# Un générateur de bruit blanc gaussien sur un FPGA pour la simulation rapide de systèmes de transmissions

Emmanuel Boutillon[(1)], Adel Ghazel[(2)], Jean-Luc Danger[(3)], Glenn Gulak[(4)]

[(1)]LESTER. University of Bretagne Sud, - BP 92116 - 56321 LORIENT Cedex, France

[(2)]UTIC - Ecole Supérieure des Communications, Rte de Raoued km 3.5 – 2083 El Ghazala –Tunisia

[(3)]ENST PARIS, 46 rue Barrault, 75634 PARIS CEDEX 13, France

[(4)] E . S. Rogers Sr. Dpt of ECE, University of Toronto, 10 King's College Street Toronto, M5S 3G Ontario

adel.ghazel@supcom.rnu.tn, emmanuel.boutillon@univ-ubs.fr, danger@enst.fr, gulak@eecg.toronto.edu

**Résumé :** Un générateur matériel de bruit blanc gaussien de "bonne qualité" est développé sur une cible FPGA, dans le cadre d'une émulation de canal de communications mobiles. Une grande précision et un faible coût matériel du générateur sont obtenus par l'utilisation conjointe de la méthode de Box-Muller et du théorème de la limite centrale. 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.

**Abstract :** A hardware White Gaussian Noise Generator (WGNG) is developed for mobile communication channel emulation in FPGA circuit. High accuracy, fast and low-cost hardware are reached by combining the Box-Muller and Central limit methods. 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.

## 1. Introduction

La conception d'un circuit intégré pour une application de communication numérique (code correcteur d'erreur, démodulateur) requiert souvent des compromis entre complexité matérielle et performances. Les méthodes formelles de calcul de performances [1] 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. La solution généralement utilisée consiste à effectuer des simulations logicielles par la méthode de Monte-Carlo 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. En effet, l'estimation d'un Taux d'Erreur Binaire de $10^{-6}$, avec plus ou moins 3% d'erreur, requiert environ $10^9$ tirages. Bien sûr, les mesures sur le terrain permettent de valider le choix des différents paramètres mais ces mesures ont lieu très tard, après la conception du circuit.

La solution que nous proposons est de réaliser l'optimisation à partir d'un prototype "reconfigurable" traitant les données en temps réel. Ce prototype doit être inscrit dans un environnement émulant le canal réel de façon réaliste. Le but de cet article est de présenter l'architecture d'un Générateur de Bruit Blanc Gaussien (GBBG) de très bonne qualité permettant la modélisation précise du canal de transmission. La bonne qualité du GBBG se caractérise par la génération d'une variable aléatoire (v.a.) $X$ ayant une densité de probabilité (d.d.p.) dite conforme à ($4\sigma$, 1%), c'est-à-dire que l'erreur relative $\xi_X(x)$ entre la d.d.p de $X$ et la d.d.p de la distribution normale $N(0,1)$ (moyenne nulle, variance $\sigma^2$ unitaire)

$$\xi_X(x) = \frac{X(x) - N(0,1)(x)}{N(0,1)(x)} \qquad (1)$$

est inférieure à 1% pour tout $|x| < 4\sigma$ . D'autre part, le GBBG doit aussi avoir une résolution d'au moins $b$=6 bits après la virgule de résolution ($\sigma$/64 de précision) et une périodicité très grande (supérieure à $2^{48}$ échantillons), pour permettre de longues simulations.

L'article est structuré en trois parties. Dans la première partie, nous abordons la génération efficace, sur des FGPAs, de v.a. binaires pseudo-aléatoires en utilisant des LFSRs (Linear Feed-Back Shift Register). Ensuite, nous proposons la combinaison de la méthode de Box-Muller et du théorème de la limite centrale pour obtenir un GBBG de "bonne qualité". Enfin, nous présentons les outils permettant de quantifier la qualité et la complexité du GBBG. Des résultats de réalisation et de simulations sont aussi donnés.

## 2. Génération des v.a. binaires

Après avoir rappelé le principe de la génération d'un v.a. binaire par un LFSR, nous proposons une optimisation pour une réalisation sur FPGA.

### *2.1 Principe du LFSR*

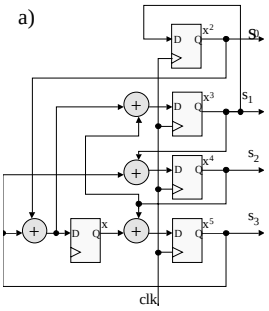
Un LFSR de polynôme caractéristique G[x] d'ordre n permet, de générer, si l'état initial est non nul, une séquence binaire pseudo-aléatoire de périodicité $p=2^n-1$. La réalisation d'un

LFSR sur un circuit logique programmable de type FPGA requière au minimum *n* cellules logiques pour les *n* registres du LFSR.

### *2.2 Optimisation des LFSRs*

Dans le cas où plusieurs v.a. binaires sont à générer, nous proposons d'utiliser des LFSR calculant directement les puissances successives de $x^4$ afin de générer, pour chaque LFSR, 4 v.a. binaires indépendantes par cycle d'horloge. Le schéma logique du LFSR devient plus complexe mais en pratique, le nombre de cellules logiques utilisées reste pratiquement constant. Ainsi, plutôt que d'utiliser 4 LFSRs de longueur *n* pour générer 4 v.a. binaires (et donc 4×*n* cellules logiques), un seul LFSR de longueur n+2 (la périodicité est divisée par 4) est suffisant.

Les figures 1.a et 1.b illustre ce principe pour un polynôme générateur $G[x]=x^5+x^2+1$ d'ordre 5.




| | LFSR 1 output | | | | | LFSR 4 outputs | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| t | x | $x^2$ | $x^3$ | $x^4$ | $x^5$ | x | $x^2$ | $x^3$ | $x^4$ | $x^5$ |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 3 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 4 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 5 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 6 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 7 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 8 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 9 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 10 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 11 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 12 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |

Figure 1 : Exemple de LFSR à quatre sorties
a) schéma logique b) séquences obtenues

Notons que si plusieurs LFSRs sont utilisés simultanément, la périodicité maximale de l'ensemble est obtenue si toutes les périodes des LFSR sont premières entre elles. Il suffit, pour cela, de choisir les longueurs des LFSRs parmi les nombres de Mersenne[1].

## 3. Principe du GBBG

Une version quantifiée de la méthode de Box-Muller est tout d'abord présentée. Cette méthode est ensuite associée avec la méthode du théorème de la limite centrale de façon à obtenir un GBBG de "bonne qualité" et peu complexe.

### *3.1 Méthode de Box-Muller quantifiée.*

La méthode de Box-Muller [2] est largement utilisée pour les simulations logicielles. Elle consiste à générer la v.a. $N(0,1)$ à partir de deux v.a. réelles indépendantes $x_1$ et $x_2$ uniformément distribuées sur [0, 1] en utilisant les trois étapes suivantes :

$$f(x_1) = \sqrt{-\ln(x_1)} \quad (3)$$

$$g(x_2) = \sqrt{2}\cos(2\pi x_2) \quad (4)$$

$$n = f(x_1)g(x_2) \quad (5)$$

où $n$ est un tirage de la v.a. $N(0,1)$.

Pour éviter le calcul de la racine carrée d'un logarithme (fonction $f(x_1)$, équation (3)) et d'un cosinus (fonction $g(x)$, équation (4)), nous proposons de quantifier ces opérations et d'utiliser des tables pré-calculées stockées en mémoire.

### *3.2 Quantification non uniforme du segment* [0, 1]

Etudions tout d'abord l'équation (3). La fonction $f(x_1)$ est tracée figure 2. Comme la moyenne de la fonction $g(x)$ est proche de 1 ($2\sqrt{2}/\pi$ exactement), des valeurs de $f(x)$ supérieures à 4 doivent être générées pour obtenir des tirages de $n$ supérieurs à 4.

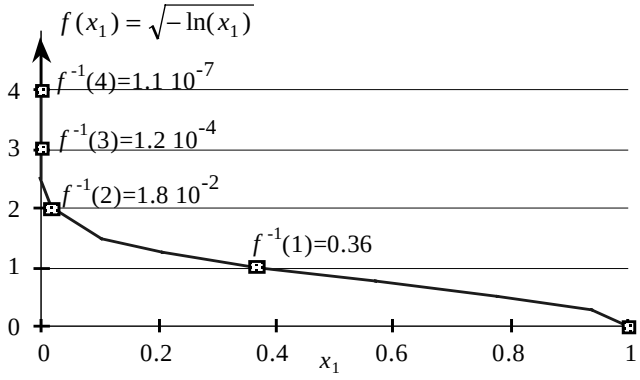



Figure 2: Fonction $f(x_1)$

Pour cela , il faut générer des valeurs de $x_1$ très faibles car $f^{-1}(4)$ est inférieur à $10^{-7}$. Un pas de quantification du segment [0,1] de l'ordre de $10^{-7}$ conduit à une solution coûteuse en mémoire. Pour diminuer la taille de la mémoire, nous proposons la quantification récursive non uniforme du segment [0,1] présentée dans la figure 3.

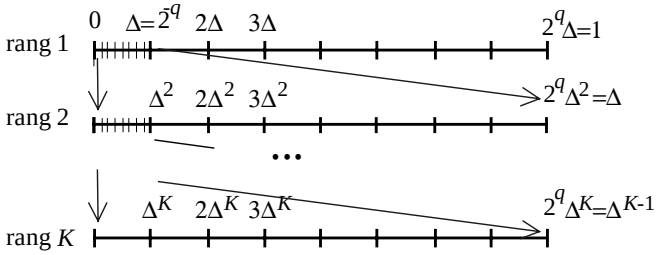



Figure 3: Quantification non-uniforme de [0,1]

Dans cette figure, $K$ représente le nombre de récurrences et $q$ représente le nombre de bits nécessaires pour sélectionner l'un des $2^q$ sous-segments de longueur $\Delta^r = 2^{-rq}$ du niveau de

[1] $w$ est un nombre de Mersenne si $2^w - 1$ est premier

récurrence $r$. Ainsi, $K$ générateurs pseudo-aléatoires $s_1$, $s_2$... $s_K$, chacun de $q$-bits, sont utilisés pour définir la position d'un "tirage" de la variable $x_1$ (c'est-à-dire, la position d'un sous-segment de [0,1]. Au premier niveau, ($r$=1), le segment [0,1] est uniformément quantifié en $2^q$ intervalles de longueurs $\Delta=2^{-q}$. La valeur de $s_1$ indique le numéro du segment [$\Delta s_1$, $\Delta(s_1+1)$[. Si $s_1$ est nul, le sous-segment [0, $\Delta$[ est alors lui même divisé en $2^q$ sous-segments de taille $\Delta^2$ indicés par $s_2$. Le processus est réitéré récursivement $K$ fois. Notons que la probabilité de sélectionner un segment $s$ de rang $r$ est égale à $\Delta^r$, c'est-à-dire, la taille du segment. Le processus est donc bien uniformément distribué sur [0,1]. A chaque segment est associé la valeur quantifiée $f_r(s)$ défini par :

$$f_r(s) = \left\lfloor 2^m \, f((s+\delta)\Delta^r) \right\rfloor \quad (\times 2^{-m}) \qquad (6)$$

avec $m$ le nombre de bits après la virgule (en binaire) de $f_r(s)$ ($f_r(s)$ est donc codé sur 3+$m$ bits, 3 pour la partie entière, $m$ pour la partie fractionnaire) et $\lfloor x \rfloor$ indique l'entier le plus grand inférieur ou égale à $x$. Le nombre $\delta$ est compris entre 0 et 1. Il indique la position de $x_1$ dans le segment [$\Delta^r s$, $\Delta^r(s+1)$[ à partir duquel est déterminé $f(x_1)$.

Pour la fonction $g(x)$, on utilise les propriétés de symétrie de la fonction cosinus. L'intervalle [0,1/4] est divisé en $2^{q'}$ intervalles de longueur $\Delta' = 2^{-q'}$. Une v.a. uniforme $s'$ de $q'$ bits permet de sélectionner l'un des sous-segments et la valeur $g(x)$ associée en utilisant :

$$g(s') = \left\lfloor 2^{m'} \sqrt{2} \cos\left( \frac{\pi\Delta'(s'+\delta')}{2} \right) \right\rfloor \quad (\times 2^{-m'}) \qquad (7)$$

ou $\delta'$ et $m'$ ont la même signification que $\delta$ et $m$ dans l'équation (6). $g(s')$ est codé sur (1+ $m'$) bits, 1 bit pour la partie entière et $m'$ bit pour la partie fractionnaire.

A partir de $f_r(s)$ et de $g(s')$, la réalisation $n^+$ de la v.a. *DBM* (Demi-Box Muller) ayant $b$ bits après la virgule est donnée par :

$$n^+ = \left\lfloor \frac{f_r(s) \times g(s')}{2^{m+m'-b}} \right\rfloor \quad (\times 2^{-b}) \qquad (8)$$

La probabilité $P$ d'obtenir un couple $(f_r(s), g(s'))$ est :

$$P(f_r(s), g(s')) = 2^{-(rq+q')} \qquad (9)$$

Soit $S_x$ le sous-ensemble des triplets ($s$,$r$,$s'$) de {0, ..., $2^q$-1}x {1, ..., K}x{0,..., $2^{q'}$-1} permettant le calcul de $n^+$ en utilisant (8). La probabilité $P(DBM=n^+)$ est donnée par :

$$P(DBM=n^+) = \sum_{(s,r,s') \in S_n} P((f_r(s), g(s'))) \qquad (10)$$

En utilisant (8), (9) et (10), la d.d.p. de *DBM* peut-être calculée. A partir de la v.a. *DBM*, la v.a. *BM* est obtenue en lui ajoutant un signe désigné par la v.a. binaire *sign* :

$$n = (1-2sign)n^+ \qquad (11)$$

A partir de (11), on peut calculer la d.d.p de *BM* à partir de celle de *DBM*.

### *3.3 Association avec le théorème de la limite centrale*

La quantification non uniforme utilisée pour la fonction $f$ ainsi que la quantification de la fonction $g$ entraînent une d.d.p. de la v.a. *BM* qui n'est qu'approximativement gaussienne (voir section suivante). Pour essayer de lisser les oscillations autour de la courbe idéale, nous proposons d'utiliser le théorème de la limite centrale. Celui-ci nous dit que si $X$ est une variable aléatoire (v.a) de moyenne $m_X$ et de variance $\sigma_X$, alors, la v.a. $X_A$ défini par:

$$X_A = \frac{1}{\sigma_X \sqrt{A}} \sum_{i=0}^{A-1} (x_i - m_x) \qquad (12)$$

avec $x_i$ , $i$=0..$A$-1, $A$ tirages indépendants de $X$, tend vers la distribution normale $N(0,1)$, quand $A$ tend vers l'infini.

Notons que de nouveau, la d.d.p de $BM_N$ (accumulation de $N$ v.a. indépendantes *BM*) peut aisément être calculée comme la N$^{\text{ième}}$ convolution de la d.d.p. de *BM* avec elle-même. L'écart type de $BM_N$ est égal à l'écart type de *BM* multiplié par un facteur $\sqrt{N}$ . En particulier, pour, $N$=4, la division par 2 de $BM_4$ donne une v.a. ayant un écart type de 1.

## 4. Qualité et complexité

Le modèle du GBBG peut être paramétré pour tenir compte des spécificités de la cible technologique FPGA. Les résultats obtenus en terme de précision et complexité matérielle sont présentés.

### *4.1 Qualité du GBBG*

Un programme MATLAB (voir annexe), a été écrit afin de calculer la distribution exacte de $BM_N$ en fonction des différents paramètres. Nous avons utiliser ce programme pour optimiser la conception d'un GBBG répondant au cahier des charges sur une cible FPGA (FLEX10K100EQC240-1, Altera [3]). Nous avons choisi a priori $q$ égal à 4 afin de créer des mémoires de la même taille que celle des cellules logiques du FPGA et $q'$=8 afin d'utiliser une mémoire embarquée pour la table des cosinus [4]. Les autres paramètres ont été choisi de façon à vérifier le cahier des charges (voir table 1).

Table 1 : Caractéristique du GBBG Box-Muller

| *Param.* | *b* | *q* | *K* | *m* | δ | *q'* | *m'* | δ' |
|---|---|---|---|---|---|---|---|---|
| $BM_1$ | 6 | 4 | 5 | 7 | 0.36 | 8 | 6 | 0.5 |

Les fonctions d'erreur relatives pour $BM_1$, $BM_2$ (A = 2 accumulations) et $BM_4$ (A = 4 accumulations) sont données figure 5.

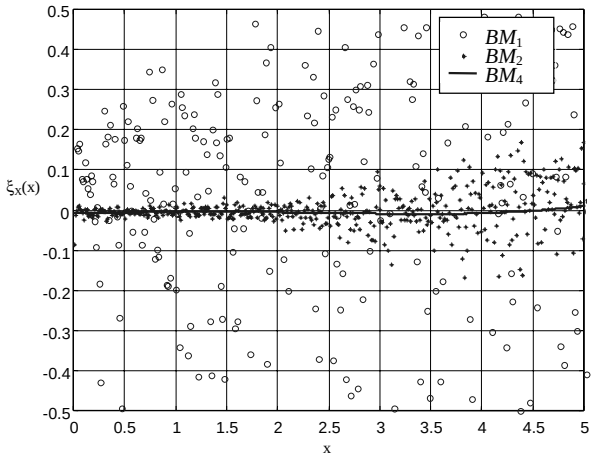



Figure 5: fonction $\xi_X(x)$ pour $BM_1$ $BM_2$ *et* $BM_4$

Cette figure montre que la fonction $BM_4$ remplit largement le cahier des charges car l'erreur est de l'ordre de 1% à 4σ.

### *4.2 Complexité de GBBG*

La Figure 4 montre l'architecture générale du GBBG :

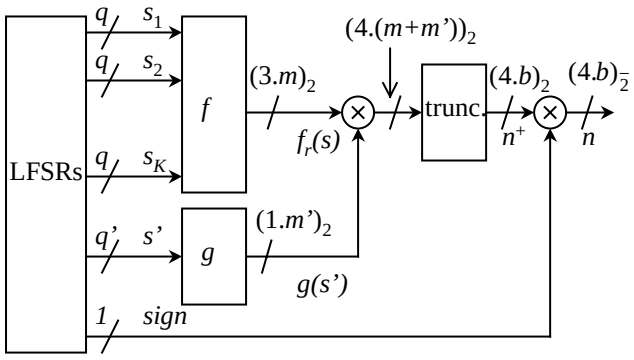



Figure 4: Architecture générale du GBBG

Dans cette figure $(a.b)_2$, respectivement $(a.b)_{\bar{2}}$, représente un nombre non signé (respectivement signé) avec *a* et *b* bits avant et après la virgule..

Il faut rajouter à ce schéma un accumulateur de *A* valeurs consécutives sur la sortie, permettant d'appliquer le théorème de la limite centrale.

Notons que le GBBG requiert la génération de $(K*q + q' + 1)$ variables aléatoires binaires, *K* ROM de taille $2^q \times (3+m)$, une ROM de taille $2^{q'} \times (1+m')$, un multiplieur $(3+m) \times (1+m')$ et enfin, un additionneur pour l'ajout du signe. La complexité matérielle peut donc être facilement calculée.

La table 2 donne les caractéristiques matérielles du GBBG obtenu. Quatre cycles d'horloge sont nécessaires pour effectuer la génération et la somme de 4 échantillons $BM_4$.

Table 2 : Résultats de synthèse

| Nombre de cellule | Numbre de LAB | Fréquence horloge | Débit sortie |
|---|---|---|---|
| 434 | 1 | 98 MHz | 24.5 MHz |

La complexité totale est faible (10% du total des cellules FPGA) et, comme indiqué figure 5, les valeurs des d.d.p théoriques et mesurées sont cohérentes.

## 5. Conclusion

Dans cette étude, il est montré que l'utilisation conjointe de la méthode de Box-Muller et du théorème de la limite centrale pour générer une variable aléatoire gaussienne donne d'excellents résultats pour obtenir une d.d.p. conforme à une précision de (4σ, 1%). Un programme MATLAB générique a été écrit. Il permet de prendre en compte les spécificités de la cible technologique et de générer une erreur relative de façon à vérifier la conformité de la d.d.p. obtenue. L'utilisation de paramètres adaptés à une architecture FPGA génère une faible complexité et une précision de 1 % à 4σ.

Ce générateur matériel de variable gaussienne est la base d'autres travaux en cours visant à émuler des canaux radio-mobiles de type Rayleigh ou Rice.

## 6. Références


[1] J.G. Proakis, "*Digital communications"*, Mc GRAW-HILL International Editions, Electrical Engineering Series, 1998.

[2] Donald E. Knuth, "The Art of computer programming", ADDISON-WESLEY, 1998.

[3] ALTERA Data Book 1998.

[4] 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.


Les fichiers MATLAB et VHDL de l'émulateur se trouve à la rubrique émulateur canal du site WEB :

http://lester.univ-ubs.fr:8080/%7Eboutillon/anglais.html