

# DESIGN AND PERFORMANCE ANALYSIS OF A HIGH SPEED AWGN COMMUNICATION CHANNEL EMULATOR

**Adel Ghazel<sup>(1)</sup>, Emmanuel Boutillon<sup>(2)</sup>, Jean-Luc Danger<sup>(3)</sup>, Glenn Gulak<sup>(4)</sup>, Hédi Laamari<sup>(1)</sup>**

<sup>(1)</sup>UTIC - Ecole Supérieure des Communications, Rte de Raoued km 3.5 – 2083 El Ghazala –Tunisia

<sup>(2)</sup>LESTER. University of Bretagne Sud, - BP 92116 - 56321 LORIENT Cedex, France

<sup>(3)</sup>ENST PARIS, 46 rue Barrault, 75634 PARIS CEDEX 13, France

<sup>(4)</sup>ECCG, 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

**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. The complexity and the performance level are given for some configurations and show the interest of the proposed model.

## 1 Introduction

The software estimation of the performances of a communication system is very time consuming. Indeed, with a Monte-Carlo simulation, an accurate (+3.3%) estimation of a Binary Error Rate around  $10^{-6}$  needs  $10^9$  iterations. Moreover, many variables (sampling frequency, digital format, carrier resolution, rounding and quantization etc.) have to be optimized for satisfying the best trade-off between performances and complexity. In order to speed up the final parameter optimization of a design, we proposed to perform direct hardware simulation (emulation) on an FPGA. Such a simulation needs a hardware emulation of the communication channel.

In this paper, the authors (a joined research group composed of ENST-Paris (France), SUP'COM (Tunisia), LESTER (France) and the University of Toronto (Canada)) focus their attention on the design of a "high quality" White Gaussian Noise Generator (WGNG). The "high quality" WGNG considered in this paper performs the generation of a random variable  $X$  with the following characteristics:

- at least  $b=6$  bits of resolution after the dot,
- a periodicity of the WGNG greater than  $2^{48}$  samples (function rand48 of C ANSI),
- a flat spectrum.
- a " $(4\sigma, 1\%)$  normal-like probability density function (p.d.f.)" i.e. the absolute value of the relative error  $\xi_X(x)$  defined as:

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

between the p.d.f. of  $X$  and the normal distribution  $N(0,1)$  - mean 0 and standard deviation  $\sigma = 1$  - is less than 1% for all  $|x| < 4\sigma$ .

0-7803-7080-5/01/\$10.00 ©2001 IEEE

With the use of this WGNG, the Additive White Gaussian Noise (AWGN) channel can be emulated as well as more complex channels like Rayleigh channel and Ricean channel (using filtering and appropriate mathematical functions [1]).

Sections 2 and 3 briefly present the two well-known methods for generating a Gaussian noise, namely, the central limit and the Box-Muller methods, and give some developments for achieving better accuracy. Section 4 presents an efficient combination of the two methods. Performances and hardware complexity are evaluated.

## 2 Central limit based method

The central limit theorem tells us that if  $X$  is a real random variable (r.v.) of mean  $m_x$  and standard deviation  $\sigma_x$ , the random variable (r.v.)  $X_N$  defined as:

$$X_N = \frac{1}{\sigma_x \sqrt{N}} \sum_{i=0}^{N-1} (x_i - m_x) \quad (2)$$

where  $x_i$ ,  $i=0..N-1$  are  $N$  independent determinations of the variable  $X$ , tends towards the normal distribution  $N(0,1)$ , when  $N$  tends towards infinity.

The central limit theorem gives a very simple method to generate a white gaussian noise. Indeed, it is well known that a Linear Feedback Shift Register (LFSR) of length  $l$  can generate a very good random-like binary variable of periodicity  $2^l-1$ . The concatenation of  $q$  different LFSRs, gives a  $q$ -bit vector  $U^q$ . This vector can be seen as a random variable uniformly distributed over  $\{0, 1, 2, \dots, 2^q-1\}$ . Thus, the random variable  $U^q_N$  obtained using (2) with  $N$  independent determinations of  $U^q$  leads, after appropriate scaling, to a good approximation of  $N(0,1)$  if  $N$  is large enough. Figure 1 shows the  $\xi_X(x)$  function obtained for  $X=U^q_N$ , with  $q=8$  and  $N=2, 4, 8, 16, 32$ . One can note, that even with  $N$  large, the approximation of the distribution  $N(0,1)$  between 0 and  $4\sigma$  is not adequate ( $|\xi_X(x)| > 5\%$  for  $|x| > 3\sigma$ ) considering the present requirements. Moreover, in terms of hardware implementation, this solution is costly since  $qN = 256$  binary variables (and thus 256 LFSRs) are needed to obtain a single sample. Thus, this method used alone is inadequate for the present purpose.

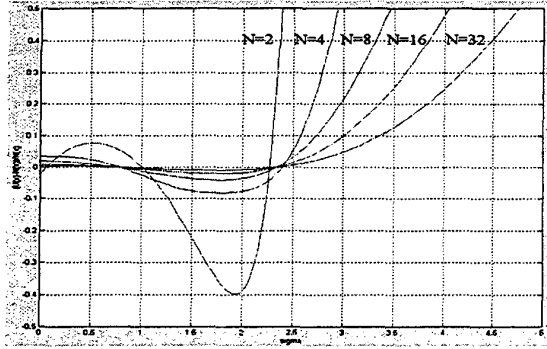


Figure 1:  $\xi_X(x)$  function for  $X=U^8_N$ ,  $N=2, 4, 8, 16, 32$ .

The box-Muller method is now investigated and the WGNG quality is analyzed at low hardware cost.

### 3 Box-Muller based reference model

The Box-Muller method is first presented. Then, a new quantized version of the Box-Muller method matched for hardware implementation is proposed. Hardware complexity of the WGNG is described as well as an example of realization.

#### 3.1. Description of the method

The Box-Muller method is widely used in simulation software to generate a random variable (see [3] for a C program example). This method includes 3 steps: the first two ones generate independent values  $x_1$  and  $x_2$  of a random variable uniformly distributed over  $[0, 1]$ . In the third step, the functions,  $f(x_1)$  and  $g(x_2)$  are derived from  $x_1$  and  $x_2$  by :

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

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

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

gives a sample of the  $N(0,1)$  distribution (see [3] for a proof).

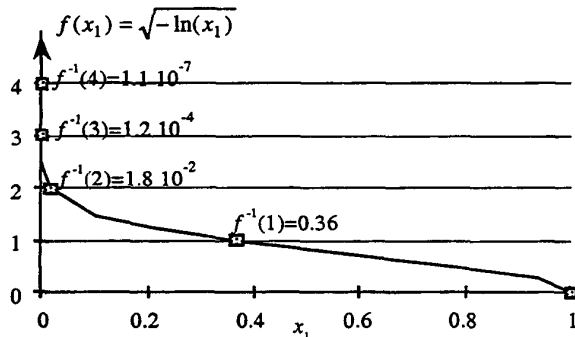


Figure 2: Plot of function  $f(x_1)$

Using a 32-bit floating point CPU, equations (3), (4) and (5) are efficiently computed in a small number of clock cycles. Unfortunately, these operations (square root of a logarithm, cosine function, multiplication) require a lot of hardware. To reduce the complexity, a quantized version of (3) and (4) using pre-computed values is proposed by the authors. Let us first focus on equation (3). The plot of the function  $f(x)$  is shown in figure 2. Since the average of  $g(x)$  is close to 1 ( $2\sqrt{2}/\pi$  exactly), a value of  $f(x)$  greater than 4 should be generated in order to generate sample  $n$  greater than 4.

#### 3.2. non-uniform quantization of segment $[0, 1]$

To do so, the quantization step of segment  $[0,1]$  should be very small, i.e. of the order of  $f^{-1}(4) < 10^{-7}$ . Since only the vicinity of 0 should be accurately quantized, a recursive non-uniform quantization of segment  $[0,1]$  is proposed (see figure 3):

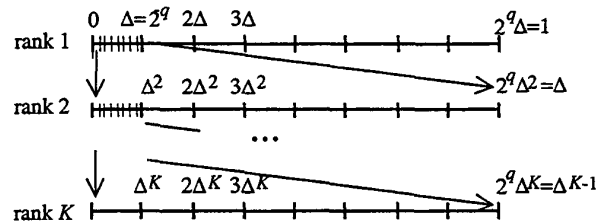


Figure 3: Non uniform quantization of segment  $[0,1]$

where  $K$  is the number of recursion and  $q$  is the number of bits required to select one of the  $2^q$  segments of length  $\Delta^r = 2^{-rq}$  at the level  $r$  of the recursion. Thus,  $K$   $q$ -bit random generators  $s_1, s_2, \dots, s_K$  are used to define the position of a sub-segment of  $[0,1]$ . At the first level ( $r=1$ ), the quantization step of  $[0,1]$  is  $\Delta=2^{-q}$ . The value  $s_1$  defines the segment  $[\Delta s_1, \Delta(s_1+1)[$ , if  $s_1$  is equal to 0, the sub-segment  $[0, \Delta[$  is sub-divided in  $2^q$  sub-segments of size  $\Delta^2$  indexed with  $s_2$ . Once again, if  $s_2$  is equal to 0, the sub-segment  $[0, \Delta^2[$  is sub-divided in  $2^q$  sub-segments indexed by  $s_3$ . The process is repeated recursively  $K$  times. The probability to select a segment  $s$  of rank  $r$  is equal to  $\Delta^r$ , i.e., the size of the segment. The process is thus uniformly distributed over  $[0,1]$ . The quantized value  $f_r(s)$  associated to this segment is given by:

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

where  $m$  is the number of fractional bits used to represent  $f_r(s)$  (i.e.  $f_r(s)$  is coded on  $3+m$  bits, 3 for the integer part,  $m$  for the fractional part) and  $\lfloor x \rfloor$  denotes the largest integer lower than  $x$  and  $\delta$ , a real number between 0 and 1, gives the relative position of the sample in the segment  $[\Delta^r s, \Delta^r(s+1)[$ .

The problem is easier for the  $g(x)$  function. Let us define  $s'$ , a  $q'$  bit random variable.  $\Delta' = 2^{-q'}$  is the quantization step of

segment  $[0, 1/4]$  (the problem of sign is analyzed later) so  $g(x)$  is quantized as:

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

where  $\delta'$  and  $m'$  have the same meanings as those of  $\delta$  and  $m$  of equation (6).  $g(s')$  is coded on  $1+m'$  bits, 1 for the integer part,  $m'$  for the fractional part.

From  $f_r(s)$  and  $g(s')$ , the quantized Half Box-Muller random variable  $HBM$  with  $b$  bits after the dot is obtained using:

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

The probability  $P$  of obtaining a given couple  $(f_r(s), g(s'))$  is:

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

Let  $S_x$  be the subset of  $\{0, \dots, 2^q-1\} \times \{1, \dots, K\} \times \{0, \dots, 2^{q'}-1\}$  of all triples  $(s, r, s')$  that give  $n^+$  using (8). The probability  $P(HBM=n^+)$  is then given by:

$$P(HBM=n^+) = \sum_{(s,r,s') \in S_x} P(f_r(s), g(s')) \quad (10)$$

Using (8), (9) and (10), the p.d.f. of  $HBM$  can easily be computed. Finally, the Box-Muller r.v.  $BM$  is obtained from the Half Box-Muller r.v.  $HBM$  using a binary r.v.  $sign$ :

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

The schematic representation of the algorithm is shown in figure 4.

### 3.3. Hardware complexity

Figure 4 show the general architecture of the WGNG:

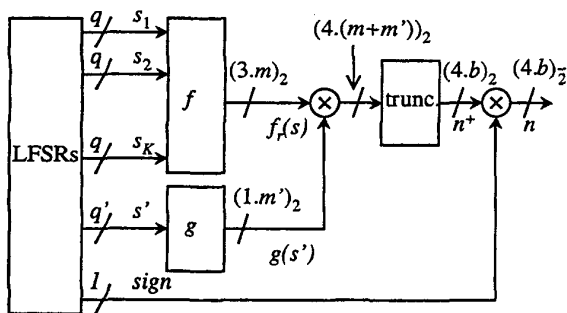


Figure 4: Overall architecture where  $(a.b)_2$ , respectively  $(a.b)_{-2}$ , indicates an unsigned (a two-complement) number with  $a$  bits before the dot and  $b$  bits after the dot.

The WGNG requires  $(K \cdot q + q' + 1)$  LFSRs for generating the binary variables,  $K$  ROM of size  $2^q \times (3+m)$ , one ROM of size

$2^{q'} \times (1+m')$ , one  $(3+m) \times (1+m')$  multiplier and one adder for the sign multiplication. Hardware complexity of the WGNW is then easy to evaluate.

In 1995, a first Box-Muller WGNG has been investigated by a training student, supervised by E. Boutillon and G. Gulak, at ENST. The characteristics of this WGNG are listed table 1 (line  $BM$ ). The quality of the result is given in figure 5.

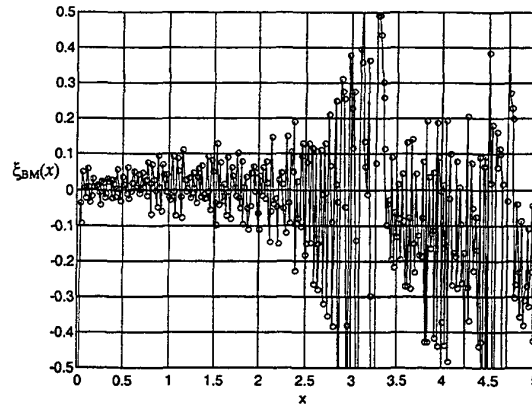


Figure 5:  $\xi_{BM}(x)$  function

As shown in figure 5, the function  $\xi_{BM}(x)$  lies around 0 on the average. However, the variation can be large and a substantial increase of the cost should be used for a substantial decrease of these variations. Thus, the Box-Muller method used alone is not appropriate for the present purpose.

## 4. Proposed new WGNG model

For improving the method we propose to use the central limit theorem to generate  $BM_N$  as the sum of  $N$  independent determinations of  $BM$  (see eq. (2)). Thus, the p.d.f. of the r.v.  $BM_N$  will be flatter than that of  $BM$ . Since the p.d.f. of  $BM$  is, on the average, "gaussian-like", the central limit theorem converges very quickly. Note that, once again, the p.d.f. of  $BM_N$  can be computed as the  $N^{\text{th}}$  convolution of the p.d.f. of  $BM$  with itself. The standard deviation of  $BM_N$  is equal to the standard deviation of  $BM$  multiplied by a factor  $\sqrt{N}$ . Thus, for  $N=4$ , the division by 2 of  $BM_4$  gives r.v. with a standard deviation of 1.

A program (MATLAB) has been elaborated which performs the exact computation of  $BM_N$  given any set of parameters. It is given in the annex. With this program, the optimization of a WGNG for a given set of parameters is very fast. As an illustration, Table 1 gives the results of a previous paper [3]. Line  $BM_1$  shows the parameter of a WGNG optimized for an FPGA circuit (namely, the FLEX10K100EQC240-1 of Altera [4]).

| Param.          | b | q   | K | m   | $\delta$ | q'  | M'  | $\delta'$ |
|-----------------|---|-----|---|-----|----------|-----|-----|-----------|
| Matlab          | b | q_f | K | m_f | d_f      | q_g | m_g | d_g       |
| BM              | 6 | 8   | 3 | 6   | 0.5      | 8   | 8   | 0.5       |
| BM <sub>1</sub> | 6 | 4   | 5 | 7   | 0.36     | 8   | 6   | 0.5       |

Table 1: Characteristics of the Box-Muller WGNG. The second line of the array gives the names of the variables used in the MATLAB program given in this annex.

In this design,  $q=4$  in order to match the size of the logical cell of the circuit. Figure 6 shows the resulting p.d.f. for  $BM_1$ ,  $BM_2$  and  $BM_4$ . One can note that  $BM_4$  resulting from the summation of 4 independent variables  $BM_1$  is "(4 $\sigma$ , 1%) gaussian like": the required specifications are fulfilled<sup>1</sup>.

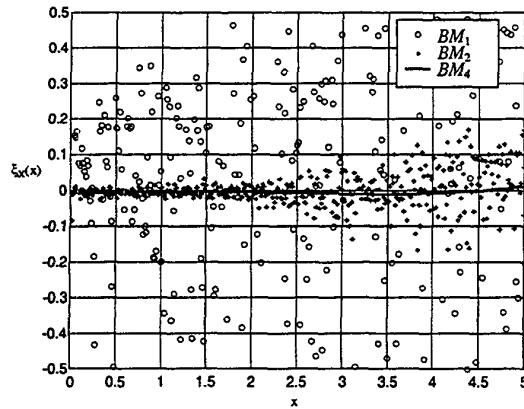


Figure 6:  $\xi_X(x)$  function for  $BM_1$ ,  $BM_2$  and  $BM_4$

Table 2 gives the characteristics of the design of  $BM_4$ . In this design (specified in VHDL), the summation of the 4 variables is made serially every clock cycle, i.e. a sample of distribution  $BM_4$  is generated every 4 clock cycles.

| Number of LCELL | Number of LAB | Maximum clock rate | Maximum output rate |
|-----------------|---------------|--------------------|---------------------|
| 434             | 1             | 98 MHz             | 24.5 MHz            |

Table 2: Characteristics of the design.

The number of LCELL includes also the LFSRs.

The overall complexity is low (10 % of the available LCELL) and the p.d.f. of the WGNG obtained by real measurements matches the theoretical p.d.f. obtained by MATLAB.

## 5. Conclusion

In this work, we use both the Central limit theorem and the Box-Muller methods for obtaining a probability density function with a (4 $\sigma$ , 1%) accuracy. Moreover, a MATLAB reference program is elaborated which takes the hardware architecture characteristics (memory size, data format,...) into account. For easy evaluation, a linear representation of the

<sup>1</sup> The parameters of table 1 have been obtained by a "try and see" method using the MATLAB program reported in the Annex.

distribution, together with its deviation from the theoretical gaussian distribution, is defined. The MATLAB code has been parameterized to allow the designer to adapt the quality of the gaussian distribution to his needs and consequently to adjust the parameters of the VHDL model.

One can note that using filtering and appropriate mathematical functions, the WGNG can also be used to emulate the Rayleigh, the Ricean channel or even more complex channels. Studies are pursued in this direction.

## References

- [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 Electronics Circuits & Systemes (ICECS'2K), Kaslik, Lebanon, Dec 2000.

## Appendix: Reference MATLAB Program

(See table 1 for the name and the value of the parameters)

```

for r=1:K % ROM fr(s)
    for s=1:pow2(q_f)-1
        rom_f(s,r)=floor(sqrt(log((s+d_f)*
            pow2(r*q_f)))*pow2(m_f));
    end;
    rom_f(pow2(q_f),r)=0;
end;

for u=1:pow2(q_g) %ROM g(x)
    rom_g(u)=floor(sqrt(2)*cos(pi*((u-
    1+d_g)*pow2(-q_g-1)))*pow2(m_g));
end;

% Initialisation of the distribution HBM
scaling = pow2(m_f + m_g - b);
max = floor(1 + rom_f(1,K)*rom_g(1)/scaling)
HBM = zeros(1,max+1);

for s=1:pow2(q_f)-1 %Construction of HBM
    for r=1:K
        for u=1:pow2(q_g)
            n=floor((rom_f(s,r)*rom_g(u)/scaling));
            HBM(n+1) = HBM(n+1) + pow2(-(r*q_f + q_g));
        end;
    end; end;

for i = 1 : max %Construction of BM
    BM(i) = 0.5*HBM(max+2-i);
    BM(2*max+2-i) = BM(i);
end;
BM(max+1)=HBM(1);

BMi = BM; % Construction of BM_N
for i = 1 : N-1
    BMi = conv(BMi , BM);
end;

```