Hardware Generation of Arbitrary Random Number Distributions From Uniform Distributions Via the Inversion Method

Ray C. C. Cheung, *Student Member, IEEE*, Dong-U Lee, *Member, IEEE*, Wayne Luk, *Senior Member, IEEE*, and John D. Villasenor, *Senior Member, IEEE*

***Abstract*—We present an automated methodology for producing hardware-based random number generator (RNG) designs for arbitrary distributions using the inverse cumulative distribution function (ICDF). The ICDF is evaluated via piecewise polynomial approximation with a hierarchical segmentation scheme that involves uniform segments and segments with size varying by powers of two which can adapt to local function nonlinearities. Analytical error analysis is used to guarantee accuracy to one unit in the last place (ulp). Compact and efficient RNGs that can reach arbitrary multiples of the standard deviation σ can be generated. For instance, a Gaussian RNG based on our approach for a Xilinx Virtex-4 XC4VLX100-12 field-programmable gate array produces 16-bit random samples up to 8.2σ. It occupies 487 slices, 2 block-RAMs, and 2 DSP-blocks. The design is capable of running at 371 MHz and generates one sample every clock cycle.**

***Index Terms*—Algorithms implemented in hardware, automatic synthesis, Chebyshev approximation and theory, computer arithmetic, elementary function approximation, error analysis, gate arrays, piecewise polynomial approximation.**

I. Introduction

RANDOM numbers are key components in large scale simulations across many applications including communications [1], ray tracing [2], and financial modeling [3]. Clearly, the quality of random numbers plays a central role in ensuring that simulation results are meaningful. Although the most commonly used random number distributions are uniform and Gaussian, there are many cases in which random samples drawn from log-normal, exponential, Rician, Rayleigh, or other distributions are of interest. In the communications field, for example, noise models are highly dependent on the specific propagation environment, and are quite often non-Gaussian in nature. Thus, there is a need for fast and accurate methods for generating samples corresponding to distributions appropriate for the target environment and application.

Manuscript received June 23, 2006; revised March 2, 2007. This work was supported in part by the Croucher Foundation, by Xilinx Inc., by the U.K. Engineering and Physical Sciences Research Council under Grant EP/C509625/1, Grant EP/C549481/1, and Grant GR/R 31409, by the Office of Naval Research under Contract N00014-06-1-0253, and by the National Science Foundation under Grant CCR-0120778 and Grant CCF-0541453.

R. C. C. Cheung and W. Luk are with the Department of Computing, Imperial College London, London SW7 2AZ, U.K. (e-mail: r.cheung@imperial.ac.uk; w.luk@imperial.ac.uk).

D. Lee and J. D. Villasenor are with the Electrical Engineering Department, University of California, Los Angeles, CA 90095 USA (e-mail: dongu@icsl.ucla.edu; villa@icsl.ucla.edu).

Digital Object Identifier 10.1109/TVLSI.2007.900748

While there is a long and rich history of work relating to nonuniform random number generation [4], the overwhelming majority of this paper has targeted software implementations where high precision is easily accessible. However, the higher speed offered by hardware for simulation applications ranging from communications to finance has stimulated growing interest in hardware-based random number generators. This has led to reevaluation of many of traditional random number generator (RNG) methods in light of the constraints on precision and data flow regularity that characterize typical hardware platforms. For example, in software the best methods for Gaussian number generation are rejection-acceptance methods such as the Ziggurat method [5]. These methods can offer extremely high quality random numbers, but produce output samples conditionally, meaning that while the average output rate is known, the time-local output rate varies. This can lead to complications in applications that require new random number samples at specific clock intervals [6]. Thus, hardware implementations typically target methods that produce outputs at deterministic intervals.

In the last few years, there has been a growing body of literature specifically addressing hardware RNGs, with most of the attention focused on Gaussian random numbers. For Gaussian random variables many researchers have employed the Box–Muller method [7], which transforms pairs of uniformly distributed variables into pairs of Gaussian distributed variables and produces outputs at a deterministic rate. One of the earliest hardware designs using the Box–Muller method is described by Boutillon *et al.* [8], who utilize function approximation followed by application of the central limit theorem to reduce the effects of the function approximation errors. The design in [8] generates random samples up to 4σ and the corresponding implementation on an Altera Flex 10K1000EQC240-1 field-programmable gate array (FPGA) produced (using FPGA technology available in 2002) 24.5 million samples per second. Xilinx [9] has released an intellectual property (IP) core and Fung *et al.* [10] implemented an application-specific integrated circuit (ASIC) chip based on the architecture by Boutillon *et al.* [8]. The former has a throughput of 245 million samples per second on a Xilinx Virtex-II XC2V1000-6 FPGA, whereas the latter has a throughput of 182 million samples per second on a 0.18-μm ASIC. Alimohammad *et al.* [11] have implemented a Box–Muller-based design on a Xilinx Virtex-II XC2V4000-6 FPGA. Their design has a throughput of 132 million Gaussian random samples per second up to 6.55σ. The Box–Muller method was also the basis

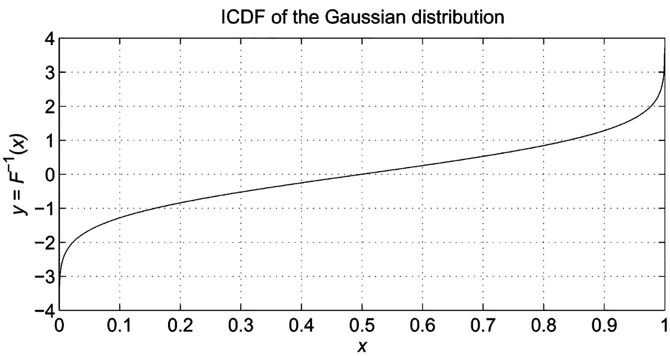


Fig. 1. Inverse cumulative distribution function of the Gaussian distribution.

for a recent design [12], which generates 16-bit samples up to 8.2σ, while guaranteeing accuracy to one ulp and achieving an output rate of 750 million samples per second on a Xilinx Virtex-4 XC4VLX100-12 FPGA.

There have been very few publications on hardware methods enabling the targeting of general (as opposed to Gaussian) distributions. One example is the work of Thomas and Luk [13], which presented an RNG design methodology for arbitrary distributions by combining multiple distributions to form a composite distribution. When applied to Gaussian random numbers, this approach is able to generate 193 million samples per second up to 5.1σ on a Xilinx Virtex-II XC2V4000-6 FPGA.

In this paper, we introduce a general RNG design generator that produces hardware designs for generating random numbers from arbitrary distributions using the inversion method [14]. The inversion method for generating nonuniform random numbers [15] utilizes the inverse cumulative distribution function (ICDF) to convert a sample x of a uniform random variable over $[0, 1)$ to a sample from the desired PDF through $y = F^{-1}(x)$. Thus, the challenge in ICDF hardware development lies in creating an efficient and accurate circuit design for evaluating the function $F^{-1}(x)$. For example, Fig. 1 shows the ICDF of the Gaussian distribution, where x is a uniform random number and y is a sample from the Gaussian distribution. Such ICDFs are generally nonlinear in the sense of having regions with high first or higher order derivatives. Hardware designs using the ICDF inversion technique have previously been implemented by McCollum *et al.* [16] and Chen *et al.* [17]. In [16], a Gaussian ICDF is implemented via linear interpolation with evenly-spaced data points. This implementation leads to a large table size of 262 kB. In [17], a precalculated ICDF inversion table using on-chip memory is utilized to transform uniform random numbers into nonuniform random numbers. This approach requires a 1-MB RAM for a 16-bit input/16-bit output lookup table.

The primary contributions of this paper are a rigorous and automated framework and the associated tools for generation of hardware RNGs for arbitrary distributions via the inversion method. Techniques including analytical error analysis, bit-width optimization, hierarchical segmentation, and piecewise polynomial approximation are used in combination to guarantee accuracy of one ulp while also offering area- or latency-optimized designs. The resulting hardware architectures are verified through FPGA implementation of designs for Gaussian, exponential, and log-normal distributions. The combination of generality, automation, and memory-efficient designs makes the method presented here suitable for a wide range of simulation environments and applications.

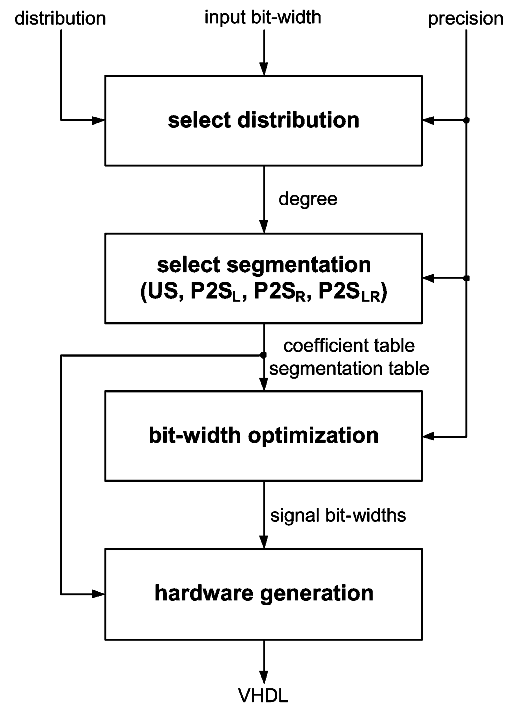


Fig. 2. Design flow of our approach.

The rest of this paper is organized as follows. Section II provides an overview of the proposed RNG design generator. Section III describes the application of hierarchical segmentation to approximate the ICDFs. Section IV presents the hardware architecture of the inversion-based RNG and its components. Section V covers the bit-width optimization technique used in the design generator. Section VI evaluates results of this paper and compares them against existing work. Concluding remarks are given in Section VII.

II. Overview

Fig. 2 shows the design flow and the design parameters of the RNG design generator is discussed here. The following design specifications are required for the design generator: target distribution, bit-width of the input x, and precision of the output y. Since the input bit-width determines how closely values of 0 and 1 can be approached, it influences the range of possible output random numbers for distributions with one-sided or two-sided tails of infinite length. The output precision decides the number of fractional bits used in representing the generated random sample.

The design generator divides the ICDF into segments for piecewise polynomial approximation using a nonuniform segmentation scheme. Chebyshev coefficients [18] are used for the polynomials. The generated coefficients and the segmentation information for a given ICDF are stored in ROM0 and ROM1, respectively (see Fig. 3).

The design generator also determines the minimum number of bits required for each signal in the datapath, while conforming

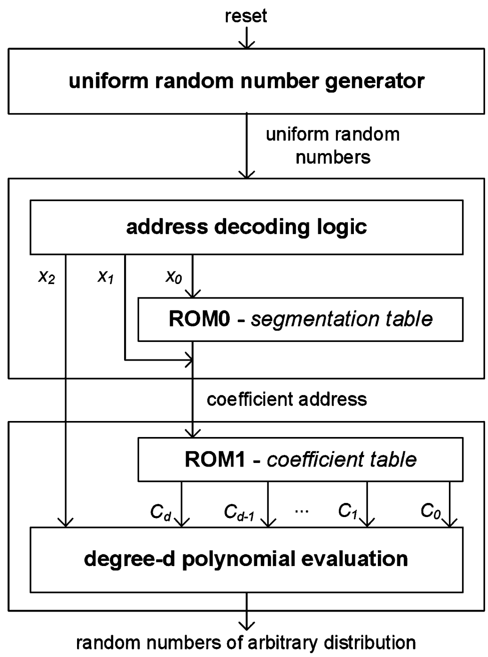


Fig. 3. Overview of the RNG architecture based on the inversion method.

to the random sample precision requirement from the design specifications. Finally, synthesizable VHDL code suitable for ASIC or FPGA realizations is produced using the ROM contents and the generated bit-widths. The entire design generation is conducted within MATLAB and is fully automated.

Fig. 3 gives an overview of the general RNG architecture based on the inversion method. When the $reset$ signal goes high, the uniform random number generator (URNG) is initialized to generate uniform random numbers from its predefined seeds. Using the URNG output, the address decoding logic extracts x_0, x_1, and x_2. x_0 is used for indexing the segmentation table ROM0, x_1 is used together with the ROM0 output for indexing the polynomial coefficients in ROM1, and x_2 is used in the polynomial evaluation.

The methods described here are demonstrated with the following three distributions:

$$f_1(x) = \left|F_1^{-1}\left(\frac{x}{2}|\mu,\sigma\right)\right|$$
$$F_1(y) = \frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^{y} e^{(-(t-\mu)^2/2\sigma^2)dt} \tag{1}$$
$$f_2(x) = F_2^{-1}(x|\mu)$$
$$F_2(y) = \int_0^y \frac{1}{\mu}e^{-(t/\mu)dt} \tag{2}$$
$$f_3(x) = F_3^{-1}(x|\mu,\sigma)$$
$$F_3(y) = \frac{1}{\sigma\sqrt{2\pi}}\int_0^y \frac{e^{\frac{-(\ln(t)-\mu)^2}{2\sigma^2}}}{t}dt \tag{3}$$

where 1) f_1 is the ICDF of the Gaussian distribution with mean $\mu = 0$ and standard deviation $\sigma = 1$; 2) f_2 is the ICDF of the exponential distribution with $\mu = 1$; and 3) f_3 is the ICDF of the log-normal distribution with $\mu = 0$ and $\sigma = 0.5$. As shown in Fig. 1, the Gaussian distribution is symmetric, in implementation the absolute value of the first half of the Gaussian ICDF $x = [0, 0.5)$ is approximated. For the reconstruction of the full distribution, a random bit is then used for the sign of the generated Gaussian sample. The exponential and log-normal distributions do not exhibit this symmetry property, and so are evaluated directly across the entire range of interest.

III. Hierarchical Segmentation of ICDFs

The most commonly used segmentation method is the uniform scheme, where all segment lengths are equal [19]–[22] and the segment count is typically limited to powers of two. The major difficulty of the proposed RNG is to approximate the ICDF of a given distribution. Although the uniform scheme leads to simple coefficient address computation, nonuniform segmentation enables segment lengths to be customized to the local function characteristics. We apply the hierarchical segmentation method (HSM) [23] to efficiently approximate ICDFs according to the behavior of the distributions.

HSM provides four basic segmentation schemes, denoted by *US*, $P2S_L$, $P2S_R$, and $P2S_{LR}$, respectively. In *US*, segments are uniformly sized. In $P2S_L$, the segment sizes increase by powers of two from the beginning of the input interval to the end of the interval, while in $P2S_R$ the segment sizes decrease by powers of two from the beginning to the end of the interval. In $P2S_{LR}$, segment sizes increase by powers of two until the midpoint of the interval and then decrease by powers of two until the end is reached. This method is hierarchical because the segmentation can be applied recursively: in the first pass, the entire interval is subdivided using one of the previous four schemes into smaller segments, then in the second pass, each segment can be further subdivided, again using any of the four schemes. During the second pass for the framework in this paper, the segmentation is fixed to *US*.

The core of the segmentation algorithm requires four parameters: the input interval, the polynomial degree d to be used for the piecewise polynomial approximation, and the desired maximum absolute error ϵ_{req} at the output. For each segment of the first pass (outer segmentation), the Chebyshev coefficients for the approximating polynomial are computed. If the Chebyshev approximation error $\epsilon_{\max}$ is too high, the number of segments of the second pass (inner segmentation) is incremented by successive powers of two until the $\epsilon_{\max}$ of all inner segments are less than or equal to the required error ϵ_{req}. This process is performed for all outer segments.

Let the bit-width of x be B_x. Using the two-level HSM segmentation, the input x, which has B_x bits, is divided into three partitions, x_0, x_1, and x_2. x_0 and x_1 are used to index the outer and inner segmentation, while x_2 is used for polynomial arithmetic.

For the first partition x_0, it is necessary to compute the segment address by detecting the number of leading zeros for segments beginning with a zero, and detecting the number of leading ones for segments beginning with a one. Consider the case when $B_x = 7$, the outer segmentation is $P2S_L$, $B_{x_0} = 4$, and $B_{x_1} = 1$. As illustrated in Fig. 4, it is possible to construct a maximum of five outer segments and five inner segments. B_{x_0} gives the number of bits used for indexing the segments in the first partition. It is determined by our design generation tool, which makes use of a linear search algorithm to calculate

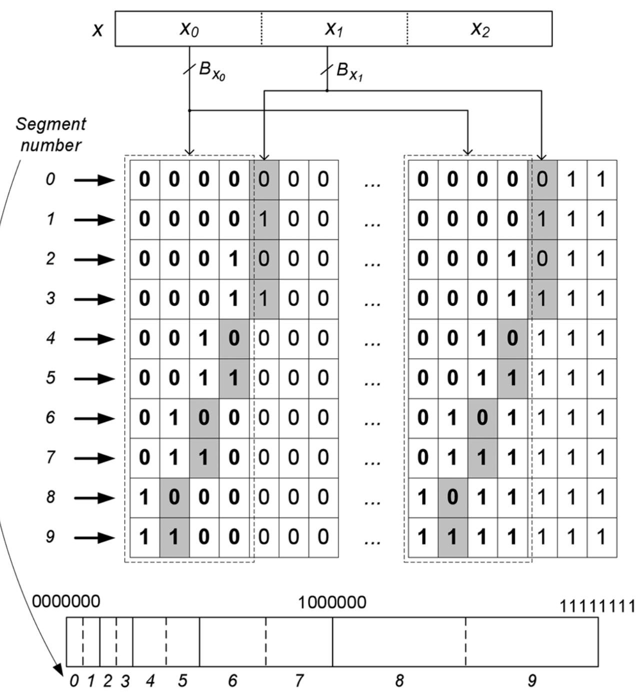


Fig. 4. Segment ranges in binary representation for $B_x = 7$, $P2S_L$ outer segmentation, $B_{x_0} = 4$, and $B_{x_1} = 1$. The four bits corresponding to x_0 are highlighted in bold. The bits to the left of the shadowed digit correspond to $\hat{x}_0$.

the minimum number of segments m for the coefficient table ROM1. Let $\hat{x}_0$ be the set of bits that remain constant (i.e., the bits left of the shadowed digit in Fig. 4) within a given segment. The next partition uses the adjacent B_{x_1} bits to the right of $\hat{x}_0$. The number of bits corresponding to the second level depends on the value of x_0, since x_0 determines the value of $B_{\hat{x}_0}$.

The absolute value of the derivative at the interval end points is used to drive the choice of the outer segmentation scheme. High derivatives at one or both ends trigger the use of $P2S_L$, $P2S_R$, or $P2S_{LR}$; in the case where both derivatives are small then uniform segmentation is used. $P2S_L$, $P2S_R$, and $P2S_{LR}$ are required for f_1, f_2, and f_3, respectively. Fig. 5 shows the resulting segmentations for degree-2 piecewise approximations with the error requirement fixed at 0.3×2^{-11} and the input fixed at 24 bits. A total of 80, 88, and 111 segments are required for f_1, f_2, and f_3, respectively. The HSM schemes offer an effective way to match the segment size according to the nonlinear regions of a function.

The proposed design generator produces two tables: ROM0 which is needed for ROM1 address computation and ROM1 which holds the polynomial coefficients for each segment. ROM0 stores the B_{x_1} and the offset corresponding to each outer segment. The offset is simply the number of rows in ROM1 prior to the row in ROM1 corresponding to the current outer segment. The hierarchical segmentation allows minimization of the number of segments for approximating highly nonlinear functions such as ICDFs considered here.

Table I shows a comparison of the number of segments for uniform and hierarchical segmentation for different error requirements for f_1. The HSM approach greatly reduces the number of segments due to its variable nature of the segment sizes. In Table II, the effect of changing the input bit-width on the number of segments is examined. With increasing input bit-width, the segment count increases slowly for the HSM scheme while it increases exponentially for the uniform scheme.

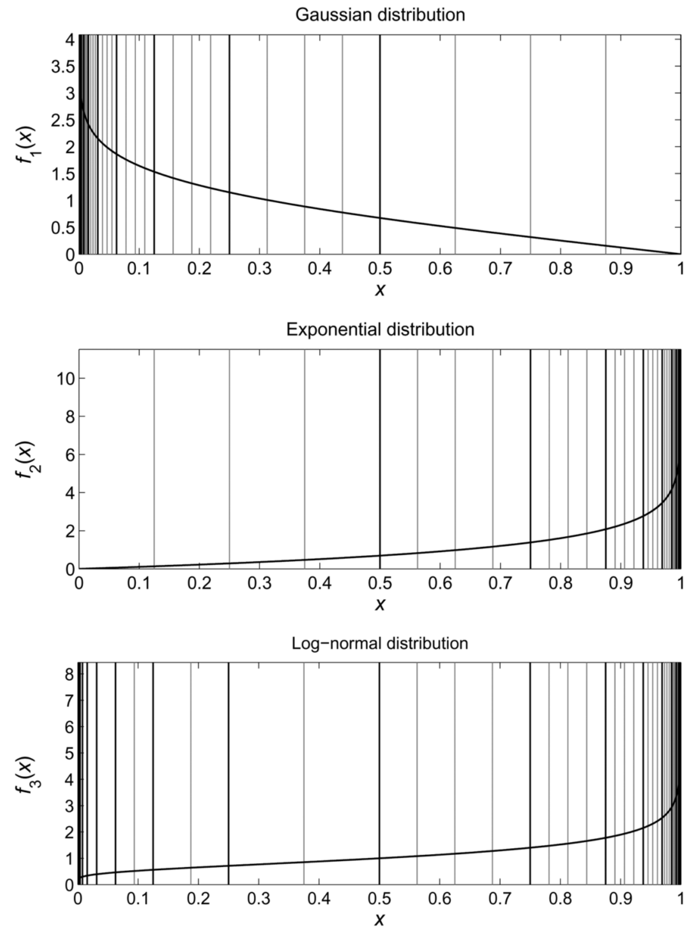


Fig. 5. Inversion plots for f_1 using the $P2S_L$ segmentation, f_2 using the $P2S_R$ segmentation, and f_3 using the $P2S_{LR}$ segmentation for the degree-2 piecewise polynomial approximations with the error requirement fixed at 0.3×2^{-11} and the input fixed at 24 bits. The black and grey vertical lines represent the boundaries for the outer segmentation and inner segmentation, respectively.

TABLE I
VARIATION OF THE NUMBER OF SEGMENTS WITH ERROR REQUIREMENTS FOR UNIFORM AND HIERARCHICAL SEGMENTATION OF THE FUNCTION f_1 WITH 16 BITS INPUT

error requirement	2^{-8}	2^{-10}	2^{-12}	2^{-14}	2^{-16}
uniform	32768	32768	32768	65536	65536
HSM	16	27	42	58	104

IV. INVERSION-BASED RNG ARCHITECTURE

Fig. 6 shows the architecture of the inversion-based RNG using the Gaussian case as an example. The architecture consists of a first stage containing a uniform RNG; a second stage containing an address decoding unit together with the segmentation table ROM0, a bit selection unit, and a $P2S$ unit; and a

TABLE II
VARIATION OF THE NUMBER OF SEGMENTS AGAINST DIFFERENT INPUT BIT-WIDTH WITH ERROR REQUIREMENT ACCURATE TO 2^{-8} FOR UNIFORM AND HIERARCHICAL SEGMENTATION OF THE FUNCTION f_1

input bit-width [bits]	10	12	14	16	18
uniform	512	2048	8192	32768	65536
HSM	10	12	14	16	17

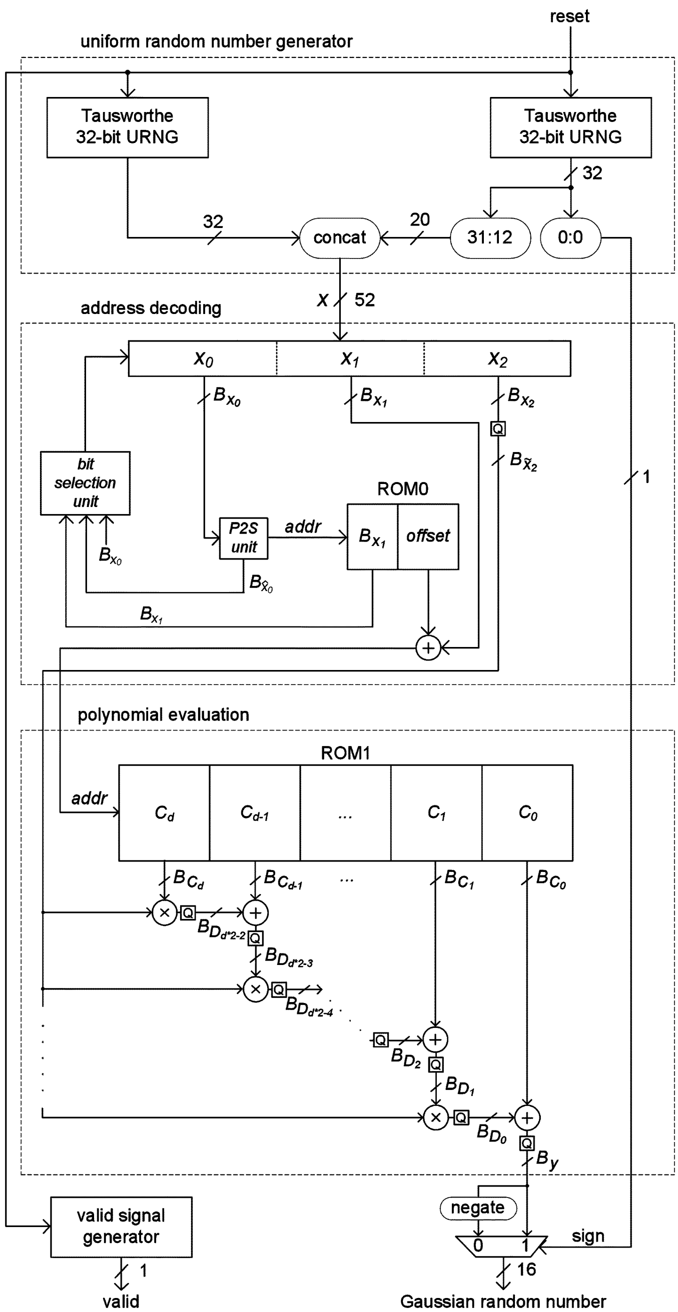


Fig. 6. Hardware architecture for generating Gaussian random numbers based on the inversion and the HSM method using degree-d approximation. ROM0 contains information on the hierarchical segmentation, while ROM1 contains the polynomial coefficients for each segment. The grey "Q" squares perform quantization at run-time.

third stage consisting of a piecewise polynomial evaluation unit incorporated with the coefficient table ROM1.

The first stage of the architecture uses the Tausworthe uniform-random number generator (URNG) [24], which is chosen

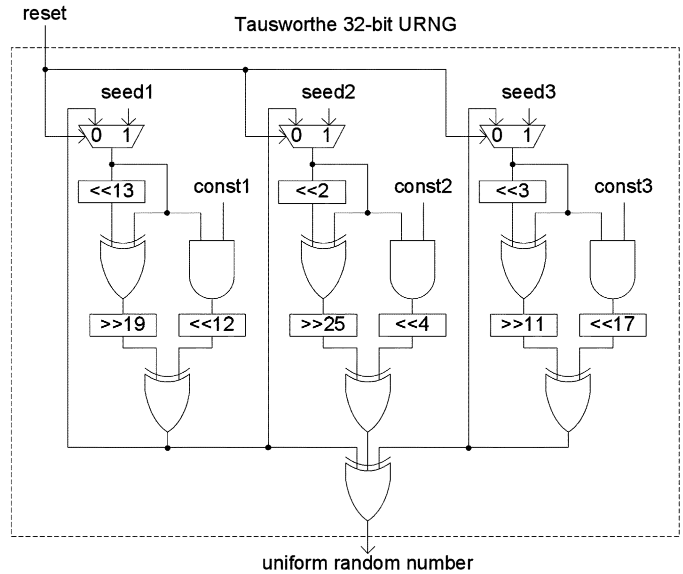


Fig. 7. Architecture of the Tausworthe URNG in Fig. 6. The output uniform random number is a 32-bit data.

TABLE III
MAXIMUM RANDOM NUMBER VALUE BY USING DIFFERENT INPUT BIT-WIDTHS

input bit-width	16	32	48
f_1	4.2	6.3	7.8
f_2	11.1	22.2	33.3
f_3	8.0	22.5	45.0

for generating the uniform random number x due to its superior properties relative to LFSRs. Note that the Tausworthe URNG is a stretching function that extends a short seed, and hence its outputs are technically pseudo-random. As shown in [6], Tausworthe URNGs provide superior randomness when evaluated using the Diehard random number test suite [25]. Three LFSR-based URNGs exist in each Tausworthe URNG in order to enhance the equi-distribution property of the generated uniform random number. It has a large periodicity of $2^{88} \approx 10^{25}$ which is sufficient for the purpose of this paper. As noted in Section I, one random bit from the URNG is used to select the sign of the final Gaussian random number which has 5 integer bits and 11 fractional bits. Fig. 7 shows the circuitry of the Tausworthe URNG component. The output from the first stage is an n-bit uniform random number x according to the input bit-width n.

The two specifications of this Gaussian random number generator (GRNG) are a periodicity of 10^{15} and 16-bit two's complement fixed-point random samples. This GRNG is adequate even for the most ambitious simulation applications such as the evaluation of low-density parity check codes in very low bit error rate [1]. For a population of 10^{15} Gaussian samples, up to 8.2σ needs to be represented. Since $x = 2^{-52}$ results in $y = 8.2$ for the Gaussian ICDF, 52 bits are allocated for x. Table III shows the maximum value of the generated random number by changing the input bit-widths. The top 52 bits and the last one bit from concatenating the two URNGs are extracted for the input and the sign control of the design.

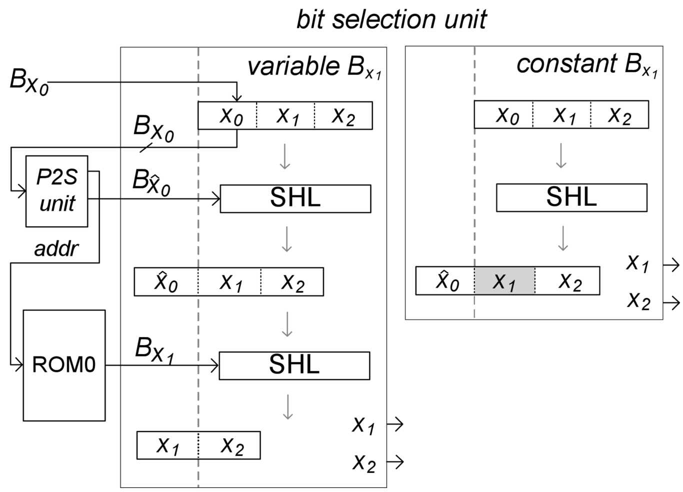


Fig. 8. Illustration of the bit selection unit in Fig. 6. The second barrel shifter is removed when the constant B_{x_1} design is used. SHL refers to a left barrel shifter.

In the second stage, the $P2S$ unit computes the outer segment address for a given x_0. The number of bits required to represent x_1 is determined by B_{x_1} in ROM0. These values are calculated by the design generator and are prestored in ROM0. The offset data represents the starting coefficient address of each outer segment. Adding this offset address with the value of x_1 enables locating the corresponding coefficient address in ROM1. Note that the size of ROM0 is negligible because its depth is limited by the number of outer segments.

The bit selection unit shown in Fig. 8 has two versions: one with variable B_{x_1} and the other with constant B_{x_1}. For the variable B_{x_1} design, the first barrel shifter is used to remove the leading $B_{\hat{x}_0}$ bits. The second left barrel shifter is used to separate the remaining bits into x_1 and x_2. For the constant B_{x_1} design, the left barrel shifter is used to remove the variable $B_{\hat{x}_0}$ bits, since only the B_{x_1} bits are used for x_1 and the remaining bits would represent x_2. Since all outer segments use the same number of inner segments, this simplification increases the total number of segments resulting in a larger ROM1 size. However, the address decoding unit complexity is reduced because the second barrel shifter is no longer needed.

In the third stage, the polynomial evaluation is performed using Horner's rule

$$y = ((C_d\tilde{x}_2 + C_{d-1})\tilde{x}_2 + \cdots)\tilde{x}_2 + C_0 \tag{4}$$

where $\tilde{x}_2$ is the input, d is the polynomial degree, and C_i are the polynomial coefficients. x_2 is used instead of x for the polynomial evaluation to reduce the size of the operators; this requires the coefficient transformation technique [12] and x_2 is further quantized to $\tilde{x}_2$. This provides the approximation of the first half of the Gaussian distribution. In order to obtain the complete Gaussian distribution, one uniform random bit is used to select between the output signal of stage 3 and its negated version.

V. Bit-Width Optimization

Bit-widths of signals are important parameters that designers can tweak to improve the quality of a design in terms of area, latency, and throughput. The goal is to use the minimal bit-widths to each signal, while respecting error constraints at the output. Two's complement fixed-point arithmetic is used. Given a signal x, its integer bit-width (IB) is denoted by IB_x and its fractional bit-width (*FB*) is denoted by FB_x, i.e., the total signal bit-width $B_x = IB_x + FB_x$. We adopt the MiniBit technique described in [26] optimized for polynomial-based function evaluation.

For *IB* determination, the local minima/maxima and the minimum/maximum input values of each signal are examined in order to compute the dynamic range. The local minima/maxima can be found by computing the roots of the derivative. Once the dynamic range has been found, the required *IB* can then be computed. In the proposed RNG generator, piecewise polynomial approximations are being used, where the polynomial evaluation circuit needs to be shared among different sets of coefficients. The *IB* for each signal is found for every segment and stored in a vector. Since the signal needs to be wide enough to avoid overflow for the data with the largest dynamic range, the largest *IB* in the vector is used.

FB determination begins by considering the three main error sources that exist when evaluating functions in digital arithmetic: 1) the inherent approximation error ϵ_∞; 2) quantization error ϵ_Q; and 3) the error of the final output rounding step, which can cause a maximum error of 1/2 ulp. In the results presented here, we allocate a maximum of 0.3 ulp for ϵ_∞ and the rest for ϵ_Q, which has been found to give a well balance between these two error sources. This explains why the error requirement has been set to 0.3 $\times 2^{-11}$ in Section III. Truncation can cause a maximum error of 2^{-FB} (1 ulp), while round-to-nearest can cause 2^{-FB-1} (1/2 ulp). To achieve faithful rounding where results are accurate to within one ulp, round-to-nearest must be performed at the output signal y which is required in this paper. For the other internal signals, truncation is used since it has a better delay and area characteristics over round-to-nearest.

The addition and multiplication error expressions [26] are applied to every operator and a condition to achieve faithful rounding is generated for the output signal. Note that the error at a signal x is denoted by ϵ_x. For the polynomial evaluation unit (stage 3 in Fig. 6), the input x_2 to the polynomial evaluation is assumed to have no error, i.e., $\epsilon_{x_2} = 0$. Since B_x equals 52 in this example, B_{x_2} can be potentially large, which can lead to increase burden on the arithmetic operators. To overcome this problem, x_2 is quantized to $\tilde{x}_2$ for the polynomial evaluation to reduce the size of the operators. We describe the *FB* analysis using a degree-1 approximation case: $y = C_1 \times \tilde{x}_2 + C_0$

$$D_0 = C_1 \times \tilde{x}_2 \tag{5}$$
$$y = D_0 + C_0. \tag{6}$$

The error ϵ_{D_0} at the signal D_0 is given by

$$\epsilon_{D_0} = C_1\epsilon_{\tilde{x}_2} + \tilde{x}_2\epsilon_{C_1} + \epsilon_{C_1}\epsilon_{\tilde{x}_2} + 2^{-FB_{D_0}} \tag{7}$$

where $2^{-FB_{D_0}}$ is the quantization error at D_0. The quantization error $\epsilon_{\tilde{x}_2}$ is $2^{-FB_{\tilde{x}_2}}$. The error ϵ_y at the output y is given by

$$\epsilon_y = \epsilon_{D_0} + \epsilon_{C_0} + 2^{-FB_y-1} + \epsilon_\infty. \tag{8}$$

For faithful rounding, the maximum output error $\max(\epsilon_y)$ needs to be less than or equal to 1 ulp, i.e.,

$$2^{-FB_y} \geq \max(\epsilon_y). \tag{9}$$

Using (7)–(9) gives

$$\begin{aligned} 2^{-FB_y} \geq\, & C_1 \times 2^{-FB_{\tilde{x}_2}} + 2^{-FB_{D_0}} + 2^{-FB_{C_0}-1} + 2^{-FB_{C_1}-1} \\ & \times \left(\tilde{x}_2 + 2^{-FB_{\tilde{x}_2}}\right) + 2^{-FB_y-1} + \epsilon_\infty \Rightarrow 2^{-FB_y} \\ & \geq \left(C_1 \times 2^{-FB_{\tilde{x}_2}} + 2^{-FB_{D_0}} + 2^{-FB_{C_0}-1}\right. \\ & \left. + 2^{-FB_{C_1}-1} \times \left(\tilde{x}_2 + 2^{-FB_{\tilde{x}_2}}\right) + \epsilon_\infty\right) \times 2. \end{aligned} \tag{10}$$

Similarly for degree-2 polynomial, we get

$$\begin{aligned} 2^{-FB_y} \geq & \left(2^{-FB_{C_0}-1} + 2^{-FB_{C_1}-1} \times \left(\tilde{x}_2 + 2^{-FB_{\tilde{x}_2}}\right)\right. \\ & + 2^{-FB_{C_2}} \times \left(\tilde{x}_2 + 2^{-FB_{\tilde{x}_2}}\right)^2 + 2^{-FB_{D_0}} \\ & + 2^{-FB_{D_1}} \times \left(\tilde{x}_2 + 2^{-FB_{\tilde{x}_2}}\right) + 2^{-FB_{D_2}} \\ & \times \left(\tilde{x}_2 + 2^{-FB_{\tilde{x}_2}}\right)^2 + C_2 \times 2^{-FB_{\tilde{x}_2}} \\ & \left. \times \left(\tilde{x}_2 + 2^{-FB_{\tilde{x}_2}}\right) + D_1 \times 2^{-FB_{\tilde{x}_2}} + \epsilon_\infty\right) \times 2. \end{aligned} \tag{11}$$

For C_1, C_2, and D_1 in (10) and (11), their absolute maximum values are used.

Equations (10) and (11) are optimization problems, where the goal is to find the *FBs* that minimize a given cost function while satisfying the previous inequalities [26]. To solve this optimization problem, adaptive simulated annealing (ASA) [27] is used with the circuit area of the operators and tables supplied as the cost function.

Table IV shows the signal bit-widths found by ASA when evaluating f_1 accurate to 11 fractional bits with degree-1 and degree-2 approximations. This table also shows the number of segments and the size of ROM1 using variable B_{x_1} and using constant B_{x_1}. By quantizing the input x_2 to $\tilde{x}_2$, significant bit-width reduction can be obtained. For instance, for the degree-2 design using constant B_{x_1}, x_2 is quantized to $\tilde{x}_2$ and the number of bits is reduced from 48 to 20. B_{x_2} is 48 bits because of the minimum sum of $B_{\hat{x}_0}$ and B_{x_1} is 4 bits. This quantization step can potentially save hardware, since for example in this design, the output y is allowed to be significantly less precise than the input x_2.

VI. Evaluation and Results

The implementation results presented in this section are realized on a Xilinx Virtex-4 FPGA. The three major resources inside the FPGAs are: 1) configurable blocks known as slices which have two four-input look-up tables, multiplexers, carry logic, and two registers; 2) DSP-blocks which can perform an 18-bit by 18-bit multiplication followed by a 48-bit addition; and 3) block-RAMs which can store a maximum of 18 kb of data, using specific data bits and memory depths.

TABLE IV
Number of Segments and Bit-Widths for Evaluating the Gaussian ICDF Function f_1 Accurate to 11 Fractional Bits With Quantized Input $\tilde{x}_2$. The Bit-Widths in the Brackets Indicate the *IBs* and the *FBs*

design	degree-1		degree-2	
B_{x_1} [bits]	variable	constant	variable	constant
segments	620	832	144	208
ROM1 [bits]	19220	25792	6912	9360
B_{C_2}	-	-	12 (-3,15)	9 (-5,14)
B_{C_1}	11 (-3,14)	10 (-3,13)	15 (-1,16)	14 (-1,15)
B_{C_0}	20 (5,15)	21 (5,16)	21 (5,16)	21 (5,16)
B_{D_2}	-	-	16 (-3,19)	14 (-5,19)
B_{D_1}	-	-	15 (-1,16)	14 (-1,15)
B_{D_0}	19 (-3,22)	19 (-3,22)	22 (-1,23)	22 (-1,23)
B_{x_2}	22 (0,22)	23 (0,23)	23 (0,23)	20 (0,20)

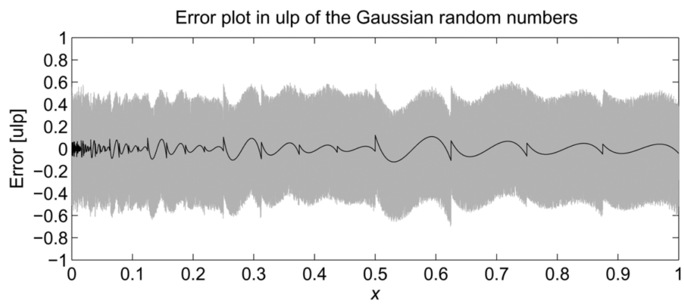


Fig. 9. Error plot in ulp using 2^{16} randomly selected samples for degree-2 approximation to function f_1 accurate to 11 fractional bits and 5 integer bits with 53 input bits and the bit-widths from Table IV incorporated. The black curve indicates the inherent approximation error ϵ_∞, while the grey curve indicates the error with finite precision effects. Over 95% of the outputs are exactly rounded: the remaining 5% are faithfully rounded.

In examining the quality of the samples, we consider the differences between the samples produced by the hardware and the corresponding samples that would be produced using an ICDF approximation with floating point accuracy. This is motivated by the knowledge that the underlying inversion method delivers perfect samples assuming infinite precision. Thus, the extent to which the output samples deviate from this ideal is directly determined by the accuracy of the hardware evaluation. Fig. 9 shows an ulp error plot of 2^{16} randomly selected samples for degree-2 approximation. Results show that 95% of the samples are exactly rounded (i.e., accurate to 1/2 ulp). Fig. 10 shows the PDF of the generated random numbers for each of the three distributions for a population of ten million. Fig. 11 shows the PDF between 7σ and 8.2σ for a population of one million for the Gaussian distribution. In both cases, the generated random numbers closely follow the true PDF of the associated distribution.

For the results shown in Figs. 12–17, the RNGs are implemented combinatorially using slices only with constant B_{x_1} and synthesized using Synplicity Synplify Pro 8.4. Xilinx ISE

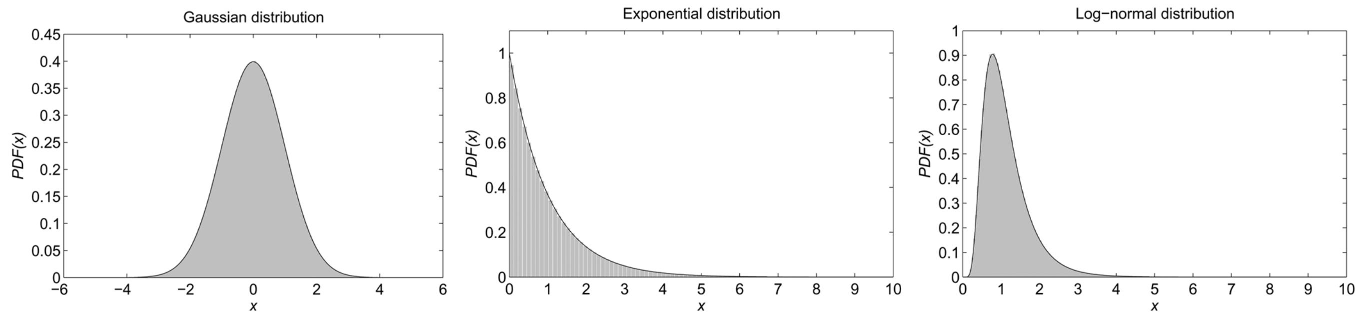


Fig. 10. PDFs of the generated random numbers from the proposed architecture for a population of ten million samples for three distributions. The black solid line indicates the ideal PDF of each distribution.

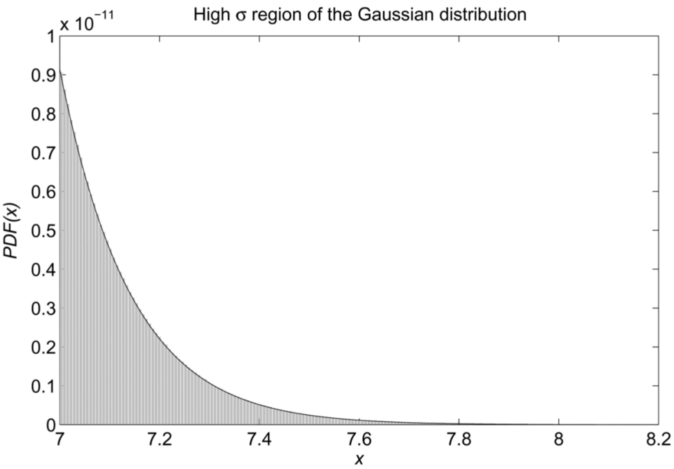


Fig. 11. PDF of the generated random numbers from the proposed architecture for a population of one million samples between 7σ and 8.2σ. The black solid line indicates the ideal Gaussian PDF.

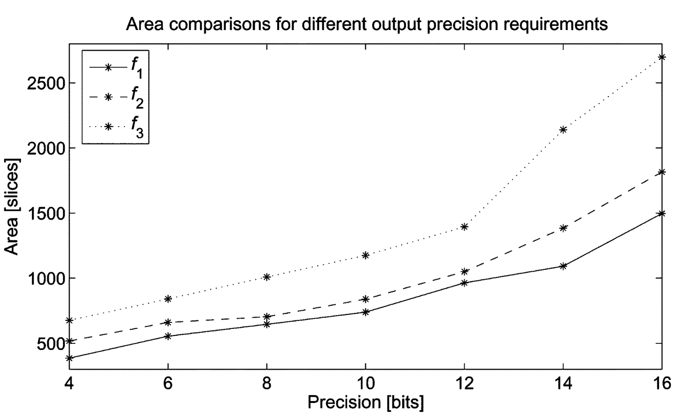


Fig. 12. Area comparisons for variable output precisions and the input is fixed at 24 bits.

8.1.02i is used for place-and-route with maximum effort. Note that precision refers to the number of fractional bits.

Figs. 12 and 13 show the area and latency variations using degree-2 approximations with the input fixed at 24 bits. The area and latency increase with precision due to the increasing ROM1 and operators in the polynomial evaluation unit. f_3 is the slowest and uses the most area due to its larger number of segment requirement and more complex address decoding ($P2S_{LR}$, rather than $P2S_L$ or $P2S_R$).

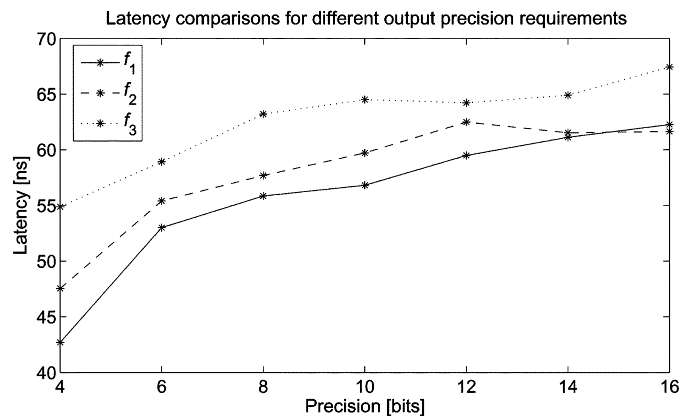


Fig. 13. Latency comparisons for variable output precisions and the input is fixed at 24 bits.

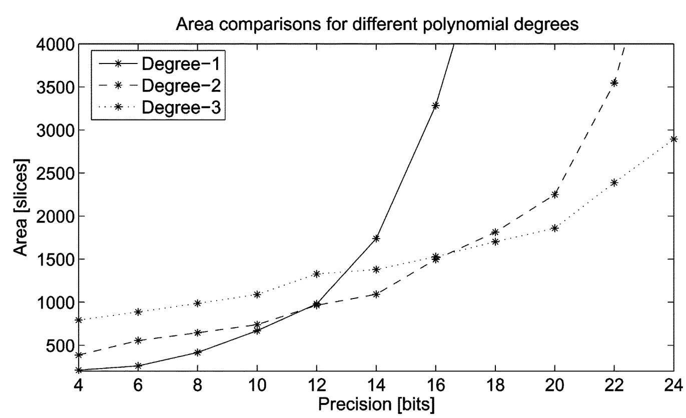


Fig. 14. Area comparisons of different degree approximations to f_1.

Figs. 14 and 15 show area and latency comparisons of different polynomial degrees for f_1 with 24 bits input. For precisions below 12 bits, degree-1 is the most area-efficient, while precisions between 12 and 16 bits and above 16 bits, degree-2 and degree-3 are the most area-efficient.

Figs. 16 and 17 examine the area and latency variations with different input bit-widths and precision fixed at 11 bits. The general trend for all three distributions is increasing because we need a larger URNG in the input and thus more bits for the address decoding circuit and the polynomial evaluation unit.

To demonstrate pipelined high-throughput designs, GRNGs are implemented using all three types of FPGA resources (slices,

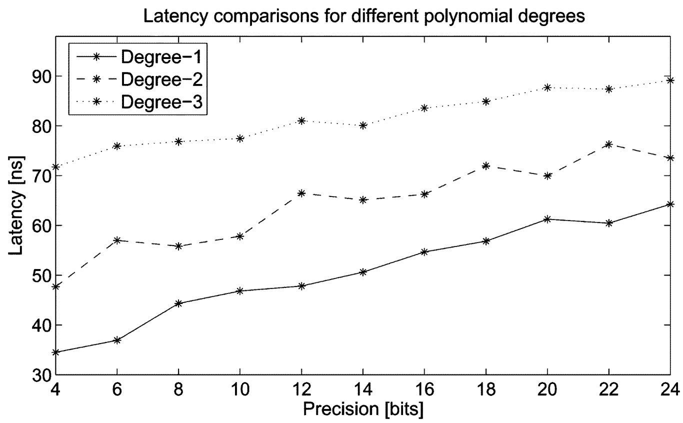


Fig. 15. Latency comparisons of different degree approximations to f_1.

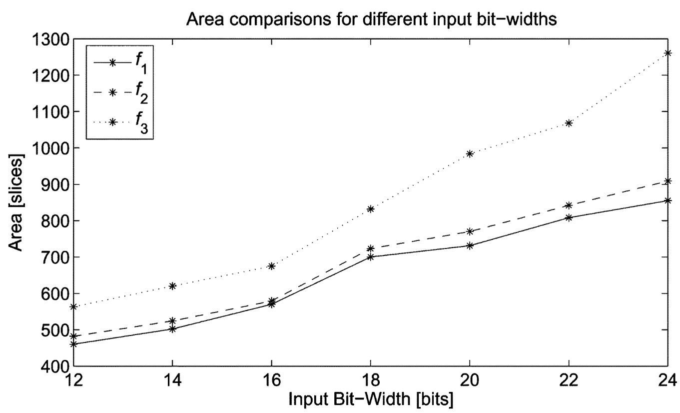


Fig. 16. Area comparisons for variable input bit-widths and the precision is fixed at 11 bits.

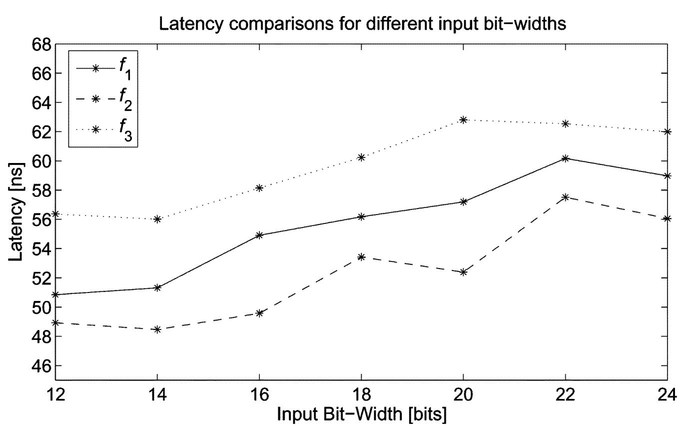


Fig. 17. Latency comparisons for variable input bit-widths and the precision is fixed at 11 bits.

DSP-blocks, and block-RAMs). Table V inspects the resource allocation of the various parts of the GRNG using degree-2 approximation. Results show that the $P2S$ address decoding circuit consumes the largest portion of the hardware resources. The three major components in the address decoding circuit are the leading zero detector (LZD), the leading one detector (LOD), and the barrel shifters. For the LZD and LOD, the method proposed by Oklobdzija [28] is used. For the logical barrel shifters, the method proposed by Pillmeier *et al.* [29] is used.

TABLE V
HARDWARE RESOURCE USAGE OF THE PROPOSED GRNG FOR DEGREE-1 APPROXIMATIONS TO f_1 USING VARIABLE AND CONSTANT B_{x_1}

component	slices	block-RAMs	DSP-blocks
	variable B_{x_1} - degree-1		
Tausworthe URNG	141	0	0
address decoding	268	0	0
polynomial evaluation	134	2	2
total	543	2	2
	constant B_{x_1} - degree-1		
Tausworthe URNG	141	0	0
address decoding	221	0	0
polynomial evaluation	125	2	2
total	487	2	2

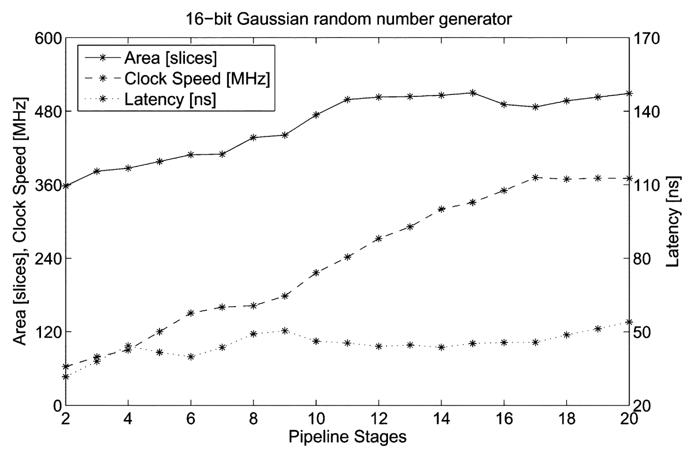


Fig. 18. Area, clock speed (i.e., throughput), and latency variation with number of pipeline stages for 16-bit GRNG with 53-bit input and 11-bit output precision using degree-1 approximation with constant B_{x_1} and quantized x_2. Block-RAMs and DSP-blocks available on the Virtex-4 device are utilized.

Fig. 18 shows the area, clock speed (i.e., throughput), and latency variation with the number of pipeline stages. The design uses degree-1 approximation with constant B_{x_1} and 11 fractional bits. We insert pipeline registers into the design according to the post place-and-route timing analysis. These additional registers breakdown the critical path improving the clock speed, but also induce an area penalty. As Fig. 18 shows, with 17 pipeline stages, the design reaches a maximum clock speed of 371 MHz, which is limited by the critical path between the output registers of the block-RAMs and the input of the DSP-blocks inside the Xilinx Virtex-4 FPGA. The addition of further pipeline stages leads to diminishing returns in terms of the performance/area ratio. Fig. 19 shows the distribution of pipelining registers for the design using degree-1 approximation and constant B_{x_1}.

Table VI compares the proposed GRNG against a recent implementation [12], which is the fastest GRNG reported in literature. Both designs generate faithfully rounded 16-bit random numbers (5 integer bits and 11 fractional bits) up to 8.2σ. This table shows that the proposed design using degree-1 approximation with constant B_{x_1} has the best performance/area ratio. Moreover, it has greatly reduced the hardware resource usage

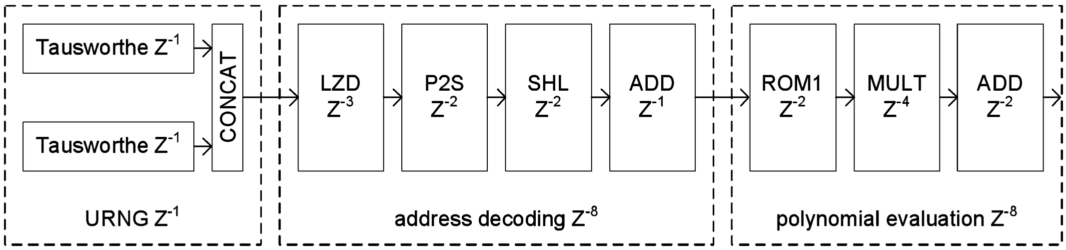


Fig. 19. Pipeline register distribution of the GRNG design using degree-1 approximation with constant B_{x_1}. SHL refers to a left barrel shifter and Z^{-1} refers to one pipeline stage.

TABLE VI
COMPARISONS OF DIFFERENT HARDWARE GAUSSIAN RANDOM NUMBER GENERATORS IMPLEMENTED ON A XILINX VIRTEX-II XC2V4000-6 (V2) FPGA AND A XILINX VIRTEX-4 XC4VLX100-12 (V4) FPGA

design	Lee [12]		proposed [variable B_{x_1}]				proposed [constant B_{x_1}]			
method	Box-Muller		Inversion [degree-1]		Inversion [degree-2]		Inversion [degree-1]		Inversion [degree-2]	
device	V2	V4	V2	V4	V2	V4	V2	V4	V2	V4
slices	1528	1452	548	543	585	579	502	487	542	523
block-RAMs	3	3	2	2	1	1	2	2	1	1
DSP-blocks	12	12	2	2	4	4	2	2	4	4
clock speed [MHz]	233	375	232	371	231	370	232	371	232	367
samples / clock cycle	2	2	1	1	1	1	1	1	1	1
million samples / sec	466	750	232	371	231	370	232	371	232	367
throughput / slice	0.305	0.518	0.423	0.683	0.395	0.639	0.462	0.762	0.428	0.702

TABLE VII
HARDWARE IMPLEMENTATION RESULTS OF THE GRNG USING DEGREE-1 APPROXIMATION WITH CONSTANT B_{x_1} USING DIFFERENT TYPES OF FPGA RESOURCES [BLOCK-RAMS (BRAMS) AND DSP-BLOCKS (DSPS)] ON A XILINX VIRTEX-4 XC4VLX100-12 FPGA

FPGA resources used	slices	BRAMs	DSPs	speed [MHz]
slices + BRAMs + DSPs	487	2	2	371
slices + BRAMs	611	2	-	370
slices + DSPs	1509	-	2	222
slices	1628	-	-	220

of block-RAMs and DSP-blocks. The reduction in hardware resources is partially due to the fact that a single function evaluation is required for the inversion method, whereas multiple function evaluations are needed in [12].

Table VII explores the area and speed tradeoffs of designs using different types of hardware resources. For instance, a lookup table can be implemented using block-RAM or distributed-RAM based on slices. The design using only slices requires more than three times the number of slices than the GRNG design utilizing all three FPGA resources. Also, the area and speed penalty of using slices to implement tables instead of block-RAMs is particularly high. Hence, dedicated FPGA resources such as block-RAMs and DSP-blocks should be used for area-efficient high-performance designs.

VII. CONCLUSION

We have presented an automated methodology for producing hardware-based nonuniform RNG designs using the inversion method. The designs are capable of generating random numbers from arbitrary distributions provided that the ICDFs is known. The ICDF is approximated via piecewise polynomial approximation and hierarchical segmentation techniques. This enables random samples corresponding to arbitrary distributions to be produced by providing the approximation circuit with samples from a uniform random number generator. The approach is demonstrated using three distributions: Gaussian, exponential, and log-normal using fixed-point arithmetic, and offers designers a range of tradeoffs involving latency, area, and precision.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers and D. Thomas, L. Howes, and T. Todman for their useful remarks and assistance.

REFERENCES

[1] L. Sun, H. Song, Z. Keirn, and B. V. Kumar, "Field programmable gate array (FPGA) for iterative code evaluation," *IEEE Trans. Magn.*, vol. 42, no. 2, pp. 226–231, Feb. 2006.

[2] J. Lawrence, S. Rusinkiewicz, and R. Ramamoorthi, "Efficient BRDF importance sampling using a factored representation," *ACM Trans. Graph*, vol. 23, no. 3, pp. 496–505, 2004.

[3] A. Brace, D. Gatarek, and M. Musiela, "The market model of interest rate dynamics," *Math. Finance*, vol. 7, no. 2, pp. 127–155, Apr. 1997.

[4] D. Knuth, *Seminumerical Algorithms*, ser. The Art of Computer Programming, 3rd ed. Reading, MA: Addison-Wesley, 1997, vol. 2.

[5] G. Marsaglia and W. Tsang, "The Ziggurat method for generating random variables," *J. Statist. Softw.*, vol. 5, no. 8, pp. 1–7, 2000.

[6] G. Zhang, P. Leong, D. Lee, J. Villasenor, R. Cheung, and W. Luk, "Ziggurat-based hardware Gaussian random number generator," in *Proc. IEEE Int. Conf. Field-Program. Logic Appl.*, 2005, pp. 275–280.

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

[8] E. Boutillon, J. Danger, and A. Gazel, "Design of high speed AWGN communication channel emulator," *Analog Integr. Circuits Signal Process.*, vol. 34, no. 2, pp. 133–142, 2003.

[9] Xilinx Inc., San Jose, CA, "Additive white Gaussian noise (AWGN) Core v1.0," (2002). [Online]. Available: http://www.xilinx.com

[10] E. Fung, K. Leung, N. Parimi, M. Purnaprajna, and V. Gaudet, “ASIC implementation of a high speed WGNG for communication channel emulation,” in *Proc. IEEE Workshop Signal Process. Syst.*, 2004, pp. 304–409.

[11] A. Alimohammad, B. Cockburn, and C. Schlegel, “An iterative hardware gaussian noise generator,” in *Proc. IEEE Pacific Rim Conf. Commun., Comput. Signal Process.*, 2005, pp. 649–652.

[12] D. Lee, J. Villasenor, W. Luk, and P. 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.

[13] D. Thomas and W. Luk, “Efficient hardware generation of random variates with arbitrary distributions,” in *Proc. IEEE Int. Symp. Field-Program. Custom Comput. Mach.*, 2006, pp. 57–66.

[14] P. Bratley, B. Fox, and E. Schrage, *A Guide to Simulation*. New York: Springer-Verlag, 1983.

[15] L. Devroye, *Non-Uniform Random Variate Generation*. New York: Springer-Verlag, 1986.

[16] J. McCollum, J. Lancaster, D. Bouldin, and G. Peterson, “Hardware acceleration of pseudo-random number generation for simulation applications,” in *Proc. IEEE Southeastern Symp. Syst. Theory*, 2003, pp. 299–303.

[17] J. Chen, J. Moon, and K. Bazargan, “Reconfigurable readback-signal generator based on a field-programmable gate array,” *IEEE Trans. Magn.*, vol. 40, no. 3, pp. 1744–1750, May 2004.

[18] J. Muller, *Elementary Functions: Algorithms and Implementation*. Berlin, Germany: Birkhauser Verlag AG, 1997.

[19] J. E. Stine and M. J. Schulte, “The symmetric table addition method for accurate function approximation,” *J. VLSI Signal Process.*, vol. 32, no. 2, pp. 167–177, Jun. 1999.

[20] F. d. Dinechin and A. Tisserand, “Multipartite table methods,” *IEEE Trans. Comput.*, vol. 54, no. 3, pp. 319–330, Mar. 2005.

[21] J. Piñeiro, S. Oberman, J. Muller, and J. Bruguera, “High-speed function approximation using a minimax quadratic interpolator,” *IEEE Trans. Comput.*, vol. 54, no. 3, pp. 304–318, Mar. 2005.

[22] D. Lee, A. A. Gaffar, O. Mencer, and W. Luk, “Optimizing hardware function evaluation,” *IEEE Trans. Comput.*, vol. 54, no. 12, pp. 1520–1531, Dec. 2005.

[23] D. Lee, W. Luk, J. Villasenor, and P. Cheung, “Hierarchical segmentation schemes for function evaluation,” in *Proc. IEEE Int. Conf. Field-Program. Technol.*, 2003, pp. 92–99.

[24] P. L’Ecuyer, “Maximally equidistributed combined tausworthe generators,” *Math. Computation*, vol. 65, no. 213, pp. 203–213, Jan. 1996.

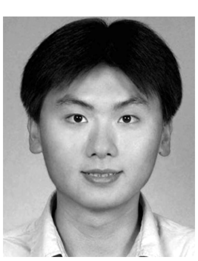
[25] G. Marsaglia, “Diehard: A battary of tests of randomness,” (1997). [Online]. Available: http://www.stat.fsu.edu/pub/diehard/

[26] D. Lee, A. A. Gaffar, R. Cheung, O. Mencer, W. Luk, and G. Constantides, “Accuracy guaranteed bit-width optimization,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 25, no. 10, pp. 1990–2000, Oct. 2006.

[27] L. Ingber, “Adaptive simulated annealing (ASA) 25.15,” (2004). [Online]. Available: http://www.ingber.com/#ASA

[28] V. Oklobdzija, “An algorithmic and novel design of a leading zero detector circuit: Comparison with logic synthesis,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 2, no. 1, pp. 124–128, Mar. 1994.

[29] M. Pillmeier, M. Schulte, and E. G. Walters, III, “Design alternatives for barrel shifters,” in *Proc. SPIE Adv. Signal Process. Algorithms, Arch., Implementations*, 2002, pp. 436–447.



Ray C. C. Cheung (S’01) received the B.Eng. and M.Phil. degrees in computer engineering and computer science and engineering from The Chinese University of Hong Kong (CUHK), Hong Kong, in 1999 and 2001, respectively. He is currently pursuing the Ph.D. degree in computing from the Imperial College London, London, U.K.

From January 2002 to December 2003, he was an Instructor with the Department of Computer Science and Engineering, CUHK. He visited Stanford University, Stanford, CA, and the University of California, Los Angeles, in 2005 and 2006 as a Visiting Scholar. His current research interests are computer arithmetic hardware designs and design exploration of system-on-chip (SoC) designs and embedded systems. He is a student member of the ACM and is the newsletter and web editor of the SIGDA U.K. chapter.

Dong-U Lee (M’05) received the B.Eng. degree in information systems engineering and the Ph.D. degree in computing from Imperial College London, London, U.K., in 2001 and 2004, respectively.

He is currently a Postdoctoral Researcher with the Electrical Engineering Department, University of California, Los Angeles (UCLA), where he is working on algorithms and implementations for deep-space communications with the Jet Propulsion Laboratory, and hardware designs of mathematical function evaluation units. His research interests include computer arithmetic, communications, design automation, reconfigurable computing, and video image processing.

Wayne Luk (SM’06) received the M.A., M.Sc., and D.Phil. degrees in engineering and computer science from the University of Oxford, Oxford, London, U.K.

He is currently a Professor of computer engineering with the Department of Computing, Imperial College London, and a Visiting Professor with Stanford University, Stanford, CA, and Queen’s University Belfast, Belfast, Ireland. His research interests include theory and practice of customizing hardware and software for specific application domains, such as graphics and image processing, multimedia, and communications. Much of his current work involves high-level compilation techniques and tools for parallel computers and embedded systems, particularly those containing reconfigurable devices such as field-programmable gate arrays.

John D. Villasenor (SM’98) received the B.S. degree from the University of Virginia, Charlottesville, in 1985, the M.S. and the Ph.D. degrees from Stanford University, Stanford, CA, in 1986 and 1989, respectively, all in electrical engineering.

He joined the Electrical Engineering Department, University of California, Los Angeles (UCLA), in 1992, where he is currently a Professor. He served as Vice Chair of the Department from 1996 to 2002. From 1990 to 1992, he was with the Radar Science and Engineering Section, Jet Propulsion Laboratory, Pasadena, CA, where he developed methods for imaging the earth from space. At UCLA, his research efforts lie in communications, computing, imaging and video compression, and networking.