2010 International Conference on Reconfigurable Computing

A New Hardware Efficient Inversion Based Random Number Generator for Non-Uniform Distributions

Christian de Schryver, Daniel Schmidt, Norbert Wehn
Microelectronic Systems Design Research Group
University of Kaiserslautern
Erwin-Schroedinger-Str., 67663 Kaiserslautern, Germany
{schryver, schmidt, wehn}@eit.uni-kl.de

Elke Korn, Henning Marxen, Ralf Korn
Stochastic Control and Financial Mathematics Group
University of Kaiserslautern
Erwin-Schroedinger-Str., 67663 Kaiserslautern, Germany
{korn, marxen}@mathematik.uni-kl.de

***Abstract*—For numerous computationally complex applications, like financial modelling and Monte Carlo simulations, the fast generation of high quality non-uniform random numbers (RNs) is essential. The implementation of such generators in FPGA-based accelerators has therefore become a very active research field. In this paper we present a novel approach to create RNs for different distributions based on an efficient transformation of floating-point inputs. For the Gaussian distribution we can reduce the number of slices needed by up to 48% compared to the state-of-the-art while achieving a higher output precision in the tail region. Our architecture produces samples up to 8.37σ and achieves 381MHz. We also present a comprehensive testing methodology based on stochastic analysis and verification in practical applications.**

***Keywords*-Random Number Generator, Normal Distribution**

I. Introduction

Random numbers (RNs) are essential for a wide range of technical simulation applications including simulations of wireless communication systems or Monte Carlo (MC) simulations, used for example in financial modeling. Due to the high computational complexity of the aforementined applications, there is a very active research effort for FPGA accelerators, containing harware random number generators [1]–[6]. In nearly all fields the quality of the used RNs has a large impact on the achieved results and the reliability of the simulation output. While all standard random number generators (RNGs) provide uniformly distributed RNs, other distributions are necessary in many applications. Most commonly, Gaussian (normally) distributed RNs (GRNs) are needed, but other distributions such as log-normal, exponential or Rayleigh are also very common. As MC simulations are extremely computation intensive, there is an expressed need for fast and area-efficient random number generators for different distributions in FPGA-based hardware implementations, in order to achieve high processing speeds.

In this paper, we present a novel, FPGA-optimized architecture for non-uniform distributions. We show that for Gaussian random numbers our solution provides an improved output precision compared to state-of-the-art implementations while necessitating up to 48% fewer slices on a Virtex-4 FPGA. By using floating-point (FP) numbers as the input, the address generation for the LUTs is significantly simplified while allowing to easily extend the range of the Gaussian samples to 8.37σ and beyond with very little or no added overhead. Furthermore, other than previously published Gaussian random number generators (GRNGs), our proposed inversion based approach retains a very high resolution of the inverse cumulative distribution function (ICDF) even in the tail regions. We also show how to efficiently generate uniformly distributed FP numbers of arbitrary precision from the outputs of any standard RNG. Our hardware implementation can reach up to 8.37σ in the normal distribution consuming only slightly more than one 32 bit uniform RN per output sample, instead of 54 bit as required by previous approaches.

Since no standardized verification strategies for non-uniform RNGs exist we developed a comprehensive testing methodology for our new GRNG, based on stochastic analysis and practical application tests. We test the uniformity of the output of our floating-point converter with TestU01 suite [7]. The quality of the generated Gaussian random samples is assessed with several statistical tests and we verify the correctness of the simulations in two practical applications. The novel contributions of our work are:

- We show how the FP representation helps to improve precision and save area.
- We present optimized architectures for the floating-point converter and the non-uniform inversion unit.
- We subject our GRNG to an exhaustive testing methodology for non-uniform random number generators.

II. Related Work

The main methodical approaches for generating non-uniformly distributed RNs out of uniformly distributed RNs (which all standard number generators provide) are transformation, rejection, and inversion methods. A very good overview for the common task of creating Gaussian RNs is

We gratefully acknowledge the partial financial support from Center of Mathematical and Computational Modeling $(CM)^2$ of the University of Kaiserslautern.

978-0-7695-4314-7/10 $26.00 © 2010 IEEE
DOI 10.1109/ReConFig.2010.20

given in [8]. Here we will only highlight the most common methods very shortly.

A very common method in use for GRNGs is based on the Box-Muller method [9] that transforms a pair of uniformly distributed random numbers into a pair of Gaussian distributed RNs by applying trigonometric functions. One advantage of the Box-Muller method is that it deterministically provides two RNs in each step, but a lot of hardware ressources are needed to accurately calculate the trigonometric functions while achieving a high throughput [1,2].

For software implementations of GRNGs, rejection methods (as the Ziggurat method [10]) provide high quality random numbers [6]. Some hardware implementations of the Ziggurat method highly optimized for FPGAs exist as well [3,4]. They achieve up to 400 millions of samples per second at moderate ressource costs. On the downside, not every input RN is transformed into one GRN, but instead some are rejected. This behaviour may cause problems for quasi random sequences as input [5].

Inversion based methods combine many desireable properties: by applying the respective ICDF they transform every input sample $x \in (0, 1)$ from a uniform distribution to one output sample $y = icdf(x)$ of the desired output distribution. Thus, inversion is applicable also to transform quasi random sequences. As the ICDF of every distribution is continuous and monotone, it also enhances the effectiveness of variance reduction methods in MC simulations [5]. For many distributions, however, there is no known closed form expression for the ICDF and it can only be approximated. The most common approximations for the Gaussian ICDF are given by Acklam [11] and Moro [12]. Both of them are based on rational polynomials and are thus not suitable for area efficient hardware implementations.

Hardware implementations of the Gaussian ICDF based on piecewise polynomial interpolation and look-up tables (LUTs) have been presented in various publications, most notably [5] and [6]. In both these works, the ICDF was segmented in a hierarchical way, using smaller segments in the steeper parts of the ICDF. In every segment the ICDF is approximated using a minimax polynomial. This approach results in a good trade-off between accuracy and storage requirements for polynomial coefficients.

All of these inversion methods use fixed-point numbers as their inputs. We show that using floating-point RNs as input of an inversion based non-uniform RNG reduces the consumed hardware area significantly for the same output precision.

III. Inversion Based Non-Uniform Random Number Generation

A. State-of-the-Art Implementations

Starting from uniformly distributed random numbers, the inversion method can generate any favored output distribution by using the inverse cumulative distribution function

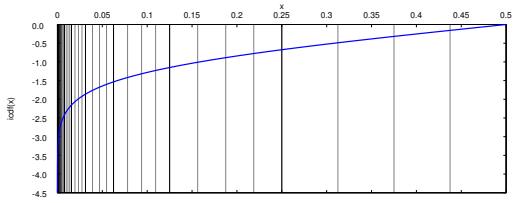

Figure 1. Segmentation of the First Half of the Gaussian ICDF

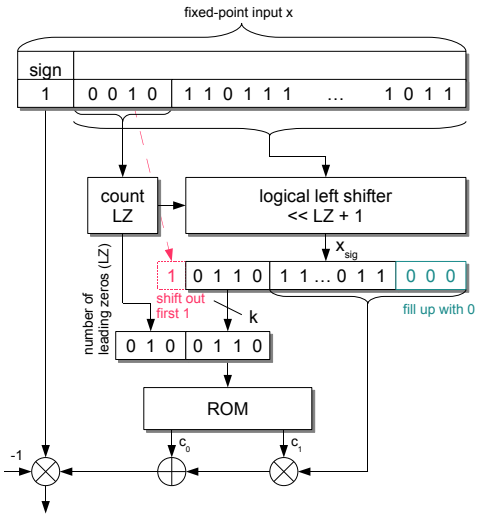

Figure 2. State-of-the-Art Architecture

$y = icdf(x)$. The ICDF of the normal distribution is centrically symmetric at $x = 0.5$, thus in this case it is sufficient to implement the inversion only for values $x \in (0, 0.5)$ and use one additional random bit to cover the full range. This part of the Gaussian ICDF is shown in Fig. 1.

One hardware efficient method to implement the inversion for various different distributions is based on piecewise polynomial approximation. In [6] the authors proposed a hierarchical segmentation scheme that allows for a good trade-off of hardware requirements and accuracy. For the normal ICDF, the range $(0, 0.5)$ is divided into non-equidistant segments with doubling segment sizes from the beginning of the interval to the end of the interval. Each of these segments is then subdivided into inner segments of equal size. Thus, the steep regions of the ICDF close to 0 are covered by more smaller segments than the regions close to 0.5, where the ICDF is almost linear. This way, a polynomial approximation of a fixed degree can be used in all segments to obtain an almost constant maximal absolute error in every segment. The inversion is performed by determining in which segment the input x is contained, retrieving the coefficients c_i of the polynomial for this segment from a LUT and evaluate the polynomial $y = \sum c_i \cdot x^i$.

Fig. 2 shows how the number of the segment (i.e., the address for the LUT) in which a given input x in fixed-point representation is located can be determined. First, the number LZ of leading zeros in the binary representation of

x is counted. Numbers starting with a 1 (half of all numbers) lie in the segment $[0.25, 0.5)$, numbers starting with the sequence 01 (one fourth of all numbers) lie in the segment $[0.125, 0.25)$ and so forth. The input x is shifted left by $LZ+1$ bits, such that x_{sig} is the bit sequence following the most significant 1-bit in x. The equally spaced subsegments are determined by the k most significant bits (MSBs) of x_{sig}. Thus, the LUT address is the concatenation of LZ and $MSB_k(x_{sig})$. The remaining bits of x_{sig} are then used to evaluate the approximating polynomial for the ICDF in that segment. Fig. 2 shows the architecture for the case of linear interpolation, as was presented in [6] for a maximum absolute error of $0.3 \cdot 2^{-11}$.

The works of Luk et al. [6,13,14] are based on this scheme. One possible segmentation arrangement for the Gaussian ICDF is shown in Fig. 1. They use it in an analogous manner to create RNGs for the log-normal and the exponential distributions, with only minor changes in the segmentation scheme. For the exponential distribution, the largest segment starts near 0, followed by segments of half the size each towards 1, and for the log-normal distribution neighbouring segments double in size between 0 and 0.5 and halve in size towards 1.

But this approach has a number of drawbacks:

- *Need for two input RNGs to achieve a large output range.* The output range is limited by number of input bits. As the smallest value larger than 0 that can be represented by an m-bit fixed-point number is 2^{-m}, the largest output value of this ICDF with a 32-bit input is $icdf(2^{-32}) = 6.33\sigma$. To achieve a larger range of up to 8.21σ, the authors of [6] concatenate the input of two 32-bit RNGs and feed a 53-bit fixed-point input into the inversion unit, at the cost of one additional RNG. This large number of input bits also increases the size of the leading-zero counter and the shifter unit, which dominate the hardware usage of the design.
- *A large number of input bits is wasted.* A multiplier with a 53-bit input for the evaluation of the polynomial would require a large amount of hardware ressources. Thus, the input is quantified to 20 significant bits before function evaluation, sufficient for an accuracy of $0.3 \cdot 2^{-11}$. Effectively, a large number of the generated input bits is wasted.
- *Low resolution in the tail region.* For the tail region, however, there are much less than 20 significant bits left after shifting over the leading zeros, which limits the resolution in the tail region. As there are no input values between 2^{-53} and 2^{-52} in this fixed-point representation, this inversion method does not generate output samples between 8.21σ and 8.13σ.

B. Floating-Point Based Inversion

The aforementioned drawbacks are all linked to properties of the fixed-point representation of the input numbers.

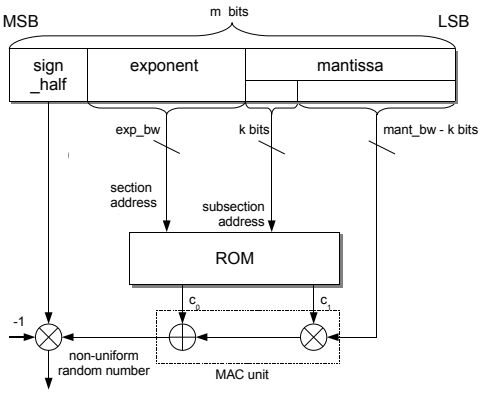

Figure 3. ICDF Look-up Unit Structure (for linear approximation)

To overcome the identified problems, we propose to use floating-point numbers as the input to the inversion unit. We do not use any FP arithmetics, as the representation is only used for the addressing of the segmentation table and calculations are only done with the mantissa.

The value of a FP number represented by a mantissa m and an exponent e is $2^e \cdot (1+m)$. The 1 being added to the mantissa is called the *hidden bit* and assures that there is only one floating-point representation for any given number. It can easily be verified, that the absolute value of a negative exponent e corresponds to the number of leading zeros of a fixed-point representation of the same value, and the hidden bit corresponds to the most significant 1-bit. Consequently, the address for the LUT to retrieve the coefficients for the segment containing a given floating-point number can be directly generated from the exponent e and the k most significant bits (MSBs) of the mantissa m.

This is shown in Fig. 3. Here, we use one additional bit $sign\_half$, to generate a symmetrical coverage of the range $(0, 1)$. This bit is interpreted in the following way:

$$x = \begin{cases} 2^e \cdot (1+m) & : \text{ if sign\_half=0,} \\ 1 - 2^e \cdot (1+m) & : \text{ if sign\_half=1.} \end{cases} \quad (1)$$

For the Gaussian ICDF the bit can be exploited as $icdf(1-x) = -icdf(x)$, as is shown in Fig. 3.

Besides enabling a very efficient address decoding for the hierarchical segmentation scheme, FP numbers offer the advantage to represent values very close to 0 with the same *relative* precision as the values close to 1. The smallest representable number is not limited by the bitwidth of the mantissa, but approaches 0 exponentially with a growing bitwidth of the exponent. E.g., a FP number with 6-bit exponent and 23 bit mantissa only uses 29 bits, but can represent values as small as 2^{-65}. Moreover, there are as many FP numbers in $[2^{-65}, 2^{-64})$ as in $[2^{-2}, 2^{-1})$, in this example 2^{23}. Thus, the tail region of the ICDF shows the same resolution as the central region and all bits of the FP input number are used for the transformation.

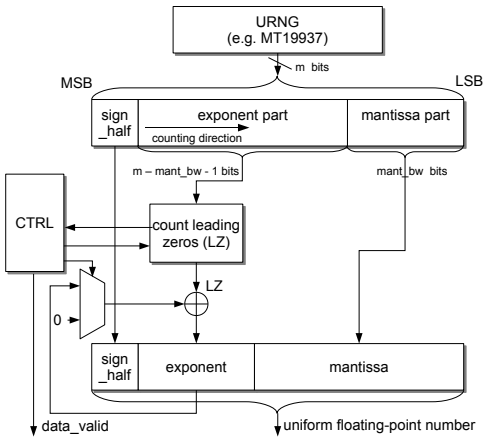

Figure 4. Architecture of the Proposed Floating-Point RNG

IV. Floating-Point Random Number Generation

In this section we present an efficient hardware optimized architecture for generating floating-point values out of bitvectors. As input for our floating-point generator (FPG) any URNG may be used. Our converter unit maintains the original properties with respect to randomness and distribution of the input RNs, as show the results in Section V-B1.

Fig. 4 shows how m input bits from URNG are splitted into the floating-point components. The mantissa part is $mant\_bw$ bits width and therefore (with a hidden bit) can have the values $1, 1 + \frac{1}{2^{mant\_bw}}, 1 + \frac{2}{2^{mant\_bw}}, \ldots, 2 - \frac{1}{2^{mant\_bw}}$. We extract one sign bit $sign\_half$ that we need to determine which half of the Gaussian ICDF shall be considered. From the exponent part, we count the number of leading zeros (LZ) and use these as the exponent in our representation of the FP number. In order to obtain IEEE754 compliant FP numbers, an offset has to be added to the exponent. For our ICDF look-up unit we employ the LZ directly as the segment address.

Initially, with this straightforward approach we would be limited to a maximum exponent value of $m - mant\_bw - 1$ with all bits in the exponent part being 0. This would not allow us to achieve the desired precision for values close to 0. Therefore we have introduced another parameter max_exp, defining the maximum value of the exponent of the FP number. If all bits in the exponent part input are detected to be 0, we store the values for sign and mantissa and consume another random number. Now we count again the LZ in the exponent part and add these number to the stored value. This is continued until a 1 is detected in the exponent part or if the number of accumulated LZ exceeds max_exp.

Pseudo code for our new FPG unit is given in Algorithm 1. The corresponding hardware architecture with the correlation of input and output components is shown in Fig. 4. The leading zero counter is efficiently implemented as a comparator tree. The amount of additionally needed RNs

$rn \longleftarrow get\_random\_number();$
$s\_half \longleftarrow rn.get\_s\_half();$
$mant \longleftarrow rn.get\_mantissa();$
$exp \longleftarrow rn.get\_exponent();$
$LZ \longleftarrow exp.count\_leading\_zeros();$
while $(exp == 0)$ *and* $(LZ < max\_exp)$ **do**
  $rn \longleftarrow get\_random\_number();$
  $exp \longleftarrow rn.get\_exponent();$
  $LZ \longleftarrow LZ + exp.count\_leading\_zeros();$
end
$exp \longleftarrow min(exp, max\_exp - 1);$
return s_half, $mant$, LZ;

Algorithm 1: floating-point Generation Algorithm

depends on the parameter max_exp and the available bits for the exponent part: a second random number is needed with the probability of $P_2 = \frac{1}{2^{m-mant\_bw-1}}$, a third with $P_3 = \frac{1}{2^{2\cdot(m-mant\_bw-1)}}$.

V. Experimental Results and Tests

A. Synthesis Results

To create the LUT with the coefficients of the minimax polynomials for every segment we developed a tool that can be used with the freeware mathematical program Scilab. This allowed us to optimize the bitwidths of the coefficients and analyze the resulting approximation errors. This tool is available from our homepage[1] and allows others to adjust the parameters (bitwidths, interpolation degree, segmentation scheme) of the LUT to their needs for precision and hardware requirements.

We optimized the bitwidths to optimally use the multiply-accumulate (MAC) unit in a Virtex-5 DSP48_E slice, which supports a $18x25\ bit + 48\ bit$ MAC operation. We used the following parameters: input bitwidth $m = 32$, $mant\_bw = 20$, $max\_bw = 54$, and $k = 3$ for subsegment addressing. The coefficients c_0 and c_1 of the linear interpolation are quantized with 46 and 23 bits, respectively.

We synthesised our proposed architecture with Xilinx ISE 11.3 for a Virtex-5FX70T-3 using default settings for synthesis, mapping, place and route (PAR). The floating-point random number occupied only 29 slices (62 slice flip-flops (FFs) and 40 slice LUTs). The complete Gaussian random number converter uses only 44 slices (109 FFs and 46 LUTs), 1 36Kb block RAM for the coefficients and 1 DSP48_E slice for the linear interpolation. The design achieves a clock frequencies (after PAR) of 381 MHz. In addition to this converter a 32-bit URNG is needed to provide the input samples.

Every Virtex-5 slice contains 4 6-input LUTs and 4 flip-flops, while on the Virtex-4 every slice containt 2 4-input LUTs and 2 flip-flops. Thus, for a fair comparison

[1] http://ems.eit.uni-kl.de/fileadmin/downloads/ICDF_LUT_generator.tgz

with the results from [6] we synthesized the floating-point converter unit for a Virtex-4LX100-12 (the same FPGA used as in [6]). It occupies 55 slices (62 FFs, 55 LUTs) and can replace the complete address decoding logic from the design in [6], which requires 221 slices. The floatin-point converter achieves clock frequencies in excess of 400 MHz with standard synthesis and PAR settings. The polynomial evaluation including the LUT for the coefficients requires on the Virtex-4 occupies 2 DSP48 slices, 2 18Kb BRAMs and 124 slices. So, with our improved architecture 166 of 346 slices needed for the Gaussian conversion can be saved. Also, the design from [6] requires two 32-bit URNGs while with our architecture one 32-bit URNG is sufficient. We achieve a better resolution of the tail and a slightly larger range of up to 8.37σ as opposed to 8.21σ. At 70 slices for a 32-bit Tausworthe generator, our approach saves 48% slices compared to the state of the art. With bigger (and better) URNGs the benefits of our architecture grow. However, in approximately one out of 1000 clock cycles, our design requires a second clock cycle to generate the FP input.

B. Quality Tests

So far no standardized test suites exist for non-uniform RNs. Therefore we split the verification of our GRNG into two parts: first we checked the quality of our floating-point generator with the TestU01 test battery, then we checked the transformed RNs for normality. Finally, we tested our RNs in two typical target applications.

1) Uniform Floating-Point Generator: The TestU01 suite [7] is a comprehensive test portfolio for uniformly distributed RNs, written in C. It mandates an equivalent fixed-point precision of at least 30 bits, for the Big Crush test set even 32 bits. We created uniformly distributed floating-point RNs in the described manner with a 31-bit mantissa from the output of a Mersenne Twister MT19937 using our proposed method and tested them with the Small Crush, Crush, and Big Crush test suites. With the exception of the two tests that the MT19937 is known to fail itself, all tests were passed. We thus conclude floating-point RN converter preserves the properties of the input generator and shows the same excellent structural properties.

For reasons of hardware efficiency, the RNs with which we feed into the ICDF look-up unit have a precision of only 23 bit. It thus has a lower resolution than the fixed-point input in some regions and a higher resolution in other regions. The good distribution for two-dimensional vectors can be seen in Fig. 5. Even zoomed in the area around zero, no patterns, clusters, or big holes can be seen. The equidistribution of our RNs was also empirically tested with several variants of the frequency test [15]. While checking the uniform distribution of the RNs up to 12 bits no extreme p-value appeared.

2) Normally Distributed Numbers: The quality of the normally distributed RNs was tested with various χ^2-tests,

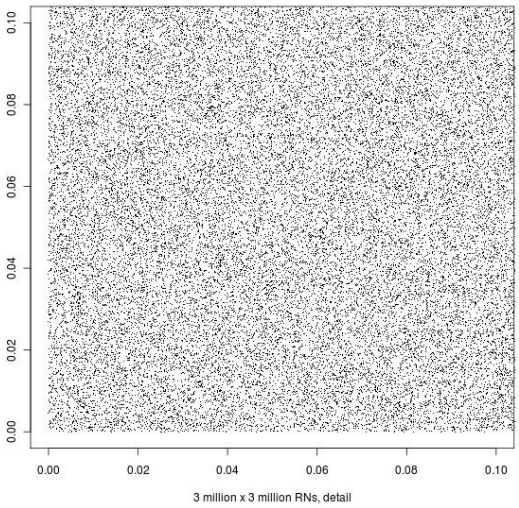

Figure 5. Detail of Uniform 2D-Vectors around 0

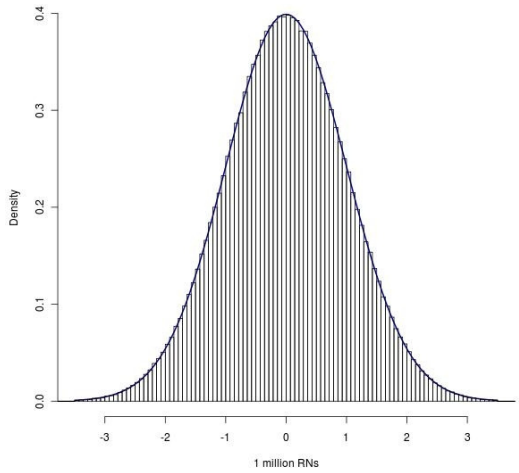

Figure 6. Histogram of Gaussian Random Numbers

which compare the empirical number of observations in several groups with the theoretical number of observations. Test results with an extremely low probability would indicate a poor quality of the RN, meaning that either the structure does not fit to a collection of independent normally distributed RNs or shows more regularity than expected from a random sequence. With respect to the precision of the RNs, we did not get suspicious results. The batch of RNs in Fig. 6 shows that the density of the normal distribution is well approximated, the corresponding χ^2-test (goodness-of-fit test) with 100 categories had a p-value of 0.37.

The distribution of the RNs was also tested in various batches with the Kolmogorov-Smirnov test, which compares the empirical with the theoretical cumulative distribution function. Nearly all tests were perfectly passed, those not passed did not reveal an extra-ordinary p-value. A refined Kolmogorov-Smirnov test, as described in Knuth [15] on page 51, sometimes had quite low p-values. We attribute this to the lower precision in some regions of our RNs,

as the continuous CDF can not be perfectly approximated with RNs with fixed gaps. Other normality tests, including a Shapiro-Wilk test, showed no deviation from normality. We also compared our RNs with the well-accomplished normally distributed RNs of R, which are also based on the Mersenne-Twister. The Kolmogorov-Smirnow tests revealed no difference in distribution. Comparison of variance with the F-test and of mean with the t-test gave no suspicious results. Our RNs seem to have the same distribution as standard RNs, with the exception of the reduced precision in the central area and the improved precision in the extreme values. As a result even the area of extreme values is evenly filled without any large gaps. As the smallest value in our FPG is 2^{-54}, values of -8.37σ and 8.37σ can be produced from only 32-bits of input. Our approximation of the ICDF is injective, symmetric, monotone, and has an absolute error of less than $0.4 \cdot 2^{-11}$. This ensures that our inversion preserves the good structure of the uniform RNs in the generated normally distributed GRNs. We expect our hardware-optimized normally distributed RNs to be especially suitable for extremely long and detailed simulations, where extreme events are of importance.

3) Application Tests: For the presented hardware implementation of our GRNG we ran application tests to verify the applicability of our GRNG in practical scenarios. In an Octave-based MC simulation for option price modeling using the Heston model we replaced the Octave RNG $randn()$ by a bit true model of our presented GRNG-hardware and observed the same convergence behavior obtaining the same results, both for options with and without barriers. Also an extensive set of simulations of the communications performance of a duo binary Turbo Code from the WiMax standard over an AWGN channel showed no significant deviations neither in the bit error rate nor in the frame error rate between the results simulated using our proposed GRNG and a standard GRNG based on the Mersenne Twister and inversion using the highly accurate Moro approximation.

VI. Conclusions

Implementing non-uniform hardware random number generators on FPGAs is a very active research field. In our work we have shown that floating-point representation for inversion based methods can save up to 48% of slices while increasing the output precision compared to state-of-the-art implementations. We have presented hardware architectures for our proposed floating-point generator and the ICDF look-up unit. Synthesis results for Xilix Virtex-4 and Virtex-5 FPGAs have been given. The quality of our random numbers has been verified with stochastic methods and two practical simulation applications in a communication system and for financial mathematics. Our architecture is suitable also for other distributions and we also will extend it for runtime adaption.

References

[1] A. Ghazel, E. Boutillon, J. Danger, G. Gulak, and H. Laamari, "Design and performance analysis of a high speed AWGN communication channel emulator," in *IEEE PACRIM Conference, Victoria, BC*. Citeseer, 2001, pp. 374–377.

[2] D.-U. Lee, J. Villasenor, W. Luk, and P. Leong, "A Hardware Gaussian Noise Generator Using the Box-Muller Method and Its Error Analysis," *Computers, IEEE Transactions on*, vol. 55, no. 6, pp. 659 –671, Jun. 2006.

[3] G. Zhang, P. Leong, D.-U. Lee, J. Villasenor, R. Cheung, and W. Luk, "Ziggurat-based hardware Gaussian random number generator," in *Field Programmable Logic and Applications, 2005. International Conference on*, 24-26 2005, pp. 275 – 280.

[4] H. Edrees, B. Cheung, M. Sandora, D. B. Nummey, and D. Stefan, "Hardware-Optimized Ziggurat Algorithm for High-Speed Gaussian Random Number Generators," in *International Conference on Engineering of Reconfigurable Systems & Algorithms, ERSA*, Jul. 2009, pp. 254–260.

[5] N. A. Woods and T. VanCourt, "FPGA acceleration of quasi-Monte Carlo in finance," in *Proc. Int. Conf. Field Programmable Logic and Applications FPL 2008*, 2008, pp. 335–340.

[6] R. C. C. Cheung, D.-U. Lee, W. Luk, and J. D. Villasenor, "Hardware Generation of Arbitrary Random Number Distributions From Uniform Distributions Via the Inversion Method," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 8, pp. 952–962, Aug. 2007. [Online]. Available: http://dx.doi.org/10.1109/TVLSI.2007.900748

[7] P. L'Ecuyer and R. Simard, "TestU01: A C library for empirical testing of random number generators," *ACM Trans. Math. Softw.*, vol. 33, no. 4, p. 22, 2007.

[8] D. B. Thomas, W. Luk, P. H. Leong, and J. D. Villasenor, "Gaussian Random Number Generators," *ACM Comput. Surv.*, vol. 39, no. 4, p. 11, Oct. 2007.

[9] G. Box and M. Muller, "A note on the generation of random normal deviates," *The Annals of Mathematical Statistics*, vol. 29, no. 2, pp. 610–611, 1958.

[10] G. Marsaglia and W. W. Tsang, "The Ziggurat Method for Generating Random Variables," *Journal of Statistical Software*, vol. 5, no. 8, pp. 1–7, 2000.

[11] P. J. Acklam. (2010, January) An algorithm for computing the inverse normal cumulative distribution function. [Online]. Available: http://home.online.no/ pjacklam/notes/invnorm/

[12] B. Moro, "The full Monte," *Risk Magazine*, vol. 8(2), pp. 57–58, February 1995.

[13] D.-U. Lee, W. Luk, J. Villasenor, and P. Y. Cheung, "Hierarchical Segmentation Schemes for Function Evaluation," in *Field-Programmable Technology (FPT), 2003. Proceedings. 2003 IEEE International Conference on*, 15-17 2003, pp. 92 – 99.

[14] D.-U. Lee, R. Cheung, W. Luk, and J. Villasenor, "Hierarchical Segmentation for Hardware Function Evaluation," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 1, pp. 103 –116, Jan. 2009.

[15] D. E. Knuth, *Seminumerical Algorithms*, 3rd ed., ser. The Art of Computer Programming. Reading, Massachusetts: Addison-Wesley, 1998, vol. 2.