdf

[9](MichaêlDardaillon)perso.citi.insa-lyon.fr/trisset/cours/rts12/slides/FPGA-handout.pdf.

[10] Debyo SAPTONO ,Thèse, Conception d'un outil de prototypage rapide sur le FPGA le 04 novembre 2011en.

[11] « adrien Blanchardon ».tel.archives-ouvertes.fr/tel-00935118/document

[12] perso.citi.insa-lyon.fr/trisset/cours/rts12/slides/FPGA-handout.pdf

[13] Hoong Le-Hug //w3.gel.ulaval.ca/~lehuy/intromatlab/intromat.pdf

[14] www.nandland.com/vhdl/modules/lfsr-linear-feedback-shift-register.html

[15] Jimmy, thomas Djiako, Eric Schiller, Bocar Sada sy. « le pompage photovoltaique, manuel de cours a l'intention des ingénieurs et des techniciens ».Universitéd'ottawa,eric schiller,1998..

[16] PHILIPPE LE-HUY :'SIMULATION TEMPS REEL DE CONVERTISSEURS DE PUISSANCE A L'AIDE DE FPGA ',mémoire dans le cadre du programme de maitrise en génie électronique pour l'obtention du grade de maitrises sciences (M,SE) ,la faculté des études supérieures lava, 2006.

[17]Jean –Gabriel Mailloux :'PROTOTYPAGE RAPIDE DE LA COMMANDE VECTORIELLE SUR FPGA A L'AIDE DES OUTILS SIMULINK-SYSTEM GENERATOR ,mémoire dans le cadre du programme de maitrise d'ingénierie,l'université du QUEBECA CHICOUTIMI ,MARS 2008.

Andreas Klein, *Linear Feedback Shift Registers*, 20 avril 2013