Hardware Architecture of a Gaussian Noise Generator Based on the Inversion Method

R. Gutierrez, V. Torres, and J. Valls

***Abstract*—In this brief, we present a hardware-based Gaussian noise generator (GNG) with low hardware cost, high generation rate, and high Gaussian tail accuracy. The proposed generator is based on a piecewise polynomial approximation of the inverse cumulative distribution function (ICDF). We propose to avoid the area-demanding barrel-shifter of the ICDF approximation by means of creating a new uniform random sequence from the uniform random number generator output. The GNG architecture has been implemented in field-programmable gate array devices, and the implementation results are compared with other published designs, achieving a higher deviation with fewer hardware resources. Our GNG generates 242 Msps of random noise and achieves a tail of 13.1σ with 442 slices, two multipliers, and two Block-RAM of a Virtex-II device. The generator output successfully passed commonly used statistical tests.**

***Index Terms*—Additive white Gaussian noise (AWGN), inversion method, piecewise polynomial approximation.**

I. Introduction

THE ADDITIVE white Gaussian noise (AWGN) channel is an important block in the design of communication systems. Widely used performance parameters, such as the bit error ratio (BER) or the frame error ratio (FER), are often obtained in Monte Carlo simulations with AWGN channels, for different signal-to-noise ratios (SNRs).

Analog methods for generating Gaussian random samples are based on some physical phenomena and can generate truly random samples, but they are subject to the influence of external factors such as temperature, and they provide low throughputs. On the other hand, digital methods are preferred due to their robustness, flexibility, speed, and predictable and controllable behavior.

Monte Carlo simulations with a very high SNR, which lead to a very low BER, require the transmission and reception of a huge amount of symbols. This could be the case of the computation of very low BERs for forward error correcting codes in optical communications, as happens when the object of study is the "error floor" phenomena of low-density parity-check codes [1]. A software simulation under these assumptions is unviable. For example, taking into account that 100 to 1000 erroneous symbols are needed for the results to have a good confidence level, a simulation with an expected BER of 10^{-12} should be designed to process 10^{15} transmitted bits. This simulation would take several months in a single processor running at 2 GHz. Therefore, to achieve reasonable simulation times, hardware emulators are needed. Field-programmable gate array (FPGA) devices are well suited for the implementation of Gaussian noise generators (GNGs) as long as they provide good performance with cost effectiveness and are easily reconfigured.

Although noise values with a great magnitude rarely exist, they are precisely the ones that can cause an error in systems with a very high SNR. Theoretically, the tails of the Gaussian distribution extend to infinity, but in a real hardware implementation, this value is bounded by a limit of typically a few times the standard deviation σ of the Gaussian distribution. This limit determines the maximum simulated SNR for an AWGN.

In this brief, we present a hardware GNG with an extremely high repetition period (2^{175}) that generates 16-bit noise samples with an accuracy of one unit in the last place (ulp), and a wide output range, reaching $\pm 13.1\sigma$, which is a high value if compared with other published hardware GNG. The proposed design is based on the approximation of the inverse cumulative distribution function (ICDF) with a nonuniform segmentation scheme, as proposed by Cheung *et al.* [2]. Our main contribution is that we create a new uniform variable from the one generated by the uniform random number generator (URNG), and as a consequence of that, the barrel-shifter is eliminated, and a simpler bit-masking block is introduced to cope with the specific problem of interpolation inside of the smaller segments. This approach has a high impact on the resulting area. The GNG has been tested using Xilinx FPGA devices.

This brief is organized as follows. Section II reviews the most common algorithms used for the implementation of hardware GNG. Section III describes the method we propose. Section IV presents the implementation results and compares them with other published implementations. Finally, the conclusions of this work are summarized in Section V.

II. Background

Most digital methods for generating Gaussian random variables are based on the transformation of uniformly distributed random variables (see [3] for a complete review). Here, we summarize the most important methods for the design of hardware GNG.

Manuscript received December 23, 2011; revised April 19, 2012; accepted June 6, 2012. Date of publication July 10, 2012; date of current version August 10, 2012. This work was supported by Fondo Europeo de Desarrollo Regional and the Spanish *Ministerio de Ciencia e Innovación* under Grant TEC2008-06787 and Grant TEC2011-27916. This brief was recommended by Associate Editor Y. Ha.

R. Gutierrez is with the Departamento de Ingeniería de Comunicaciones, Universidad Miguel Hernandez, 03202 Elche, Spain (e-mail: roberto.gutierrez@umh.es).

V. Torres and J. Valls are with the Departamento de Ingeniería Electrónica, Universidad Politécnica de Valencia, 46022 Valencia, Spain (e-mail: vtorres@eln.upv.es; jvalls@eln.upv.es).

Digital Object Identifier 10.1109/TCSII.2012.2204119

A. CLT

One of the methods proposed to design a hardware GNG is the straight implementation of the central limit theorem (CLT). This theorem states that the sum of a sufficiently large number of independent uniformly distributed random variables will be approximately normally distributed. High accuracy GNGs need to sum up a large number of samples. Atiniramit [8] proposes a hardware GNG based on the CLT method and uses four linear feedback shift registers (LFSRs) with 28, 29, 30, and 31 bits, three adders, and one accumulator. The 10 least significant bits (LSBs) of each register are interpreted as a two's complement integer and produce one valid random output every 12 clock cycles by adding 48 integers of 10 bits. Andraka and Phelps [9] add 128 random bits, which is a useful approach when the fractional accuracy does not need to be high [3]. The CLT method is not suitable for high-speed applications.

B. B-M Method

The Box–Muller [15] (B-M) method has been widely used to generate Gaussian noise samples. This method is based on the transformation of two independent uniformly distributed random numbers, i.e., U(0,1), using elementary functions (sqrt(), ln(), and cos()/sin()). Usually, the B-M method is implemented followed by a CLT stage to improve the quality of the generated random samples. Boutillon *et al.* [4] were the first to propose a GNG based on the use of the B-M method combined with a CLT stage. Xilinx has developed an IP-core [16] based on the architecture proposed in [4]. The main drawbacks of Xilinx's architecture are the poor quality of the generated samples [14] and a maximum magnitude below $\pm 4\sigma$. Lee *et al.* in [5] present a hardware GNG based on the B-M method combined with a CLT stage. They use a piecewise linear interpolation for elementary functions, with a nonuniform segmentation scheme. The samples generated reach up to $\pm 6.7\sigma$. This design is optimized in [6], reaching a maximum output of $\pm 8.2\sigma$. Alimohammad *et al.* [7] propose another GNG based on the B-M method using polynomial curve approximation with a hybrid segmentation (logarithmic and uniform) scheme, reaching $\pm 9.4\sigma$.

C. ICDF

The inversion method is based on the use of the ICDF of the Gaussian distribution to transform a uniformly distributed random variable x into a Gaussian variable y through $y = \text{ICDF}(x)$. Using this method, it is possible to generate random samples for arbitrary distributions [17]. The boundary regions of the Gaussian ICDF are highly nonlinear, complicating its approximation if the output has to reach values with a magnitude several times σ.

Chen *et al.* [10] propose a hardware implementation based on the use of a lookup table to store the ICDF. This approach requires a large memory to generate Gaussian samples with a large magnitude. McCollum *et al.* [11] use linear interpolation to approximate the ICDF of the Gaussian distribution, reducing the storage needs. Cheung *et al.* [2] propose a piecewise polynomial approximation with a more efficient nonuniform segmentation scheme. With such a scheme, polynomial coefficients would differ in several orders of magnitude among the set of segments. To avoid this problem, Lee *et al.* [5] proposed storing scaling factors (powers of two) along with the coefficients and performing the scaling operation dynamically using shifters.

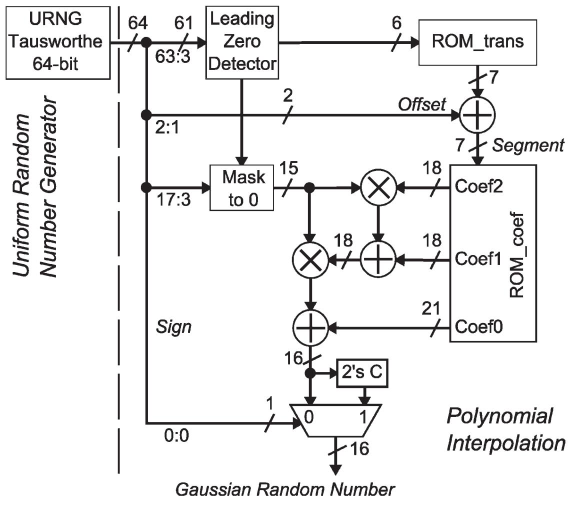

Fig. 1. Proposed 64-bit GNG architecture. Signal widths are shown for a design that uses a single Tausworthe and reaches 9.1 σ.

D. Rejection–Acceptance Methods

The rejection–acceptance methods, basically the Polar method [18] and the Ziggurat method [19], are the most important methods in software random number generation [20]. Their main drawback is the use of conditional statements that lead to a nonconstant output rate. A constant output rate can be achieved using a first-in first-out memory. Fan and Zilic [12] propose a hardware implementation of the Polar method in conjunction with a CLT stage to improve the quality of the noise samples. Its output reaches $\pm 2.88\sigma$. Zhang *et al.* [13] propose a hardware implementation of the Ziggurat method.

E. Wallace Method

The Wallace method [21] is based on the fact that linear combinations of Gaussian-distributed random numbers are also Gaussian distributed. This method avoids the evaluation of transcendental functions. A hardware implementation is proposed by Lee *et al.* [14], generating values up to $\pm 7\sigma$.

III. Proposed Method

The inversion method generates Gaussian samples via the ICDF of the Gaussian distribution. A uniformly distributed random sample $x \in \text{U}(0,1)$ is transformed into a sample y with the desired probability density function applying $y = \text{ICDF}(x)$. Fig. 1 shows the architecture of the proposed GNG. It has two main blocks: URNG and ICDF approximation via polynomial interpolation. The ICDF is symmetric with respect to the point (0.5,0); hence, we only approximate it for positive outputs. One bit from the URNG is used to randomly negate the output.

Our design uses a nonuniform segmentation scheme for the ICDF approximation. This architecture is similar to the one proposed in [2] for the hierarchical segmentation. The segmentation of the function leads to two basic blocks in the ICDF computation: a) segment selection and b) interpolation inside the selected segment. Typically, the most significant bits (MSBs) of the URNG output determine the segment in block a) and from the rest of the bits, the most significant ones are used as the input for b). By doing so, other authors compute ICDF(u), u being the value obtained if the bits of the URNG output are interpreted as an unsigned binary representation. In this case, a barrel-shifter is usually used to select the appropriate bits from the URNG output as the block b) input. When working with big word lengths, the barrel-shifter implies an important hardware cost.

In contrast to other authors, we do not compute ICDF(u). Instead, we take into account that which the ICDF needs to be calculated for a uniform variable, but not necessarily u. Our hardware computes ICDF(u'), u' being another uniform variable, which is different from u, created also from the URNG output bits. ICDF(u') can be computed more efficiently than ICDF(u), both being uniformly distributed variables with the same word length and, therefore, leading to the same output precision. Basically, the idea is that both blocks, i.e., a) and b) need independent uniform variables as input values, and by a proper selection of the bits from the URNG output, we can achieve a significant reduction in the hardware resources without spoiling the performance of the design, particularly the precision. In our design, a barrel-shifter is no longer needed, and a much simpler masking block is used to keep the inputs from a) and b) mutually independent. Obviously, this idea cannot be extrapolated to systems where a function has to be approximated for the specific values at its input. However, we think the same idea could be applied to other systems where the input is a uniform random value.

A. Implementation Details

In this brief, we present two designs with two different URNG widths: 64 and 128 bits. In both cases, the output samples have a width of 16 bits, 11 of them fractional. Our URNG uses a combined Tausworthe generator (TG) [22]. The TG combines three LFSR-based random number generators to improve the statistical properties. For the 128-bit generator, we concatenated two 64-bit TG as in [2]. According to (1) the maximum TG output value (using $x = 1 - 2^{-64}$) for the 64-bit generator sets a GNG output limit of $\pm 9.1\sigma$, and the limit for the 128-bit version is $\pm 13.1\sigma$. Thus

$$\mathrm{ICDF}(x) = \sqrt{2}\,\mathrm{erf}^{-1}(2x - 1). \tag{1}$$

The chosen TG has a periodicity of 2^{175}, which is enough to effectively reach $\pm 13.1\sigma$ values.

As previously stated, our design uses a nonuniform segmentation scheme for the ICDF approximation. We force the same interpolation polynomial order (using Chebyshev polynomials) for all the segments. A software routine was programmed to calculate the limits of each segment. The routine is analogous to the one proposed in [23, Fig. 1], but it was programmed in Python, using the mpmath library, because of the extreme precision needed. In order to use the minimum amount of embedded multipliers, we limit to 18 bits two of the interpolation

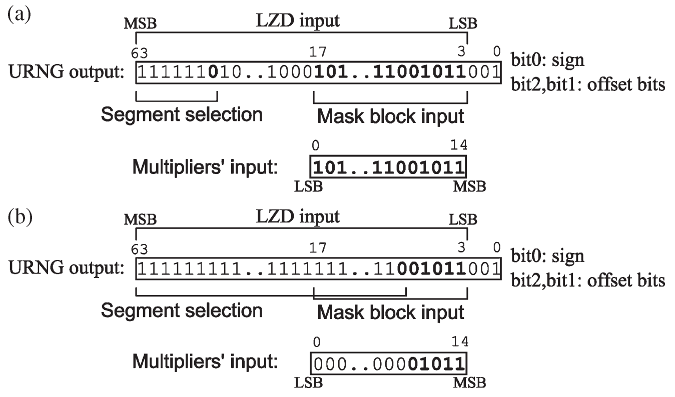

Fig. 2. Two examples of the assignation scheme for the LZD and the multipliers. (a) The leading zero appears within bits 63:18. In this case, bits 17:3 are not used for the segment identification. (b) The leading zero appears within bits 17:3. In this case, the bits 17:8 are used for segment identification; hence, they are not used as input for the multipliers.

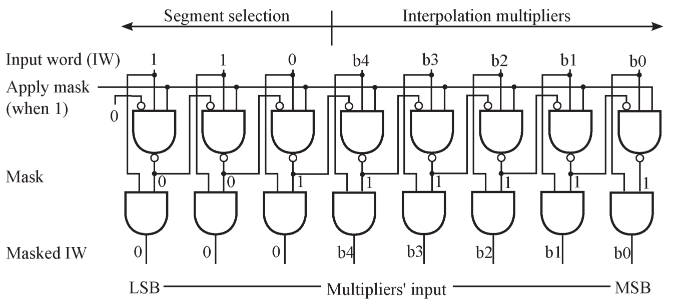

Fig. 3. Example of an 8-bit "mask to 0" block when an input word "$110b_4b_3b_2b_1b_0$" is masked.

polynomial coefficients (Coef1 and Coef2 in Fig. 1). In the mentioned routine:

- The MSBs of the input variable are the same for a whole segment, and the LSBs change from all 0s to all 1s.
- The segment's input is normalized to the range [0,1), leading to transformed coefficients, which imply smaller memory, lower hardware complexity, and higher throughput. The normalization procedure can be found in [6].
- The approximation is checked again for the transformed coefficients. If the output accuracy is not satisfied, more bits are added to the coefficients (if possible) or a smaller segment is considered.
- Following the ideas exposed in [2], we identify the segments using both the position of the leading zero in the URNG output word and a set of additional bits, called offset bits. In our design, 2 offset bits made the segments small enough to achieve the goal of one ulp accuracy.
- A table (ROM_trans) is filled with the number of the first segment for all the possible leading-zero positions.
- Another table (ROM_coef) is filled with the coefficients of the polynomial interpolation for all the segments.

As shown in Fig. 1, a leading-zero detector (LZD) is used to get the position of the most significant zero bit of the input word. Then, the segment number is calculated as the addition of the ROM_trans output and the offset bits. For example, for the case shown in Fig. 2(a), the LZD would output a 6 value, and the offset is 00; hence, the segment number would be ROM_trans(6) + 0. It should be noted that there are exceptions

TABLE I
COMPARISON AMONG SOME PUBLISHED FPGA IMPLEMENTATIONS (ALL DESIGNS USE THE SAME VIRTEX-2 DEVICE)

Design	**[16]**	**[5]**	**[7]**	**[6]**	**[12]**	**[13]**	**[14]**	**[2]**	**Our approach (64-bit)**	**Our approach (128-bit)**
Generation Method	B-M+ CLT	B-M+ CLT	B-M+ CLT	B-M	Polar +CLT	Ziggurat	Wallace	ICDF	ICDF	ICDF
Max. σ	±4.8σ	±6.7σ	±9.4σ	±8.2σ	----	----	±7σ	±8.2σ	±9.1σ	±13.1σ
Periodicity	2^{190}	2^{60}	2^{258}	2^{88}	----	2^{88}	2^{88}	2^{88}	2^{175}	2^{175}
Slices	480	2514	852	1528	336	891	852	585	388	442
Block-RAM	5	2	3	3	2	4	6	1	1	2
Mult18x18	5	8	3	12	----	2	4	4	2	2
Fmax (MHz)	245	133	248	233	73.5	170	155	231	242	242
Throughput (Msps)	245	133	496	466	146	168	155	231	242	242
Bit-width (precision)	16(11)	16(11)	16(11)	16(11)	19(12)	24(-)	32(-)	16(11)	16(11)	16(11)
Statistical Test	Not Passed	Passed	Passed	Passed	----	Passed	Passed	Passed	Passed	Passed

since some segments have only 1 offset bit, and the segment number has to be fixed for the following segments sharing the same LZD value (one unit has to be subtracted).

As shown in Fig. 1, and for the sake of efficiency, the word that feeds the LZD is composed of consecutive bits. The same is applied for the bits that feed the multipliers. Three of the URNG output bits are interpreted as sign (1 bit) and offset (2 bits). The LZD is fed by all the URNG output bits excluding the sign and offset bits. The word that feeds the LZD could be placed in whatever position, but in our design, we chose bits 63:3. Bits 2:1 are the offset, and bit 0 is the sign bit.

From the URNG output, other authors (see [2, Fig. 8] as an example) take as input for the polynomial interpolation the bits not used to identify the segment. From those bits, they use the MSBs. For example, if they used 15 bits for feeding the multipliers, in the case of Fig. 2(a), they would use bits 63:57 to identify the segment and, therefore, bits 56:42 to feed the multipliers. In contrast to that, in our designs, the bits that feed the multipliers are always taken from fixed positions of the URNG output. We chose the LSBs, i.e., bits 17:3, because they are not used for segment identification for most of the segments, as shown in Fig. 2(a). The finite-precision analysis indicated that 15 bits at the multipliers' inputs are enough to satisfy the target output precision (11 fractional bits). Since our design does not need a barrel-shifter, the area is reduced, as will be shown in Section IV. Consequently, if compared with other designs using exactly the same URNG, our design computes the ICDF for a different set of values than the ones used by other authors, but these values also constitute a uniform variable with the same properties.

In order to feed both the segment selection and the interpolation blocks with completely independent inputs, special care has to be taken for the smaller segments (i.e., those segments that have too many bits for the segment identification, hence leaving less than 15 bits for the polynomial input). In those small segments, the leading zero appears within the bits that feed the multipliers [see Fig. 2(b)], and we mask to zero the bits already used for segment identification. Therefore, they are not also used in the multiplication. For these segments, a system with a barrel-shifter would indeed feed the multipliers with the same amount of random URNG bits, using zero values for the rest of the multiplier input bits. Consequently, all of our blocks use the same input word lengths as an akin design with a barrel-shifter, reaching exactly the same precision with less hardware resources.

Fig. 3 shows the proposed scheme for an 8-bit masking block example. This block is only activated for the smaller segments [as in the example in Fig. 2(b)]. The nonzeroed bits should be used as the MSBs of the multiplier inputs. Instead of using multiplexors for this purpose, we always reverse the order of the bits that feed the multipliers, making the masked bits the LSBs.

IV. IMPLEMENTATION RESULTS

Table I compares our design with several recent hardware implementations from other authors. The FPGA device used in this comparison is a Xilinx Virtex-2 XC2V4000-6 for all the cases. For each design, we show the hardware resources used and a set of performance parameters. It should be noted that the precision of each design is expressed as the number of accurate fractional bits at the output. We included results for two implementations, i.e., one using a single 64-bit Tausworthe (see Fig. 1 for more detail) and another using 128-bit from two 64-bit Tausworthe. As shown in the table, our design reaches a higher output magnitude and requires smaller hardware than other implementations, even those that also use the inversion method, such as Cheung *et al.* [2]. Compared with the work in [2], the hardware cost of our design is significantly reduced mainly due to the lack of barrel-shifters and a segment selection strategy that guarantees the minimum amount of embedded multipliers (i.e., imposing an 18-bit width limit for the multiplier inputs). For example, our GNG that achieves $\pm 9.1\sigma$ requires one Block-RAM as in [2]; however, it just uses 388 slices and two embedded multipliers, whereas [2] reaches $\pm 8.2\sigma$ with 585 slices and four embedded multipliers for the same output precision of 11 fractional bits. One ulp precision is guaranteed in both designs. Fig. 4 shows that the absolute error is below 1 ulp in our 64-bit design.

If the target output precision is increased, the design would need more Block-RAM and more multipliers. For example, Table II compares the throughput and hardware resources needed by a 13.1σ GNG (A) using a barrel-shifter and (B) following our approach, on a Virtex-5 device for different output precision values.

The quality of the generated Gaussian noise samples has been analyzed using two statistical tests, namely, Chi-square and Anderson–Darling, with the results shown in Table III. In order to evaluate events in the tail of the distribution, the distribution has been split in several zones that were

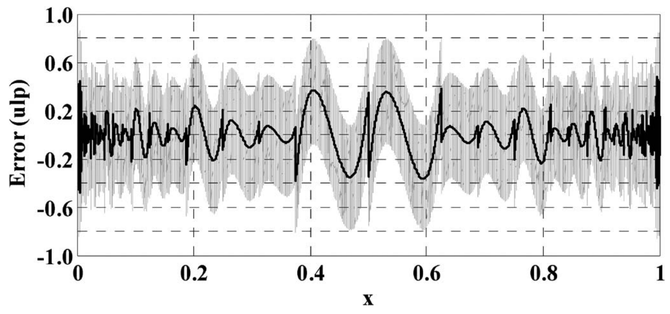

Fig. 4. Error plot in ulp in the ICDF(x) approximation using 3105 randomly selected samples for our 64-bit GNG. The black solid curve represents the systematic approximation error (with finite-precision coefficients though), whereas the gray points show the output samples' error with data-path finite-precision effects. For 93.7% of the outputs, the magnitude of the error is < 0.5 ulp, and for the remaining 6.3%, it is < 1 ulp.

TABLE II
HARDWARE RESOURCES NEEDED BY GNG REACHING 13.1σ USING (A) A BARREL-SHIFTER AND (B) OUR APPROACH, IN A VIRTEX-5 XC5VLX110T-3 DEVICE FOR DIFFERENT OUTPUT PRECISION VALUES

Precision (bits)	10	12	14	16	18	20
LUT6 (A)	722	743	814	832	873	963
(B)	315	340	382	421	448	512
Block-RAM (A,B)	1	1	1	1	2	3
Mult18x18 (A,B)	2	2	2	4	4	4
Fmax (MHz) (A)	472	472	470	450	450	449
(B)	488	488	480	462	462	460

TABLE III
STATISTICAL RESULTS

	CHI-SQUARE TEST		A-D TEST		
PDF Range	p-value	Test $\chi^2_{0.05,197}$	A'^{2}	p-value	Test A-D (α=0.05)
$\mid\sigma\mid \leq 6.2$	0.896	Passed	0.190	0.898	Passed
$6.2 <\mid\sigma\mid \leq 7.8$	0.662	Passed	0.211	0.856	Passed
$7.8 <\mid\sigma\mid \leq 9.2$	0.621	Passed	0.289	0.615	Passed
$9.2 <\mid\sigma\mid \leq 10.6$	0.515	Passed	0.325	0.523	Passed
$10.6 <\mid\sigma\mid \leq 12.0$	0.425	Passed	0.389	0.384	Passed
$12.0 <\mid\sigma\mid \leq 13.0$	0.410	Passed	0.452	0.272	Passed

independently evaluated. This is necessary due to the low probability associated with the events in the tails. The results shown in Table III indicate that the proposed GNG passed all the tests.

Finally, our design was also tested in an FPGA device as a channel noise generator for a simulated binary phase-shift keying communications system. The BER results for the system fitted theoretical values. For the longest simulation, more than 10^{14} noise samples were generated.

V. CONCLUSION

This brief has introduced a fast and compact hardware-based Gaussian number generator based on the inversion method. Our approach presents new solutions that could be used to significantly reduce the hardware resources needed by systems fed by uniform variables, e.g., the use of the inversion method for other distributions than the Gaussian.

The ICDF has been approximated using piecewise polynomial interpolation with nonuniform segmentation. In contrast to other authors, our GNG does not use a barrel-shifter, a change that produces important area savings without sacrificing precision for complexity. Furthermore, when compared with other GNG hardware implementations, ours has shown better maximum deviation and less hardware cost, with akin routing complexity. The architecture proposed has been modeled using VHSIC hardware description language and implemented in several Xilinx FPGA devices, achieving up to 488 Msps in a Virtex-5 device.

REFERENCES

[1] J. T. Richarson, "Error floors of LDPC codes," in *Proc. 41st Allerton Conf.*, Monticello, IL, Oct. 2003, pp. 1426–1435.
[2] R. C. C. Cheung, D. Lee, W. Luk, and J. Villasenor, "Hardware generation of arbitrary random number distributions from uniform distributions via the inversion method," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 8, pp. 952–962, Aug. 2007.
[3] D. B. Thomas, W. Luk, P. H. W. Leong, and J. D. Villasenor, "Gaussian random number generators," *J. ACM Comput. Surveys (CSUR)*, vol. 39, no. 4, pp. 11:1–11:38, 2007.
[4] E. Boutillon, J. L. Danger, and A. Gazel, "Design of high speed AWGN communication channel emulator," *Analog Integr. Circuits Signal Process.*, vol. 34, no. 2, pp. 133–142, Feb. 2003.
[5] D. Lee, W. Luk, J. Villasenor, and P. Y. K. Cheung, "A Gaussian noise generator for hardware-based simulation," *IEEE Trans. Comput.*, vol. 53, no. 12, pp. 1523–1533, Dec. 2004.
[6] D. Lee, J. Villasenor, W. Luk, and P. H. W. Leong, "A hardware Gaussian noise generator using the Box–Muller method and its error analysis," *IEEE Trans. Comput.*, vol. 55, no. 6, pp. 659–671, Jun. 2006.
[7] A. Alimohammad, S. F. Fard, B. F. Cockburn, and C. Schlegel, "A compact and accurate Gaussian variate generator," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 5, pp. 517–527, May 2008.
[8] P. Atiniramit, "Design and implementation of an FPGA-based adaptative filter single-use receiver," M.S. thesis, Virginia Polytechnic Inst. State Univ., Blacksburg, VA, 1999.
[9] R. Andraka and R. Phelps, "An FPGA based processor yields a real time high fidelity radar environment simulator," in *Proc. Mil. Aerosp. Appl. Programm. Devices Technol. Conf.*, 1998.
[10] J. Chen, J. Moon, and K. Bazargan, "Reconfigurable readback-signal generator based on field-programmable gate array," *IEEE Trans. Magn.*, vol. 40, no. 3, pp. 1744–1750, May 2004.
[11] J. McCollum, J. M. Lancaster, D. W. Bouldin, and G. D. Peterson, "Hardware acceleration of pseudo-random number generation for simulation applications," in *Proc. 35th Southeastern Symp. Syst. Theory*, 2003, pp. 299–303.
[12] Y. Fan and Z. Zilic, "BER testing of communication interfaces," *IEEE Trans. Instrum. Meas.*, vol. 57, no. 5, pp. 897–906, May 2008.
[13] G. Zhang, P. H. W. Leong, D. Lee, J. D. Villasenor, R. C. C. Cheung, and W. Luk, "Ziggurat-based hardware Gaussian random number generator," in *Proc. IEEE Int. Conf. Field-Programm. Logic Appl.*, 2005, pp. 275–280.
[14] D. Lee, W. Luk, J. Villasenor, G. Zhang, and P. H. W. Leong, "A hardware Gaussian noise generator using the Wallace method," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 8, pp. 911–920, Aug. 2005.
[15] G. E. P. Box and M. E. Muller, "A note on the generation of random normal deviates," *Ann. Math. Stat.*, vol. 29, no. 2, pp. 610–611, 1958.
[16] Xilinx Inc., Additive White Gaussian Noise (AWGN) core v1.02002.
[17] W. Hörmann and J. Leydold, "Continuous random variable generation by fast numerical inversion," *ACM Trans. Model. Comput. Simul.*, vol. 13, no. 4, pp. 347–362, Oct. 2003.
[18] L. Devroye, *Non-Uniform Random Variate Generation*. New York: Springer-Verlag, 1986.
[19] G. Marsaglia and W. W. Tsang, "The Ziggurat method for generating random variables," *J. Stat. Softw.*, vol. 5, no. 8, pp. 1–7, Oct. 2000.
[20] J. E. Gentle, *Random Number Generation and Monte Carlo Methods*. New York: Springer-Verlag, 2003.
[21] C. S. Wallace, "Fast pseudorandom generators for normal and exponential variates," *ACM Trans. Math. Soft.*, vol. 22, no. 1, pp. 119–127, Mar. 1996.
[22] P. L'Ecuyer, "Maximally equidistributed combined Tausworthe generators," *Math. Comput.*, vol. 65, no. 213, pp. 203–213, Jan.1996.
[23] D. Lee, W. Luk, J. Villasenor, and P. Cheung, "Hierarchical segmentation schemes for function evaluation," in *Proc. IEEE Int. Conf. Field-Program. Technol.*, Dec. 2003, pp. 92–99.