Hardware Implementation of Rayleigh and Ricean Variate Generators

Amirhossein Alimohammad, *Member, IEEE,* Saeed Fouladi Fard, Bruce F. Cockburn, *Member, IEEE,*

***Abstract*—Compact and fast implementations of digital Rayleigh and Ricean variate generators are presented. Polynomial curve fitting is utilized along with a combination of logarithmic and uniform domain segmentation to provide accuracy, compactness and fast variate generation. A typical instantiation of the proposed Rayleigh generator occupies 124 $(< 1\%)$ of the configurable slices, two dedicated multipliers $(< 1\%)$, and one on-chip block memory $(< 1\%)$ of a Xilinx Virtex-5 field-programmable gate array (FPGA) and operates at 317 MHz, generating 317 million Rayleigh variates per second. The Ricean variate generator implementation on the same device utilizes 366 $(< 1\%)$ of the logical slices, three on-chip block memories $(< 1\%)$, and 11 (2.8%) of the dedicated multipliers. The application of the Rayleigh and Ricean variate generators is demonstrated in a FPGA-based bit error rate simulator that measures at hardware speeds the symbol error rate performance of a typical wireless communication system over Rayleigh and Ricean fading channels.**

I. Introduction

THE Gaussian, Rayleigh and Ricean distributions have been applied to model and simulate a variety of different scientific and engineering systems. An especially important application of these distributions is to model wireless fading channels. The classical model of a communication channel is the additive white Gaussian noise (AWGN) channel, where the transmitted signal $s(t)$ is corrupted by the addition of white Gaussian noise $n(t)$ thereby producing a received signal $y(t) = s(t) + n(t)$ [1]. In a more accurate model of wireless channels, the received complex envelope is expressed as $y(t) = g(t)s(t)+n(t)$, where the fading gain $g(t)$ is a complex Gaussian random variable with independent quadrature components. If this fading process has a zero (non-zero) mean then the envelope $|g(t)|$ of the gain has the Rayleigh (Ricean) distribution [2].

The radio channel is usually the key factor that limits the performance of a wireless communication system. System performance is commonly characterized through the symbol error rate (SER) versus signal-to-noise ratio (SNR) relationship and this is typically measured experimentally using Monte Carlo (MC) simulations on workstations. Wireless communication systems are increasingly complex and the number of possible operating modes that must be verified has increased dramatically. As the number of possible operating modes increases (e.g., more than 300 modulation and coding schemes are present in the IEEE 802.11n standard), the bit-true fixed-point MC simulation times become the bottleneck to timely product design and verification.

Hardware-based simulation of digital communication systems offers significant speedups compared to software simulations with no significant loss in accuracy [3], [4]. Recently, several hardware implementations of Gaussian variate generators have been proposed (see [5], [6] and their references). However, hardware implementations of other important distributions, such as the Rayleigh and Ricean distributions, have received far less attention [7], [8]. This brief extends our earlier work on designing Gaussian variate generators (GVGs) [6]. We now present high-throughput and compact Rayleigh and Ricean variate generators that are suitable for implementation on a field-programmable gate array (FPGA). We utilize the Box-Muller (BM) algorithm [9] to efficiently implement a Rayleigh variate generator. This generator is then enhanced to generate variates with the Ricean distribution. The Ricean variate generator can in turn be used to generate variates for two other important distributions: the Gamma distribution and the Chi-squared distribution with two degrees of freedom. The Gamma and Chi-squared distributions have been used to model interference in wireless communication systems [10].

The sequel is organized as follows. Section II presents our FPGA implementation of the Rayleigh variate generator. Section III presents the new Ricean variate generator. The implementation costs and simulation results are presented in Section IV. Concluding remarks appear in Section V.

II. Rayleigh Variate Generator

Let n_i and n_q be two independent normally-distributed variates with zero means and equal variance σ^2. The variable defining the magnitude $r = \sqrt{n_i^2 + n_q^2}$ has a Rayleigh distribution with mean $\sigma\sqrt{\pi/2}$ and variance $(4-\pi)\sigma^2/2$ [11]. To implement a Rayleigh variate generator, instead of generating two independent Gaussian variables, n_i and n_q, and then computing the magnitude of the complex Gaussian-distributed variate $n = n_i + j\, n_q$, where $j^2 = -1$, we use the well-known BM algorithm. According to this algorithm, if u_1 and u_2 are two independent uniformly-distributed pseudorandom numbers (PNs) in the interval $(0, 1)$ and $f(u_1) = \sqrt{-2\ln(u_1)}$, then $n_1 = f(u_1) \times \sin(2\pi u_2)$ and $n_2 = f(u_1) \times \cos(2\pi u_2)$ are two independent variates from a zero-mean, unit-variance Gaussian distribution $\mathcal{N}(0, 1)$. Therefore the variate $r = \sqrt{n_1^2 + n_2^2} = f(u_1)$ follows the Rayleigh distribution.

The authors in [8] also use the BM algorithm to generate Rayleigh-distributed variates, but they employ the iterative CORDIC algorithm to implement the logarithm and square root operations in $f(u_1)$. For a higher throughput realization, following the design procedures developed independently in [12] and [6], we adopt a hybrid segmentation scheme over the full domain $u_1 \in (0, 1)$. First, the subintervals $(0, 0.5)$ and $(0.5, 1)$ are segmented logarithmically into ℓ_1 segments from 0.5 down to 0 and from 0.5 up to 1, respectively. Then each

segment is subdivided uniformly into ℓ_2 subsegments. In this work we utilize $\ell_1 = 62$ logarithmic segments and $\ell_2 = 8$ uniform sub-segments within each logarithmic segment. Then we utilize a linear polynomial $a_1 u_1 + a_0$ to approximate $f(u_1)$ within each segment. A polynomial curve fitting approach [13] was also utilized to approximate $f(u_1)$ with a polynomial within each segment. The optimized coefficients of each polynomial were calculated using the orthogonal least squares fit (OLSF) method [14] to minimize the residual error.

For a 32-bit representation of u_1, the coefficients of $f(u_1)$ are of the order of 2^{32}. However, the required values of $f(u_1)$ lie in $[0, 6.66]$, which can be represented sufficiently accurately in two's complement 16-bit fixed-point format with a 12-bit fraction. Storing the large coefficient values of $f(u_1)$ on-chip requires relatively large memories, increases the hardware complexity and likely slows down the variate generation rate. To overcome these problems, only suitably scaled 16-bit coefficients of $f(u_1)$ for all segments are stored in on-chip memory. An experimental range analysis of the values of a_1 and a_0 shows that these values can be represented within 16 bits in the *s16.14* and *s16.12* formats, respectively, where "s $w_1.w_2$" denotes that the variable is signed, with word length w_1 and fraction length w_2.

Using the scaled coefficients and, correspondingly, the scaled value of u_1, denoted by $\tilde{u}_1$, the linear polynomial approximation $f(u_1) = (\tilde{a}_1 \times \tilde{u}_1) + \tilde{a}_0$ is representable using 16-bit fixed-point numbers. Fig. 1 shows the hardware datapath and the fixed-point format of the intermediate signals for generating Rayleigh-distributed variates using a linear approximation. The PRNG block generates 32-bit pseudo-random uniformly-distributed variates. According to the value

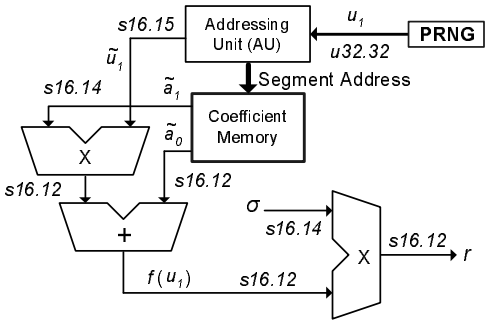

Fig. 1. The dataflow diagram for generating Rayleigh variates.

of u_1, an Addressing Unit (AU) calculates the scaled value of u_1 and generates a signed value of $\tilde{u}_1$ in *s16.15* format (to be multiplied by the signed value $\tilde{a}_1$). The AU also produces the segment address. The coefficient memory stores the $2\ell_1\ell_2 = 992$ scaled coefficients $\tilde{a}_1$ and $\tilde{a}_0$ in the 16-bit fixed-point format. Then the scaled coefficients of the linear piece can be addressed and read directly from coefficient memory to approximate $f(u_1)$. Finally, the result of $f(u_1) = (\tilde{a}_1 \times \tilde{u}_1) + \tilde{a}_0$ is multiplied by σ to support Rayleigh variates with variable noise variances. Note that even though the σ is represented in *s16.14* format (i.e., $\sigma < 2.0$), it could be represented with fewer fraction bits to support $\sigma \geq 2.0$.

To form a PRNG with equally-distributed output integers and a long period, instead of using a conventional Tausworthe generator, as in [8], we utilize a twisted generalized feedback shift register generator [15]. The 32-bit T800 uniform random number generator provides a long period of $2^{800} - 1$ and improved equi-distribution properties compared to a simple Tausworthe generator. The datapath of T800, as shown in Fig. 2, requires a dual-port memory and only shift right ($\gg$) and bitwise logical operations. The memory contains an array of 25 32-bit initial seeds and is implemented in read-before-write mode. In this mode, at every clock cycle two state values are read (always one at a constant relative offset OffsetM from the first one) before a new state is written into one of the addressed locations. The address line addrRW for the read/write port and the address line addrR for the read port are driven by mod 25 counters. The 5-bit address addrRW is initialized to zero and increments up to 24 and wraps around to address 1. Also, the 5-bit addrR is initialized with OffsetM=7 and increments to 24 and resets to 1. The parameters used in the datapath of the T800 PRNG in Fig. 2 are **a**= 8ebfd028, (s,**b**)=(7,2b5b2500), (t,**c**) = (15,db8b0000), and l=16 [15].

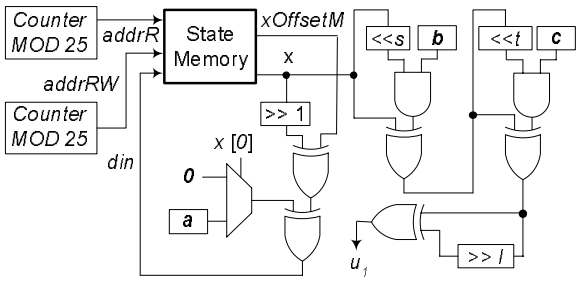

Fig. 2. Block diagram of the T800 PRNG.

The implementation of the T800 datapath on a Xilinx Virtex-5 5VFX200TFF1738-2 FPGA uses only 51 configurable slices while operating at 589 MHz, generating 589 million 32-bit uniformly-distributed random variates per second. The implementation of the Rayleigh variate generator datapath in Fig. 1 on the same FPGA requires 124 configurable slices ($< 1\%$), two of the DSP48E modules ($< 1\%$), and one of the block memories ($< 1\%$) and generates 317 million Rayleigh variates per second. Fig. 3 shows that the probability density function (PDF) of the magnitude of 4×10^7 generated Rayleigh variates closely matches the theoretical Rayleigh PDF. Fig. 4 plots on a logarithmic scale the absolute error when approximating $f(u_1)$ using $\ell_1 = 62$ for $\ell_2 = 8$ and 16 uniform subsegments. As there is relatively little improvement in the accuracy of $f(u_1)$ from $\ell_2 = 8$ to $\ell_2 = 16$, we chose $\ell_1 = 62$ and $\ell_2 = 8$ and used a smaller memory to store the scaled coefficients $\tilde{a}_1$ and $\tilde{a}_0$ of the $2\ell_1\ell_2$ segments.

III. Ricean Variate Generator

If n_i and n_q are two independent Gaussian-distributed random variates with equal variance σ^2 and non-zero means $\mu_i = \nu \sin(2\pi\theta)$ and $\mu_q = \nu \cos(2\pi\theta)$, respectively, where $\nu \geq 0$ and θ is a uniformly-distributed random variable within $(0, 1)$, then $c = \sqrt{(\sigma n_i + \mu_i)^2 + (\sigma n_q + \mu_q)^2}$ follows the

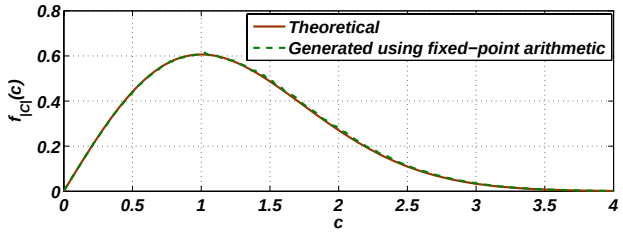

Fig. 3. PDFs of the generated and theoretical Rayleigh distributions using 4×10^7 generated samples.

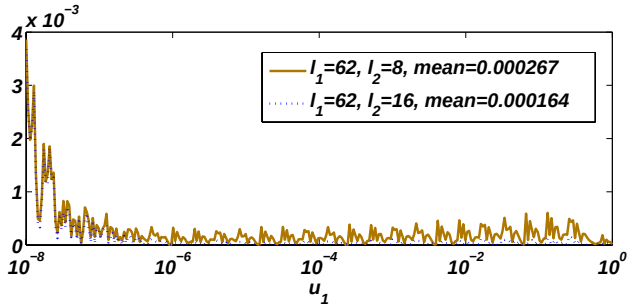

Fig. 4. Absolute approximation error of $f(u_1)$ for different numbers of uniform segments.

one can exploit the relationship between these trigonometric functions to utilize a smaller memory at the expense of some additional logic. To generate two independent variates n_i and

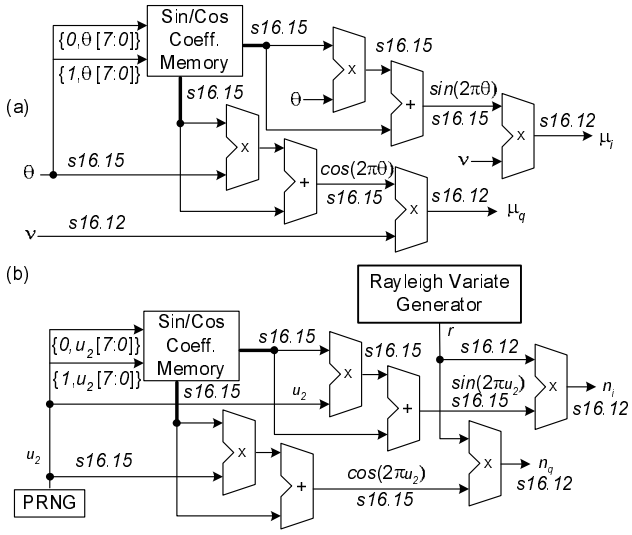

Fig. 5. The datapath for (a) generating the means of complex Gaussian variates and (b) the Gaussian variate generator.

Ricean distribution [11]. When $\nu = 0$, the Ricean distribution reduces to the Rayleigh distribution.

A general approach for generating random samples from an arbitrary distribution is the inversion method [7]. Note that a random sample x with a cumulative distribution function (CDF) F can be generated as $x = \mathrm{F}^{-1}(u)$, where u is a uniform random variate between zero and one. In [16] an approximation for the zeroth-order modified Bessel function of the first kind is used to approximate the closed-form of the CDF of a Ricean random variable. While the Ricean CDF inverse $c = \mathrm{F}^{-1}(u)$ is difficult to solve directly, a MC method was applied in [16] to generate Rice random variables.

To implement the Ricean variate generator, we start by computing the means μ_i and μ_q. While there are various standard techniques for approximating trigonometric functions, such as the CORDIC algorithm [17] and direct table-lookup [18], the choice of implementation involves balancing such requirements as throughput, latency, and logical resource costs as well as the target accuracy. Fig. 5 (a) shows a datapath that generates two mean values μ_i and μ_q by approximating $\sin(2\pi\theta)$ and $\cos(2\pi\theta)$ (where θ is an external input) using uniform segmentation of the domain $[0, 2\pi]$ and a linear approximation within each segment. By splitting $[0, 2\pi]$ into 256 segments, the Sin/Cos Coeff. Memory in Fig. 5 (a) is configured as 512×32, where the two coefficients of every linear polynomial are represented in *s16.15* format. In the variate mean generator in Fig. 5 (a), $\{0, \theta[7:0]\}$ and $\{1, \theta[7:0]\}$ address the coefficients of the 256 segments of the sine and cosine functions, which are stored in the first and second half of the 512×32-bit Sin/Cos Coeff. Memory, respectively. Note that one may share one multiplier and one adder in the datapath in Fig. 5 (a) to implement both the sine and cosine functions at the expense of lower throughput and greater latency. Also, n_q with the Gaussian distribution $\mathcal{N}(0, \sigma^2)$, as shown in Fig. 5 (b), the generated Rayleigh variate r is multiplied by $\sin(2\pi u_2)$ and $\cos(2\pi u_2)$, respectively, where u_2 is uniformly-distributed within $(0, 1)$. The sine and cosine functions are again approximated using uniform segmentation and linear polynomials. Note that since the polynomial coefficients are signed values and thus signed multipliers are required, u_2 and similarly θ in Fig. 5 (a), are represented as signed numbers in *s16.15* format.

Fig. 6 shows the datapath of the Ricean variate generator. It receives independent Gaussian variates n_i and n_q with equal variance σ^2 and means of μ_i and μ_q, respectively, all in *s16.12* format, and generates a random variate c with the Ricean distribution. The datapath is divided into two parts. Part (a) generates two biased (i.e., non-zero mean) Gaussian variates $b_i = \sigma n_i + \mu_i$ and $b_q = \sigma n_q + \mu_q$. When $\sigma = 1$, the datapath generates the random variate $\chi = (n_i + \mu_i)^2 + (n_q + \mu_q)^2$, which has a non-central chi-square distribution with two degrees of freedom and non-centrality parameter $\nu^2 = \mu_i^2 + \mu_q^2$ [11]. Accumulation of generated random values χ from the datapath in Fig. 6 (a) yields a random variate with the gamma distribution and the square root of χ yields a random variate with the Ricean distribution.

To approximate $c = \sqrt{\chi}$, we first note that the values of $r = f(u_1)$ for a 32-bit u_1 lie within the interval $[0, 6.66)$. Therefore, the two Gaussian variates n_i and n_q lie within $(-6.66, 6.66)$. For $\sigma \leq 1$, the value of random variate χ lies within $[0, 128)$. We also note that the square root function has a high-slope region near $\chi = 0$. Thus, similar to the segmentation scheme for function $f(u_1)$ in Section II, as shown in Fig. 7, the interval $[0, 128]$ is first segmented logarithmically from $\chi = 128$ down to 0 into ℓ_1 segments and then each segment is sub-divided further into ℓ_2 uniform subsegments.

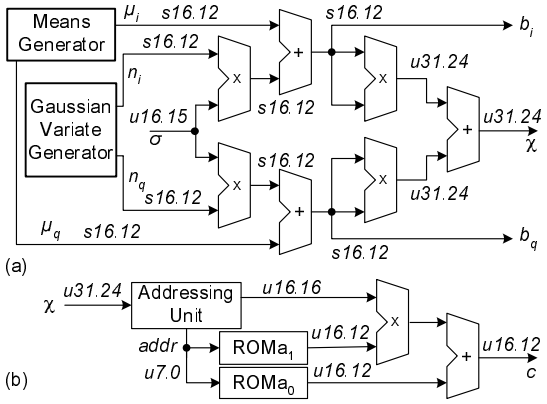

Fig. 6. Ricean variate generator datapath; (a) generating the random variate χ with non-central chi-square distribution and two degrees of freedom, (b) approximating $\sqrt{\chi}$.

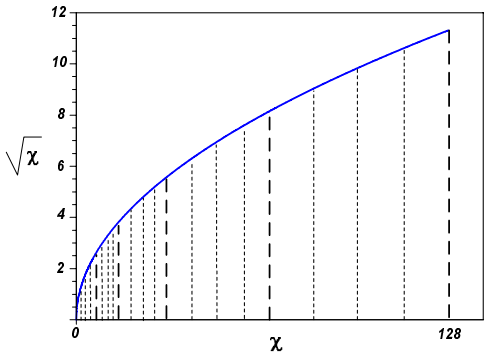

Fig. 7. Hybrid segmentation example of the square root function using five logarithmic segments and four uniform segments within each logarithmic segment.

Then the value of $\sqrt{\chi}$ within each subsegment is approximated using a linear function. We found that $\ell_1 = 16$ and $\ell_2 = 8$ gave sufficient accuracy. To approximate the coefficients a_1 and a_0 of the fitting polynomial $c = a_1\chi + a_0$ for each subsegment, we utilize linear regression. As shown in Fig. 6 (b), the coefficients a_1 and a_0 of 128 linear polynomials are stored in distributed memories ROMa_1 (in *u16.16*) and ROMa_0 (in *u16.12*), respectively. The Addressing Unit returns the 7-bit address for ROMa_1 and ROMa_0 as well as returning the scaled value of χ. The relatively small approximation errors shown in Fig. 8 demonstrate the effectiveness of our segmentation scheme in equally limiting the approximation error over the interval $[0, 128]$. Fig. 9 shows the PDF's of the generated and theoretical Ricean distributions with $\sigma = 0.25$ and $\nu = 0.5$ using 8×10^4 generated samples. Note that the PDF of the generated variates closely follows the Ricean distribution.

IV. Simulation and Implementation Results

Table I summarizes the implementation results for the key modules on a Xilinx Virtex-5 5VFX200TFF1738-2 FPGA. Our Ricean variate generator is at least four orders of magnitude faster than the software model implemented using our optimized fixed-point library on an Intel Core 2 Quad processor running at 1.58 GHz and 3 GB of RAM. On a Xilinx Virtex-4 XC4VLX200-FF1513-11 FPGA, the Ricean variate generator

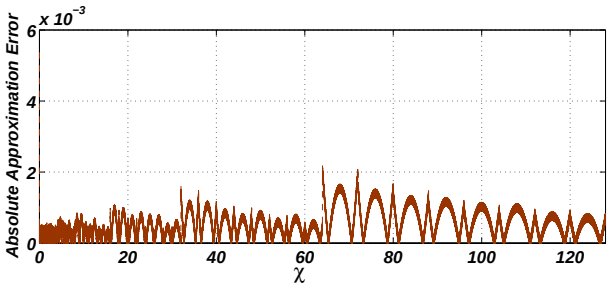

Fig. 8. Approximation error of the fixed-point square root error using 16 logarithmic segments and 8 linear subsegments within each logarithmic segment.

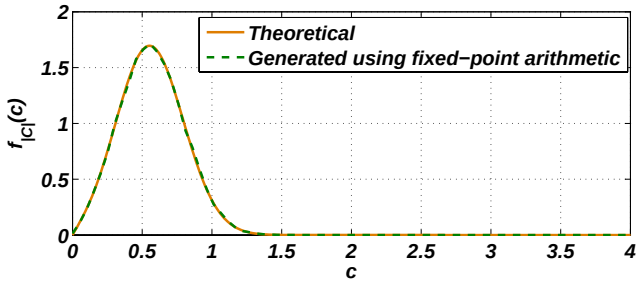

Fig. 9. PDFs of the generated and theoretical Ricean distributions with $\sigma = 0.25$ and $\nu = 0.5$ using 8×10^4 generated samples.

uses 1002 (1.1%) of the logical slices, 3 (< 1%) dedicated memory blocks, 11 (11.4%) on-chip 18×18-bit multipliers and runs at 275 MHz. On a Xilinx Spartan XC3S4000L-4FG900 FPGA, the Ricean variate generator uses 1044(3.7%) of logical slices, 3(3.1%) memory blocks, 15(15.6%) 18×18-bit multipliers and runs at 116 MHz. Note that both the Rayleigh and Ricean variate generators are especially compact, which should make them convenient for implementation along with the other signal processing modules of a wireless communication system, quite possibly on a single FPGA. We utilized our fixed-point Rayleigh and Ricean variate generator in a FPGA-based error rate measurement system to evaluate the performance of a simple wireless communication system. This system uses 16-QAM modulation with Gray coding at the transmitter. The receiver uses a minimum mean-square error (MMSE) detector and channel state information is assumed to be available at the receiver. Fig. 10 plots the theoretical and measured symbol error rates over a Rayleigh fading channel for different E_b/N_o values, where E_b is the bit energy and N_o is the noise power spectral density. Fig. 11 plots the theoretical and measured SER over a Ricean fading channel, where the Ricean factor is assumed to be 10 dB and the maximum Doppler frequency is 100 Hz. These plots show that our Rayleigh and Ricean variate generators can be efficiently used in the hardware-accelerated error rate measurement of wireless communication systems over Rayleigh and Ricean fading channels. FPGA-based baseband performance verification of wireless communication systems under realistic fading channel models greatly speeds up the performance evaluation of wireless communication systems at high SNR regions and/or systems utilizing strong channel codes, such as low-density parity-check codes (LDPCs), which require very lengthy simulations.

TABLE I
CHARACTERISTICS OF THE IMPLEMENTED MODULES ON A XILINX VIRTEX-5 FPGA

Module	T800	Sine/ Cosine	Means generator	Square root	Rayleigh	Gaussian	Ricean
Output rate (Msamp/sec)	589	404	404	419	317	317	317
Configurable slices	51	18	16	143	124 (< 1%)	197 (< 1%)	366 (< 1%)
DSP48E modules	-	2	4	1	2 (< 1%)	6 (1.5%)	11 (2.8%)
On-chip memory blocks	–	1	1	–	1 (< 1%)	2 (< 1%)	3 (< 1%)

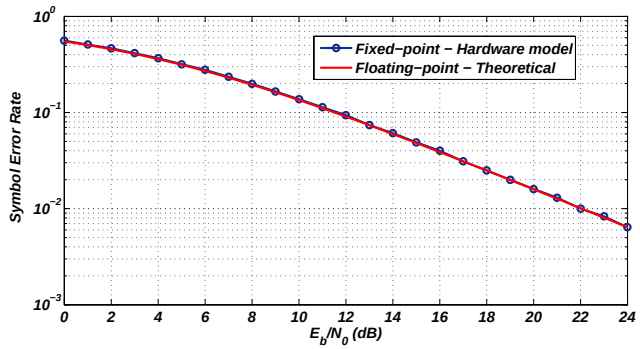

Fig. 10. Theoretical and measured SER of a 16-QAM modulated communication system over a Rayleigh fading channel.

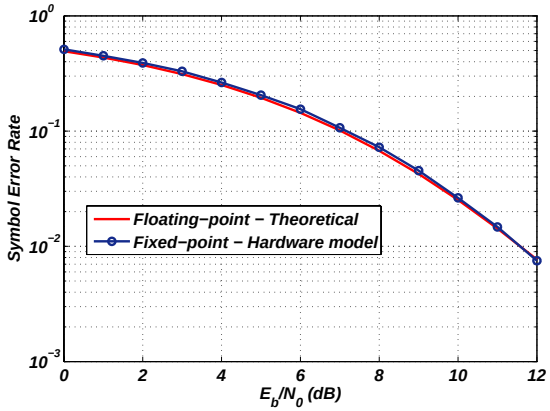

Fig. 11. Theoretical and measured SER of a 16-QAM modulated communication system over a Ricean fading channel with a Ricean factor of 10 dB and a maximum Doppler frequency of 100 Hz.

V. CONCLUSIONS

This brief presented efficient implementations of random variate generators with accurate Rayleigh and Ricean distributions. These two distributions are widely used in communication system analysis, such as in the modeling and simulation of wireless radio channels. We utilized a hybrid domain segmentation, which combined logarithmic and uniform segmentations as well as polynomial curve fitting to accurately approximate required non-linear functions. On a Xilinx Virtex-5 FPGA, the Rayleigh and Ricean variate generators use less then 1% of the logical slices, less then 1% of on-chip memories, 2.8% of DSP48E modules and generate 317 million samples per second. The sample generation rate is at least four orders of magnitude faster than a fixed-point software model implemented on a 1.58 GHz Intel Core 2 Quad processor. The Rayleigh and Ricean variate generators were implemented and used in a hardware-based error rate measurement system to evaluate the performance of a typical wireless communication system. The compactness and fast output rate of these two fading variate generators, together with their compatibility with rapid prototyping design verification using FPGAs, should make the generators an attractive tool in the hardware-accelerated verification of modern wireless systems.

REFERENCES

[1] J. G. Proakis, *Digital Communications*. 4th ed., McGraw-Hill, 2001.

[2] G. L. Stüber, *Principles of Mobile Communication*. New York: Kluwer Academic Publishers, 2001.

[3] A. Alimohammad, S. F. Fard, and B. F. Cockburn, "Hardware-based error rate testing of digital baseband communication systems," in *IEEE Int. Test Conference*, 2008, pp. 1–10.

[4] ——, "FPGA-based accelerator for the verification of leading-edge wireless systems," in *IEEE Int. Design Automation Conference*, 2009, pp. 844–847.

[5] D.-U. Lee *et al*., "A hardware Gaussian noise generator using the Wallace method," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 8, pp. 911–920, 2005.

[6] 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, 2008.

[7] W. Hörmann and J. Leydold, "Continuous random variate generation by fast numerical inversion," *ACM TOMACS*, vol. 13, no. 4, pp. 347–362, 2003.

[8] C. Wei, L. Chengshu, and S. Xin, "FPGA implementation of universal random number generator," in *IEEE ICSP*, 2004, pp. 495–498.

[9] G. E. P. Box and M. E. Muller, "A note on the generation of random normal deviates," *The Annals of Math. Statistics*, vol. 29, pp. 610–611, 1958.

[10] P. Li and D. Paul and R. Narasimhan and J. Cioffi, "On the distribution of SINR for the MMSE MIMO receiver and performance analysis," *IEEE Trans. Inf. Theory*, vol. 52, no. 1, pp. 271–286, 2006.

[11] A. Papoulis and S. U. Pillai, *Probability, Random Variables and Stochastic Processes*. McGraw-Hill, 2002.

[12] D.-U. Lee, W. Luk, J. Villasenor, and P. Cheung, "Hierarchical segmentation schemes for function evaluation," in *IEEE International Conference on Field-Programmable Technology*, 2003, pp. 92–99.

[13] *MATLAB Curve Fitting Toolbox, User's Guide*, The Mathworks, 2007.

[14] N. Chernov, C. Lesort, and N. Simanyi, "On the complexity of curve fitting algorithms," *Journal of Complexity*, vol. 20, no. 4, pp. 484–492, August 2004.

[15] M. Matsumoto and Y. Kurita, "Twisted GFSR generators," *ACM Trans. on Modeling and Computer Simulation*, vol. 2, no. 3, pp. 179–194, 1992.

[16] M. Zhuang, D. Guo, and B. Wu, "Method of generating statistical Rice random variables over wireless fading channels," in *IEEE CAS Symp. on Emerging Technologies: Frontiers of Mobile and Wireless Communication*, 2004, pp. 165–168.

[17] J. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. on Electronic Computers*, vol. EC-8, no. 3, p. 3.4.1, 1959.

[18] J.-M. Muller, *Elementary Functions. Algorithms and Implementation*. Birkhauser, 1997.

ACKNOWLEDGMENT

Bruce F. Cockburn wishes to acknowledge the support of the Natural Sciences and Research Council of Canada (NSERC) and the Alberta Informatics Circle of Research Excellence (iCORE).