A Hardware Gaussian Noise Generator Using the Box-Muller Method and Its Error Analysis

Dong-U Lee, *Member*, *IEEE*, John D. Villasenor, *Senior Member*, *IEEE*, Wayne Luk, *Member*, *IEEE*, and Philip H.W. Leong, *Senior Member*, *IEEE*

Abstract—We present a hardware Gaussian noise generator based on the Box-Muller method that provides highly accurate noise samples. The noise generator can be used as a key component in a hardware-based simulation system, such as for exploring channel code behavior at very low bit error rates, as low as 10^{-12} to 10^{-13}. The main novelties of this work are accurate analytical error analysis and bit-width optimization for the elementary functions involved in the Box-Muller method. Two 16-bit noise samples are generated every clock cycle and, due to the accurate error analysis, every sample is analytically guaranteed to be accurate to one unit in the last place. An implementation on a Xilinx Virtex-4 XC4VLX100-12 FPGA occupies 1,452 slices, three block RAMs, and 12 DSP slices, and is capable of generating 750 million samples per second at a clock speed of 375 MHz. The performance can be improved by exploiting concurrent execution: 37 parallel instances of the noise generator at 95 MHz on a Xilinx Virtex-II Pro XC2VP100-7 FPGA generate seven billion samples per second and can run over 200 times faster than the output produced by software running on an Intel Pentium-4 3 GHz PC. The noise generator is currently being used at the Jet Propulsion Laboratory, NASA to evaluate the performance of low-density parity-check codes for deep-space communications.

Index Terms—Algorithms implemented in hardware, computer arithmetic, error analysis, elementary function approximation, field programmable gate arrays, minimax approximation and algorithms, optimization, random number generation, simulation.

1 Introduction

Due to recent advances in field-programmable technology, hardware-based simulations are getting increasing attention due to their huge performance advantages over traditional software-based methods. Naive hardware implementations can be slow and can generate misleading results. Hence, care must be taken when mapping algorithms into hardware. In particular, the resulting hardware design should meet the performance targets while making efficient use of the available resources and properly managing errors due to, for instance, finite precision effects.

The availability of normally distributed random samples is essential in a large number of computationally intensive modeling and simulation applications, including channel code evaluation [1], molecular dynamics simulation [2], and financial modeling [3]. Our work is originally motivated by ongoing advances in communications systems involving channel codes. Notably, turbo codes [4] and low-density parity-check (LDPC) codes [1] are currently the focus of intensive research in the coding community due to their ability to approach the Shannon bound very closely. Software-based simulations can take several days to several weeks when the behavior at very low bit error rates (BERs) of such codes is being examined. Hardware-based simulations equipped with a fast and accurate noise generator offer the potential to speed up simulation by several orders of magnitude. Transferring software-generated noise samples to the hardware device is highly inefficient and can be a performance bottleneck, hence it is desirable to have the noise generator on the hardware device itself.

For simulations involving large numbers of samples, the quality of the noise samples plays a key factor. Deviations from the ideal Gaussian probability density function (PDF) can degrade simulation results and lead to incorrect conclusions. Hence, we believe the presence of a rigorously designed and characterized hardware Gaussian noise generator is crucial. Attention needs to be paid to the samples that lie at the tails of the Gaussian PDF, i.e., samples that lie multiples of σ (standard deviations) away. These samples are rare in a relative sense, but they are important because they can cause events of high interest.

The principal contribution of this paper is a hardware Gaussian noise generator based on the Box-Muller method and the error analysis of its elementary functions. It generates 16-bit noise samples accurate to one unit in the last place (ulp) up to 8.2σ, which models the true Gaussian PDF accurately for a simulation size of over 10^{15} samples. Generally, when evaluating channel codes, one needs 100 to 1,000 bits in error to draw conclusions on a simulation with enough confidence. Hence, with 10^{15} samples, one can examine channel code behavior for bit error rates as low as 10^{-12} to 10^{-13}. The noise generator is relatively small, while producing 750 million samples per second at a clock speed

- *D. Lee and J.D. Villasenor are with the Electrical Engineering Department, University of California, Los Angeles, 420 Westwood Blvd., Los Angeles, CA 90095-1594. E-mail: {dongu, villa}@icsl.ucla.edu.*
- *W. Luk is with the Department of Computing, Imperial College London, 180 Queen's Gate, London, SW7 2AZ, UK. E-mail: w.luk@imperial.ac.uk.*
- *P.H.W. Leong is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong. E-mail: pwl@cse.cuhk.edu.hk.*

Manuscript received 24 June 2005; revised 25 Dec. 2005; accepted 24 Jan. 2006; published online 21 Apr. 2006.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0211-0605.

of 375 MHz on a Xilinx Virtex-4 XC4VLX100-12 FPGA. The highlights of this paper include:

- a hardware architecture for the Box-Muller method,
- piecewise polynomial based function approximation units with range reduction,
- accurate error analysis and bit-width optimization leading to a guaranteed maximum absolute error bound of 1 ulp,
- exploration of hardware implementation of the proposed architecture targeting both advanced high-speed FPGAs and low-cost FPGAs, and
- the only reported Gaussian noise generator with a formal error analysis.

The rest of this paper is organized as follows: Section 2 covers background material and previous work. Section 3 provides an overview of the proposed design flow and the Box-Muller hardware architecture. Section 4 describes how we evaluate the elementary functions associated with the Box-Muller method. Section 5 introduces the MiniBit bit-width optimization approach. Section 6 describes the error analysis and bit-width optimization procedures for our Box-Muller architecture. Section 7 describes technology-specific implementation of the hardware architecture on Xilinx FPGAs. Section 8 discusses evaluation and results and Section 9 offers conclusions.

2 Background

In simulation environments, digital methods for generating Gaussian random variables are preferred over analog methods. Analog components allow truly random numbers, but are highly sensitive to environmental changes such as temperature and provide low throughputs of tens of kilobits per second. Such methods are often used for generating random seeds in cryptographic applications [5]. In contrast, digital methods are more desirable due to their robustness, flexibility, and speed. Although the resulting number sequences are pseudorandom as opposed to truly random, the period can be made sufficiently large such that the sequences never repeat themselves, even in the largest practical simulations.

The majority of the digital methods for generating Gaussian random variables are based on transformations on uniform random variables [6]. Popular methods include the Ziggurat method [7], the inversion method [8], the Wallace method [9], and the Box-Muller method [10]. Ziggurat is a class of rejection-acceptance methods, meaning that the output rate is not constant, making it less desirable to a hardware simulation environment. The inversion method involves the approximation of the inverse Gaussian cumulative distribution function (CDF), which is highly nonlinear, making it less suitable for a fixed-point hardware implementation. In contrast to all other methods, Wallace does not require the evaluation of elementary functions; new noise samples are generated by applying linear transformations to the previous pool of noise samples. Unfortunately, due to the feedback nature of the Wallace method, correlations can occur between successive transformations. These correlations can be made insignificant for any given simulation environment through proper parameter choice, but the need to manage them represents an additional complication inherent in the Wallace method. Our choice for hardware implementation is the Box-Muller method, which transforms two uniformly distributed variables into two normally distributed variables through a series elementary function evaluations.

Recently, there have been notable research contributions on the hardware implementations of the methods discussed above. Boutillon et al. [11] were the first to realize a hardware Gaussian noise generator based on the Box-Muller algorithm and the central limit theorem. The central limit theorem is employed to overcome approximation errors of the mathematical functions of the Box-Muller method. Their design occupies 437 logic cells on an Altera Flex 10K1000EQC240-1 FPGA and has a throughput of 24.5 million samples per second. Xilinx [12] have released an IP core and Fung et al. [13] have implemented an ASIC chip based on Boutillon et al.'s architecture. 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 six metal layer 0.18μm ASIC. Unfortunately, there are two drawbacks of this architecture: 1) It is limited to noise samples with magnitude less than 4σ and 2) statistical tests reveal that the quality of the noise samples is poor [14]. Unlike the two Box-Muller architectures above, the design presented in this paper employs highly accurate elementary function evaluation techniques, eliminating the need for the central limit theorem altogether.

In [15], we presented an architecture also based on the Box-Muller method and central limit theorem, but with more sophisticated function approximation techniques, resulting in significantly higher quality noise samples. The key differences between our previous work [15] and the work presented here are:

1. the way the mathematical functions are evaluated,
2. the accuracy of the noise samples,
3. the noise quality in the tails, as expressed by the maximum attainable σ multiple, and
4. the hardware efficiency, as illustrated in Table 2.

In [15], we evaluated the nonlinear functions of the Box-Muller method using degree one piecewise polynomials with nonuniform segmentation. The approximation and quantization errors were found to be high, forcing us to use the central limit theorem to improve noise quality. This resulted in an output rate of one sample per clock cycle. In addition, the maximum attainable σ multiple was 6.7. By contrast, in this work, we evaluate the elementary functions one by one and rigorously analyze the errors involved and, in doing so, are able to obtain a maximum σ multiple of 8.2. This added performance is critical for many large communications simulations. The central limit theorem is not required, resulting in an output rate of two samples per clock. Both samples are guaranteed to be accurate to 1 ulp, while using minimal bit-widths for the various signals in the data paths. In terms of hardware efficiency on an FPGA, this work achieves more than five times the throughput and occupies 40 percent less logic than [15].

In [14] and [16], we also describe hardware architectures of the Wallace and the Ziggurat methods. We provide

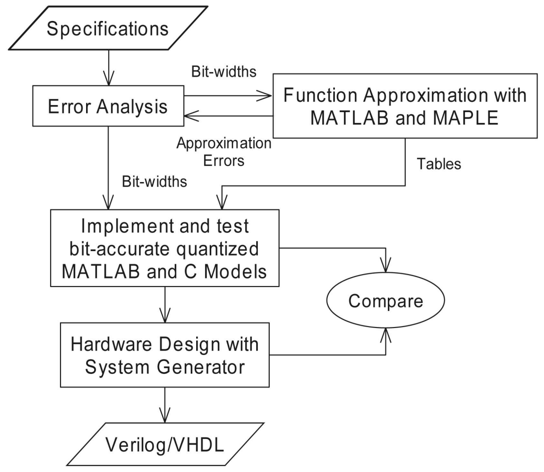
Specifications
Error Analysis
Bit-widths
Function Approximation with MATLAB and MAPLE
Approximation Errors
Bit-widths
Tables
Implement and test bit-accurate quantized MATLAB and C Models
Compare
Hardware Design with System Generator
Verilog/VHDL

Fig. 1. Design flow of our Box-Muller implementation.

detailed comparisons between different architectures in Section 8.

We choose a Xilinx Virtex-4 XC4VLX100-12 FPGA to realize our noise generator hardware architecture. Xilinx Virtex-4 FPGAs have the following three main types of resources: 1) user configurable elements known as “slices,” 2) storage elements known as “block RAMs,” and 3) multiply-and-add units known as “DSP slices.” A single slice is composed of 2.25 logic cells, which is the fundamental building block of Xilinx FPGAs. A logic cell is comprised of a 4-input lookup table, which can also be used as a 16×1 RAM or a 16-bit shift register, a multiplexor, and a register. A slice contains additional resources, such as multiplexors and carry logic, and, therefore, a slice is counted as being equivalent to 2.25 logic cells. Each block RAM can store 18Kb of data and a DSP slice can perform 18-bit by 18-bit multiplication followed by 48-bit addition. The Xilinx Virtex-4 XC4VLX100-12 device contains 49,152 slices, 240 block RAMs, and 96 DSP slices.

3 Design Flow and Architecture

This section provides an overview of our design methodology and introduces the operations involved in the Box-Muller method. We also discuss the specifications given for our noise generator and their implications for the Box-Muller method.

3.1 Design Flow

The design flow of our Box-Muller implementation is illustrated in Fig. 1. We first start by devising the specifications for the noise generator, which includes the periodicity of the samples, noise precision requirements, and throughput requirements. Although software implementations often use floating-point arithmetic, fixed-point arithmetic is often preferred for hardware implementations due to its area efficiency.

In order to meet the accuracy requirements of the noise samples, careful error analysis needs to be performed on the fixed-point data paths to determine the minimal bit-widths required. We generate the polynomial coefficient tables using MATLAB and the MATLAB Symbolic Toolbox, which contains the kernel of the MAPLE linear algebra package. After the bit-widths have been determined and the polynomial coefficient tables have been generated, we implement MATLAB and C models of the noise generator. These software models are programmed to be bit-accurate to the actual hardware realization, by emulating the quantization effects of the arithmetic operations.

Through comparisons with IEEE double-precision floating-point arithmetic, tests are conducted to check if the accuracy requirements of the noise samples are met. After the software implementations are finalized, we implement the hardware design using Xilinx System Generator [17], which is a MATLAB Simulink library and can generate Verilog or VHDL. The hardware design is verified carefully to ensure that it behaves the same way as the software models.

3.2 Architecture for the Box-Muller Method

The Box-Muller method starts with two independent uniform random variables, u_0 and u_1, over the interval $[0, 1)$. The following mathematical operations are performed to generate two samples, x_0 and x_1, of a Gaussian distribution $N(0, 1)$.

$$e = -2\ln(u_0), \tag{1}$$

$$f = \sqrt{e}, \tag{2}$$

$$g_0 = \sin(2\pi u_1), \tag{3}$$

$$g_1 = \cos(2\pi u_1), \tag{4}$$

$$x_0 = f \times g_0, \tag{5}$$

$$x_1 = f \times g_1. \tag{6}$$

The above equations lead to an architecture depicted in Fig. 2. Although traditional linear feedback shift registers (LFSRs) are often sufficient as a uniform random number generator (URNG), Tausworthe URNGs [18] are fast and occupy less area. Furthermore, they provide superior randomness when evaluated using the Diehard random number test suite [19] as demonstrated in [16]. Our Tausworthe URNG follows the algorithm presented by L'Ecuyer [20], which combines three LFSR-based URNGs to obtain improved statistical properties. It generates a 32-bit uniform random number per clock and has a large period of $2^{88} (\approx 10^{25})$. Its implementation in C code is illustrated in Fig. 3.

The implementation of the three function evaluation units, logarithm, square root, and the sine and cosine (sin/cos) unit, will all be analyzed in Section 4 and Section 6. The two numbers shown in brackets for each signal in Fig. 2 indicate the total bit-width and the fraction bit-width of the signal. The derivation of these bit-widths will be discussed in detail in Section 6.

3.3 Specifications

Two key specifications are set for our noise generator: periodicity of 10^{15} and 16-bit noise samples. In order to meet the periodicity requirement, the URNGs should have a

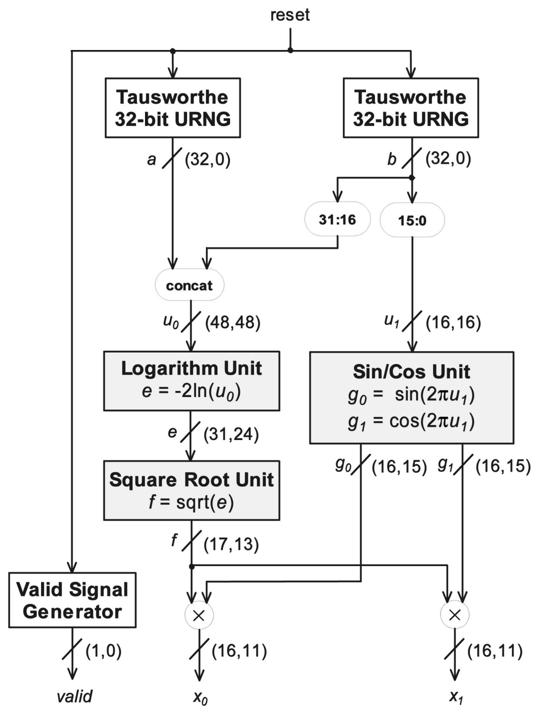

Fig. 2. Overview of our Gaussian noise generator architecture based on the Box-Muller method.

```
unsigned long s0, s1, s2, b;

unsigned long taus()
{
   b  = (((s0 << 13) ^  s0) >> 19);
   s0 = (((s0 & 0xFFFFFFFE) << 12) ^ b);
   b  = (((s1 <<  2) ^  s1) >> 25);
   s1 = (((s1 & 0xFFFFFFF8) <<  4) ^ b);
   b  = (((s2 <<  3) ^  s2) >> 11);
   s2 = (((s2 & 0xFFFFFFF0) << 17) ^ b);
   return s0 ^ s1 ^ s2;
}
```

Fig. 3. Description of the Tausworthe URNG in C code.

period of at least 10^{15}. But, at the same time, the noise samples should follow the ideal Gaussian distribution as closely as possible over the period. By examining the normal distribution $N(0,1)$ using MAPLE, we observe that we need to be able to represent up to 8.1σ for a population of 10^{15} samples. In other words, the probability of the absolute value of a single sample from that population being larger than 8.1σ is less than 0.5 (0.444 to be exact). By examining (1)-(6), the maximum σ value is determined by the smallest value of f, which in turn is determined by the smallest value of u_0, i.e.,

$$8.1 \geq \sqrt{-2\ln(u_0)} \Rightarrow u_0 \leq 5.66 \times 10^{-15}. \tag{7}$$

We use 48 bits for u_0, which gives a minimum value of $u_0 = 2^{-48} = 3.55 \times 10^{-15}$, meeting the requirement in (7). This also means that the maximum σ value we can attain is 8.2. There is no logical way to determine the number of bits required for u_1, except that it should have a good resolution. We use 16 bits for u_1, which is the same bit-width as the noise samples. Hence, conveniently, two 32-bit Tausworthe URNGs are utilized to provide the 48 bits and 16 bits required for u_0 and u_1.

We use two's complement fixed-point representation for the noise samples. The maximum absolute value of 8.2 means that 5 bits are sufficient for the integer bit-width (IB), leaving us with 11 bits for the fraction bit-width (FB). We would like to represent the noise samples as accurately as possible within the given 16 bits. Our criterion for evaluating the accuracy is ulp. The ulp of a fixed-point number with 11 bits fraction is 2^{-11}. There are two main types of rounding commonly used in computer arithmetic: faithful rounding and exact rounding. Faithful rounding means that results are accurate to 1 ulp (rounded to the nearest or next nearest) and exact rounding means that results are accurate to $1/2$ ulp (rounded to the nearest). Exact rounding is difficult to achieve due to a problem known as the table maker's dilemma [21] and has a rather large area penalty [22], hence we opt for faithful rounding in this work. Therefore, for every noise sample, the maximum absolute error compared against infinite precision should be less than or equal to 2^{-11}. For the purposes of this analysis, we regard IEEE double-precision floating-point to be "infinitely precise" since it is many orders of magnitude more accurate than the precision we are aiming for.

4 Function Evaluation

Throughout this paper, we shall denote the bit-width of a signal x as B_x, the integer bit-width as IB_x, the fraction bit-width as FB_x, and its associated error as E_x.

Consider an elementary function $f(x)$, where x and $f(x)$ have a given range $[a,b]$ and precision requirement. The evaluation $f(x)$ typically consists of three steps [23]:

1. range reduction: reducing x over the interval $[a,b]$ to a more convenient y over a smaller interval $[a',b']$,
2. function approximation on the reduced interval, and
3. range reconstruction: expansion of the result back to the original result range.

Our function evaluation steps for $-2\ln(u_0)$, $\sqrt{e}$, $\sin(2\pi u_1)$, and $\cos(2\pi u_1)$ are based on the methods presented in [22] and [24]. If a variable x is separated into a sign bit S_x, a mantissa M_x, where $M_x = [1,2)$, and an exponent E_x, i.e., $x = (-1)^{S_x} \times M_x \times 2^{E_x}$, the following mathematical identities are used for the evaluation of the logarithm and the square root when $S_x = 0$:

$$\ln(M_x \times 2^{E_x}) = \ln(M_x) + E_x \times \ln(2), \tag{8}$$

$$\sqrt{M_x \times 2^{E_x}} = \begin{cases} \sqrt{M_x} \times 2^{E_x/2}, E_x \bmod 2 = 0 \\ \sqrt{2 \times M_x} \times 2^{(E_x-1)/2}, E_x \bmod 2 = 1. \end{cases} \tag{9}$$

We observe from the equations above that the range reduction steps of the logarithm and the square root are essentially fixed-point to floating-point conversions. For the evaluation of sin/cos, we exploit the symmetric and periodic behavior of the two functions. As illustrated in Fig. 4, only $\cos(x)$ over $x = [0, \pi/2)$ needs to be approximated.

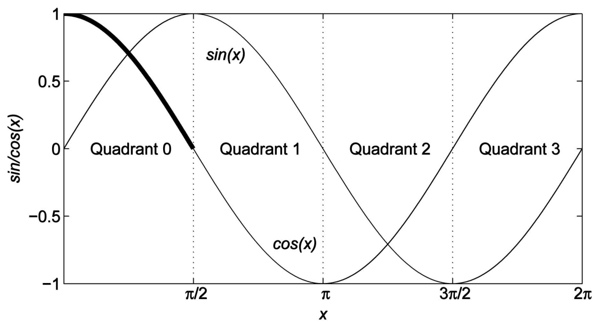

Fig. 4. Only the thick line is approximated for the evaluation of sin/cos. The most significant two bits of x are used to index one of the four quadrant and the remaining bits select a location within the quadrant.

To approximate the functions over the linear range, we use piecewise polynomials with uniform segments. Polynomials are evaluated using Horner's rule:

$$y = ((C_d x + C_{d-1})x + \ldots)x + C_0, \tag{10}$$

where x is the input, d is the polynomial degree, and C_i are the polynomial coefficients. The hardware architecture for a degree two piecewise polynomial is shown in Fig. 5. A degree one architecture is similar but without the first multiply-and-add unit. The input interval is split into $2^{B_{x_A}}$ equally sized segments. The B_{x_A} leftmost bits of the argument x serve as the index into the table, which holds the polynomial coefficients for that particular interval. Since x_A is implicitly known for a given interval, we use x_B instead of x for the polynomial arithmetic to reduce the size of the operators. We scale x_B to be over $[0,1)$ to simplify the error analysis phase in Section 6. If $x = [0,1)$, this would involve masking out the bits corresponding to x_A and shifting x by B_{x_A} bits to the left.

The polynomial coefficients are found in a minimax sense that minimize the maximum absolute error [23] using MAPLE. However, the coefficients are generated with the assumption that x will be used for the polynomial arithmetic. If we want to use x_B instead, the coefficients need to be transformed. We consider a degree one polynomial to illustrate the transformation process, but the same principle can be applied to other degrees. For a given segment, we obtain x_B by

$$x_B = (x - x_A) \times 2^{B_{x_A}}. \tag{11}$$

Rearranging the equation, we obtain

$$x = \frac{x_B}{2^{B_{x_A}}} + x_A. \tag{12}$$

A degree one polynomial is represented by the equation

$$y = C_1 x + C_0 \tag{13}$$

and, by substituting (12) into (13), we get

$$y = \frac{C_1}{2^{B_{x_A}}} x_B + C_1 x_A + C_0. \tag{14}$$

By examining the first and zeroth order terms, the new transformed polynomial coefficients $\bar{C}_1$ and $\bar{C}_0$ are now $\frac{C_1}{2^{B_{x_A}}}$ and $C_1 x_A + C_0$, respectively.

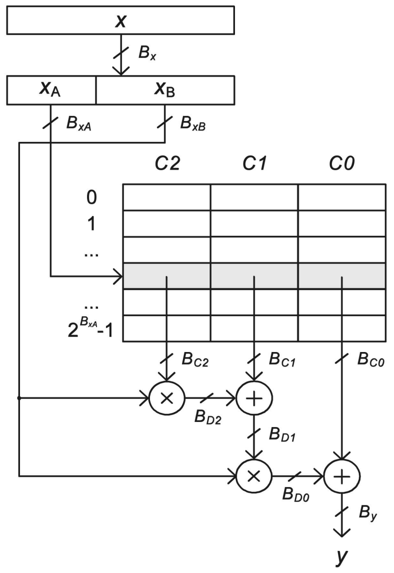

Fig. 5. Hardware architecture for degree two piecewise polynomials.

With the proposed architecture in Fig. 5, we need $d+1$ table lookups, d multiplications, and d additions. The size of the lookup table is given by

$$\text{table size} = 2^{B_{x_A}} \times \sum_{i=0}^{d} B_{C_i} \text{ bits.} \tag{15}$$

The main challenge is to find the minimal bit-widths for each signal while meeting the output error constraints. We discuss how we achieve this in the next two sections.

5 The MiniBit Bit-Width Optimization Approach

In this section, we briefly describe the fundamental principles behind the MiniBit bit-width optimization approach [25], a technique for optimizing fixed-point signals using analytical error expressions with a guaranteed maximum error bound.

In digital systems, signals need to be quantized to finite precisions. In order to minimize area and meet the accuracy requirement, each signal should use the minimal bit-width, while the final output signals should obey the accuracy requirements. There are two main ways to quantize a signal: truncation and round-to-nearest. Truncation and round-to-nearest can cause a maximum error of 2^{-FB} (1 ulp) and 2^{-FB-1} (1/2 ulp), respectively. Truncation chops bits off the least significant parts and requires no extra hardware resources. Although round-to-nearest requires a small adder, we opt for round-to-nearest since it allows for smaller bit-widths than truncation.

Let $\tilde{a}$ be the quantized version and $E_{\tilde{a}}$ be the error of the signal a. Thus,

$$\tilde{a} = a + E_{\tilde{a}}. \tag{16}$$

For addition/subtraction operations $y = a \pm b$, the error $E_{\tilde{y}}$ at the output y is given by

$$\begin{aligned}\tilde{y} &= \tilde{a} \pm \tilde{b} = a \pm b + E_{\tilde{a}} + E_{\tilde{b}} + 2^{-FB_{\tilde{y}}-1} \\ \Rightarrow E_{\tilde{y}} &= E_{\tilde{a}} + E_{\tilde{b}} + 2^{-FB_{\tilde{y}}-1},\end{aligned} \tag{17}$$

where $2^{-FB_{\tilde{y}}-1}$ is the rounding error of $\tilde{y}$. For multiplication, we get

$$\begin{aligned}\tilde{y} &= \tilde{a}\tilde{b} \\ &= ab + aE_{\tilde{b}} + bE_{\tilde{a}} + E_{\tilde{a}}E_{\tilde{b}} + 2^{-FB_{\tilde{y}}-1} \\ \Rightarrow E_{\tilde{y}} &= aE_{\tilde{b}} + bE_{\tilde{a}} + E_{\tilde{a}}E_{\tilde{b}} + 2^{-FB_{\tilde{y}}-1}.\end{aligned} \tag{18}$$

$E_{\tilde{y}}$ would be at its maximum when a and b are at their maximum absolute values.

Consider a slightly more complex example: $y = a \times b + c$. We want to quantize signals after each operation, hence, we compute the example in two steps:

$$t = a \times b, \tag{19}$$

$$y = t + c. \tag{20}$$

The error $E_{\tilde{y}}$ is given by

$$\begin{aligned}E_{\tilde{t}} &= aE_{\tilde{b}} + bE_{\tilde{a}} + E_{\tilde{a}}E_{\tilde{b}} + 2^{-FB_{\tilde{t}}-1} \\ \Rightarrow E_{\tilde{y}} &= E_{\tilde{t}} + E_{\tilde{c}} + 2^{-FB_{\tilde{y}}-1}.\end{aligned} \tag{21}$$

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

$$2^{-FB_{\tilde{y}}} \geq \max(E_{\tilde{y}}). \tag{22}$$

Suppose we want $\tilde{y}$'s fraction to be accurate to 16 bits, with a, b, and c being constants rounded to the nearest. Assume that the maximum absolute values are $\max(|a|) = 2$, $\max(|b|) = 4$. Note that $\max(|c|)$ is irrelevant to this error analysis. Then, using (21) and (22), we get

$$\begin{aligned}2^{-16} &\geq 2 \times 2^{-FB_{\tilde{b}}-1} + 4 \times 2^{-FB_{\tilde{a}}-1} \\ &\quad + 2^{-FB_{\tilde{a}}-FB_{\tilde{b}}-2} + 2^{-FB_{\tilde{t}}-1} + 2^{-FB_{\tilde{c}}-1} \\ &\quad + 2^{-17} \\ &\geq 2^{2-FB_{\tilde{a}}} + 2^{1-FB_{\tilde{b}}} + 2^{-FB_{\tilde{a}}-FB_{\tilde{b}}-1} \\ &\quad + 2^{-FB_{\tilde{c}}} + 2^{-FB_{\tilde{t}}}.\end{aligned} \tag{23}$$

The error terms are functions of the fraction bit-widths of the signals and the maximum value of the signals, hence the optimization problem is to find the minimal fraction bit-widths for the signals $\tilde{a}$, $\tilde{b}$, $\tilde{c}$, and $\tilde{t}$ while satisfying the inequality. For small problems like this, enumeration may be appropriate, but, for complex equations involving a large number of signals, methods such as simulated annealing can be used [25]. Several sets of optimal solutions exist which depend on the cost functions of the operations. One solution to this example would be $FB_{\tilde{a}} = 21$, $FB_{\tilde{b}} = 19$, $FB_{\tilde{c}} = 18$, and $FB_{\tilde{t}} = 18$. Since the remainder of the discussion that follows concerns quantized signals, we shall omit the tilde ($\sim$) over the signal notation for readability.

Regarding the optimization of the integer bit-widths (IBs), in straightforward cases, it can often be performed manually by analytical analysis of the dynamic ranges of the signals. For less intuitive cases, one can rely on range analysis techniques such as affine arithmetic [26], as demonstrated in [25].

```
--------------- Generate u0 and u1 ---------------
a  = taus(); b = taus();
u0 = concat(a,b[31:16]);
u1 = b[15:0];

------------- Evaluate e = -2ln(u0) --------------

# Range Reduction
exp_e = LeadingZeroDetector(u0)+1;
x_e   = u0 << exp_e;

# Approximate -ln(x_e) where x_e = [1,2)
# Degree-2 piecewise polynomial
y_e = ((C2_e[x_e_B]*x_e)+C1_e[x_e_B])*x_e_B
      +C0_e[x_e_B];

# Range Reconstruction
ln2 = ln(2);
e'  = exp_e*ln2;
e   = (e'-y_e)<<1;

-------------- Evaluate f = sqrt(e) --------------

# Range Reduction
exp_f = 5-LeadingZeroDetector(e);
x_f'  = e >> exp_f;
x_f   = if(exp_f[0], x_f'>>1, x_f');

# Approximate sqrt(x_f) where x_f = [1,4)
# Degree-1 piecewise polynomial
y_f = C1_f[x_f_B]*x_f_B+C0_f[x_f_B];

# Range Reconstruction
exp_f' = if(exp_f[0], exp_f+1>>1, exp>>1);
f      = y_f << exp_f';

------------ Evaluate g0=sin(2*pi*u1) ------------
------------          g1=cos(2*pi*u1) ------------

# Range Reduction
quad  = u1[15:14];
x_g_a = u1[13:0];
x_g_b = (1-2^-14)-u1[13:0];

# Approximate cos(x_g_a*pi/2) and cos(x_g_b*pi/2)
# where x_g_a, x_g_b = [0,1-2^-14]
# Degree-1 piecewise polynomial
y_g_a = C1_g[x_g_a_B]*x_g_a_B+C0_g[x_g_a_B];
y_g_b = C1_g[x_g_b_B]*x_g_b_B+C0_g[x_g_b_B];

# Range Reconstruction
switch(seg)
  case 0:   g0 =  y_g_b; g1 =  y_g_a;
  case 1:   g0 =  y_g_a; g1 = -y_g_b;
  case 2:   g0 = -y_g_b; g1 = -y_b_a;
  case 3:   g0 = -y_g_a; g1 =  y_g_b;

---------------- Compute x0 and x1 ---------------
x0 = f*g0; x1 = f*g1;
```

Fig. 6. Pseudocode of the evaluation steps for the Box-Muller architecture.

6 Error Analysis and Bit-Width Optimization for Noise Generator

In this section, we describe how the MiniBit error expressions are used to optimize the bit-widths of the signals in our Box-Muller architecture using a bottom-up approach. We shall discuss the fraction bit-width (FB) optimization problem only, which we have found to be significantly more challenging than integer bit-width (IB) optimization. In order to find the optimal IBs, we analyze the signals carefully and examine their dynamic ranges. This manual optimization process is found to be feasible for the noise generator since the dynamic ranges of the signals are straightforward and predictable. We make the assumption that all function evaluations are faithfully rounded throughout. The evaluation steps for the Box-Muller architecture are described in the pseudocode in Fig. 6.

6.1 Error Analysis at the Output

The accuracy requirement of the noise samples x_0 and x_1 is faithful rounding, i.e., they should be accurate to 1 ulp. Knowing that the samples have 11 fractional bits, the requirements are

$$E_{x_0} \leq 2^{-11} \quad \text{and} \quad E_{x_1} \leq 2^{-11}. \tag{24}$$

We shall consider the data path to x_0 only since the error analysis is identical for x_1. Assuming we faithfully round f and g_0, we get the following error expression:

$$2^{-11} \geq g_0 \times 2^{-FB_f} + f \times 2^{-FB_{g_0}}. \tag{25}$$

We are interested in the worst-case error, which occurs when g_0 and f are at their maximum of 1 and 8.157. Hence, we get

$$2^{-11} \geq 2^{-FB_f} + 8.157 \times 2^{-FB_{g_0}}. \tag{26}$$

Given that f has a deeper computation chain than g_0, we would prefer that its computation requirements are less precise so that fewer bits are needed for its implementation. Through an exhaustive search, we find that $FB_f = 13$ and $FB_{g_0} = 15$ are the minimal bit-widths that meet the inequality in (26).

Now that we have determined the bit-widths for f and g_0, we can move on to the square root unit and the sin/cos unit. We shall first perform analysis for the sin/cos unit since it is easier to analyze due to the shorter computation chain.

6.2 Error Analysis for the Sin/Cos Unit

The sin/cos unit corresponds to lines 37-56 in Fig. 6. The range reduction and range reconstruction steps of the sin/cos unit are exact, i.e., there are no quantization steps involved. Hence, we only need to worry about the approximation steps. Because of the periodic and symmetric nature of $\sin(x)$ and $\cos(x)$, as shown in Fig. 4, we only approximate $\cos(x)$ over $[0, \pi/2)$. From the 16-bit input u_1, the most significant two bits are used to select a random quadrant from one of the four quadrants of $\sin(x)$ and $\cos(x)$ and the remaining 14 bits are used for the polynomial approximation. Two variables, x_{g_a} and x_{g_b}, are obtained from the 14 bits, which are used to compute g_0 and g_1.

The approximations required are $\cos(x_{g\_a}\pi/2)$ and $\cos(x_{g\_b}\pi/2)$. Since the two approximations share the same data path characteristics, we shall discuss $\cos(x_{g\_a}\pi/2)$ only. In the following analysis, x_{g_a} will be referred to as x_g for simplicity. We first need to decide what degree polynomial to use for the approximation: A low degree polynomial will require fewer computations at the expense of a larger table. In addition, shallower computation chains will accumulate fewer quantization errors. Hence, we would like to use the lowest degree possible as long as the table size is reasonable. This will, of course, depend on the function and the precision we are aiming for.

In Section 6.1, we derived that $FB_{g_0} = 15$. Knowing that the range reconstruction step only involves sign changes, we also need $FB_{y_g} = 15$. The signal y_g needs to be faithfully rounded to 15 bits fraction, i.e., $E_{y_g} \leq 2^{-15}$. When approximating a function with piecewise polynomials, we would like to know the minimal number of segments required for a given input range, polynomial degree, and output accuracy. A MATLAB program is written which uses MAPLE to compute the minimax polynomial coefficients and maximum approximation error for a given segment. It incrementally increases the number of segments by a power of two until all segments meet the user-specified output accuracy.

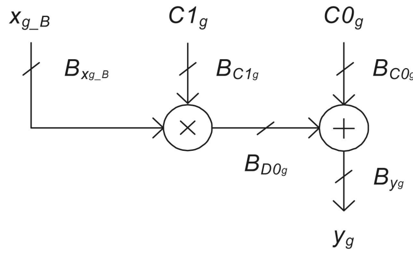

Fig. 7. Degree one polynomial approximation circuit for $y_g = \cos(x_g\pi/2)$.

When approximating $y_g = \cos(x_g\pi/2)$ with an accuracy of 2^{-15}, our MATLAB program reports that we need 128 segments for a degree one polynomial and 16 segments for a degree two polynomial. As a rough estimate of the coefficient table size, we assume that the coefficients have the same bit-width as the output y_g, i.e., 16 bits. From (15), we can infer that table sizes will be roughly 4,096 and 768 bits for degree one and two polynomials, respectively. Given that a block RAM on Virtex-II and Virtex-4 FPGAs is 18Kb, enough to fit the 4,096 bits for degree one polynomials, we opt for a degree one polynomial. Based on the above information and our experience with piecewise polynomials, a good rule of thumb is to use degree one polynomials for target precisions lower than 20 bits and degree two polynomials for above 20 bits.

Fig. 7 shows the data path for a degree one polynomial approximation to $\cos(x_g\pi/2)$. It corresponds to line 48 and line 49 in Fig. 6. Knowing that $E_{y_g} \leq 2^{-15}$ and using the MiniBit techniques from Section 5, we get the following error expression:

$$2^{-15} \geq E_{D0_g} + E_{C0_g} + 2^{-16} + E_{approx_g}. \tag{27}$$

Note that the 2^{-16} term is the rounding error of y_g. Because the function is approximated by a polynomial, there is an inherent approximation error E_{approx_g}, regardless of quantization effects. For this approximation, the maximum polynomial approximation error is reported to be 9.26458×10^{-6} by MAPLE.

The polynomial coefficients are rounded to the nearest, hence

$$2^{-16} \geq E_{D0_g} + 2^{-FB_{C0_g}-1} + 9.26458 \times 10^{-6}. \tag{28}$$

The error at $D0_g$ is expressed as

$$E_{D0_g} = x_{g\_B} \times E_{C1_g} + C1_g \times E_{x_{g\_B}} + 2^{-FB_{D0_g}-1}. \tag{29}$$

The maximum value of x_{g_B} is one and, since x_{g_B} has no errors, we get

$$E_{D0_g} = 2^{-FB_{C1_g}-1} + 2^{-FB_{D0_g}-1} \tag{30}$$

and, by substituting (30) into (28), we obtain

$$5.99421 \times 10^{-6} \geq 2^{-FB_{C1_g}-1} + 2^{-FB_{C0_g}-1} + 2^{-FB_{D0_g}-1}. \tag{31}$$

We find that $FB_{C1_g} = 18$, $FB_{C0_g} = 18$, and $FB_{D0_g} = 18$ are the minimal bit-widths that satisfy the inequality. In the actual ROM, we store the two coefficients $C1_g$ and $C0_g$ as integers. The coefficients for $C1_g$ all contain six leading zeros in the fraction part and some values of $C0_g$ turn out to be slightly larger than one. Moreover, all values of $C1_g$ and $C0_g$ are found to have the same sign. Hence, the bit-width required for $C1_g$ is 12 bits (with the six redundant leading six bits eliminated) and $C0_g$ is 19 bits (with one extra bit to cover the integer part). As mentioned earlier, 128 segments are required for a degree one approximation to $\cos(x_g\pi/2)$ and, hence, the total table size needed for the sin/cos unit is $(12 + 19) \times 128 = 3,968$ bits.

6.3 Error Analysis for the Square Root Unit

The square root unit corresponds to line 22 to line 35 in Fig. 6 and is derived from (9). It is perhaps the most challenging error analysis step since it suffers from propagation errors at its input produced by the logarithm unit. We shall discuss why the propagation errors make the error analysis difficult and how we overcome this problem.

Since u_0 is over $[0, 1 - 2^{-48}]$, we know that the output of the logarithm unit $e = -2\ln(u_0)$ will be over $[0, 66.54]$. $u = 0$ is treated as a special case in which we simply set $e = 0$. Since e is unsigned, the number of its integer bits will be seven. The leading zero detector (LZD) returns the number of leading zeros from the most significant bit. Since e has seven integer bits and we want x'_f to be over $[2, 4)$ (i.e., x'_f has two integer bits), we subtract the output of the LZD from five in order to obtain the number of bits to shift e.

The minimum value of exp_f occurs when e has just a one in its least significant bit. Hence,

$$\min(exp_f) = -(FB_e + 1). \tag{32}$$

Looking at line 34 in the range reconstruction step, we have the following relationship between exp_f and exp'_f:

$$exp'_f = \begin{cases} exp_f/2, & exp_f[0] = 0 \\ (exp_f + 1), & exp_f[0] = 1. \end{cases} \tag{33}$$

The maximum value of exp_f is five (i.e., when there are no leading zeros in e), hence the maximum value of exp'_f will be

$$\max(exp'_f) = \begin{cases} 2, & exp_f[0] = 0 \\ 3, & exp_f[0] = 1, \end{cases} \tag{34}$$

where $exp_f[0]$ denotes the least significant bit of exp_f. This shows how much the result at y_f can be shifted to the left, which would amplify its error.

By examining line 35 of the range reconstruction step, we get the following error relationship between f and y_f:

$$E_f = E_{y_f} \times 2^{exp'_f} \tag{35}$$

and, from Section 6.1, we know that f should be faithfully rounded to 13 fraction bits. Hence, we get the following inequality:

$$E_{y_f} \leq 2^{-13-exp'_f}. \tag{36}$$

From the expression above, we can see that the accuracy requirement of y_f will depend on the value of exp'_f. Based on (34), we know that $\max(exp'_f) = 3$, therefore y_f should be at least accurate to 2^{-16}.

We shall make the assumption that e is faithfully rounded, i.e., $\max(E_e) = 2^{-FB_e}$. Looking at line 26 of the range reduction step, we get the following error relationship between e and x'_f:

$$E'_{x_f} = E_e \times 2^{-exp_f} = 2^{-(FB_e + exp_f)} \tag{37}$$

and, therefore,

$$E_{x_f} = \begin{cases} 2^{-(FB_e + exp_f)}, & exp_f[0] = 0 \\ 2^{-(FB_e + exp_f - 1)}, & exp_f[0] = 1. \end{cases} \tag{38}$$

As discussed in Section 4, $E_{x_{f\_B}}$ is a masked and left-shifted version of E_{x_f} by $B_{x_{f\_A}}$. The left-shifting by $B_{x_{f\_A}}$ will amplify the error by $2^{B_{x_{f\_A}}}$. Hence, we get the following error expression at x_{f_B}:

$$E_{x_{f\_B}} = \begin{cases} 2^{-(FB_e + exp_f - B_{x_{f\_A}})}, & exp_f[0] = 0 \\ 2^{-(FB_e + exp_f - 1 - B_{x_{f\_A}})}, & exp_f[0] = 1. \end{cases} \tag{39}$$

Based on the derivations above, we can now consider the polynomial approximation part. Since the accuracy requirement of y_f is 16 bits at least, we opt for a degree one polynomial. For both cases, when $exp_f[0] = 0$ and $exp_f[0] = 1$, which correspond to the intervals $[2, 4)$ and $[1, 2)$, 64 segments are required, meaning that $B_{x_{f\_A}} = 6$. Although we use two independent coefficient tables for the two cases, a single multiply-and-add unit for the degree one polynomial arithmetic can be shared between them.

Using a similar analysis to the sin/cos approximation unit with the exception that the input x_{f_B} contains a propagated error $E_{x_{f_B}}$ and noting that $\max(x_{f\_B}) = 1$, we get the following error expression at the output y_f:

$$E_{y_f} = C1_f E_{x_{f\_B}} + 2^{-FB_{C1_f}-1} + 2^{-FB_{C1_f}-1} E_{x_{f\_B}} + 2^{-FB_{C1_f}-1} + 2^{-FB_{C0_f}-1} + 2^{-FB_{D0_f}-1} + 2^{-17} + E_{approx_f}. \tag{40}$$

Note that 2^{-17} is the rounding error at y_f. Recalling that $\min(exp_f) = -(FB_e + 1)$ and $B_{x_{f\_A}} = 6$, the product $C1_f E_{x_{f\_B}}$ is most likely to be the dominating error factor. Substituting (36) and (39) into (40) and considering only the dominating error factor, we get

$$2^{-13-exp_f/2} \geq 2^{-(FB_e + exp_f - 6)} C1_f, \quad exp_f[0] = 0, \tag{41}$$

$$2^{-13-(exp_f+1)/2} \geq 2^{-(FB_e + exp_f - 7)} C1_f, \quad exp_f[0] = 1. \tag{42}$$

Let us consider the case when $exp_f[0] = 0$ first, i.e., when FB_e is odd. It is found that $\max(C1_f) = 0.01101$ when $exp_f[0] = 0$. E_{y_f} will be at its maximum when exp_f is at its minimum of $-(FB_e + 1)$. Using (41), we get

$$2^{(FB_e+1)/2-13} \geq 2^{-(FB_e-(FB_e+1)-6)} \times 0.01101 \tag{43}$$

$$\Rightarrow FB_e \geq 25.98887 \tag{44}$$

$$\Rightarrow FB_e = 27. \tag{45}$$

For the case when $exp_f[0] = 1$, i.e., when FB_e is even, we find that $\max(C1_f) = 0.00778$. Hence,

$$2^{FB_e/2-13} \geq 2^{-(FB_e - FB_e - 6)} \times 0.00778 \tag{46}$$

$$\Rightarrow FB_e \geq 23.98881 \tag{47}$$

$$\Rightarrow FB_e = 24. \tag{48}$$

Based on the two derivations above, we choose the case when FB_e is even, which is 24 bits.

Now that we have derived FB_e, we need to compute the minimal bit-widths of the signals $C1_f$, $C0_f$, and $D0_f$ in the polynomial arithmetic. E_{approx_f} is reported to be 5.32722×10^{-6} and 3.76332×10^{-6} for the cases when $exp_f[0] = 0$ and $exp_f[0] = 1$, respectively. Since the error requirement at y_f can vary depending on the value of exp_f, we shall consider the two extreme cases when exp_f is at its minimum and maximum. At $\min(exp_f) = -25$, we get

$$\begin{aligned} 2^{-1} \geq{} & 2^6 \times 0.00778 + 2^{-FB_{C1_f}-1} \\ & + 2^{-FB_{C1_f}-1}2^6 + 2^{-FB_{C1_f}-1} \\ & + 2^{-FB_{C0_f}-1} + 2^{-FB_{D0_f}-1} \\ & + 2^{-17} + 3.76332 \times 10^{-6} \\ \Rightarrow 4.15247 \times 10^{-6} \geq{} & 65 \times 2^{-FB_{C1_f}} + 2^{-FB_{C0_f}} \\ & + 2^{-FB_{D0_f}} \end{aligned} \tag{49}$$

and, at $\max(exp_f) = 5$, we get

$$\begin{aligned} 7.73123 \times 10^{-6} \geq{} & (1 + 2^{-25}) \times 2^{-FB_{C1_f}} + 2^{-FB_{C0_f}} \\ & + 2^{-FB_{D0_f}}. \end{aligned} \tag{50}$$

Given that FB_{C1_f} is likely to be around 16 bits or larger, (50) places more stringent bit-width requirements on the signals than (49). Through enumeration on (50), we find that $FB_{C1_f} = 18$, $FB_{C0_f} = 19$, and $FB_{D0_f} = 19$ are the minimal bit-widths. As in the sin/cos units, some coefficients have leading zeros and some are larger than one. Hence, when we store the coefficients as integers, we need 12 bits and 20 bits for $C1_f$ and $C0_f$, respectively. We need to store two tables for the square root unit: one each for the intervals $[2, 4)$ and $[1, 2)$. Recalling that 64 entries are required for each table, the total table size is $(12 + 20) \times 64 \times 2 = 4{,}096$ bits.

6.4 Error Analysis for the Logarithm Unit

The logarithm unit corresponds to line 6 to line 20 of Fig. 6. Examining the range reconstruction steps in line 18 to line 20, we find that there are the two intermediate signals: $ln2$ (which stores the constant $\ln(2)$) and e'. We first need to determine the fraction bit-widths of these two signals and the output of the polynomial arithmetic y_e. Noting that $\max(exp_e) = B_{u_0} = 48$ and constant $ln2$ is rounded to the nearest, the following error relationship exists between the signals:

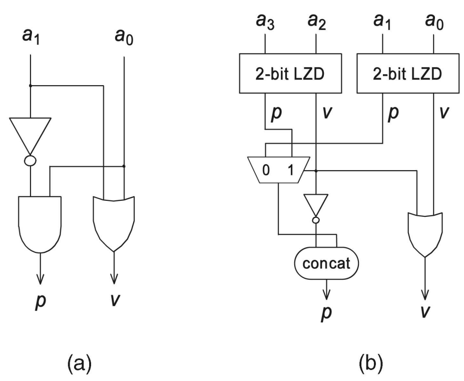

Fig. 8. Circuits for 2-bit and 4-bit leading zero detectors (LZDs): a is the input data, p indicates the number of leading zeros, and v is a valid bit. (a) 2-bit LZD. (b) 4-bit LZD.

$$\begin{aligned} E_{e'} &= 48 \times 2^{-FB_{ln2}-1} + 2^{-FB_{e'}-1} \\ E_e &= 2(E_{e'} + E_{y_e}) + 2^{-FB_e-1} \\ &= 2(48 \times 2^{-FB_{ln2}-1} + 2^{-FB_{e'}-1} + E_{y_e}) \\ &\quad + 2^{-FB_e-1}. \end{aligned} \tag{51}$$

In Section 6.3, we derived the fraction bit-width of e to be 24 bits. Assuming y_e is faithfully rounded, using (51), we get

$$2^{-26} \geq 48 \times 2^{-FB_{ln2}-1} + 2^{-FB_{e'}} + 2^{-FB_{y_e}}. \tag{52}$$

Through enumeration, the minimal fraction bit-width are $FB_{ln2} = 32$, $FB_{e'} = 28$, and $FB_{y_e} = 27$.

Since y_e needs to be accurate to 27 bits, degree one polynomials will likely require a large number of segments. Hence, we choose degree two polynomials for this approximation. The error analysis for the degree two polynomial arithmetic is performed in the same manner as the sin/cos unit case, where the input to the polynomial arithmetic contains no errors. After analysis, the minimal bit-widths for the three polynomial coefficients $C2_e$, $C1_e$, $C0_e$ are found to be 13, 22, and 30 bits, respectively. The approximation requires 256 segments, hence the total table size for the logarithm unit is $(13 + 22 + 30) \times 256 = 16{,}640$ bits.

7 Implementation

This section discusses how our Box-Muller hardware architecture is mapped into FPGA technology.

As can be seen in Fig. 3, the operations involved in the Tausworthe URNG are rather straightforward and a series of XOR and constant shifts are needed. We also need multiplexors to reload the original seeds when the reset signal in Fig. 2 is set high. The two Tausworthe URNGs occupy just 150 slices and are fast enough not to require any pipeline stages.

To implement the leading zero detectors (LZDs) of the logarithm and the square root units, we choose the methodology proposed by Oklobdzija [27]. It allows us to implement any size LZDs using a 2-bit LZD as a basic building block in a hierarchical and modular manner. Fig. 8 shows a 2-bit and a 4-bit LZD. This LZD architecture occupies little area on the device, for instance, the 48-bit LZD used in the logarithm unit occupies just 46 slices. The LZD is fast and, hence, it is used combinatorially.

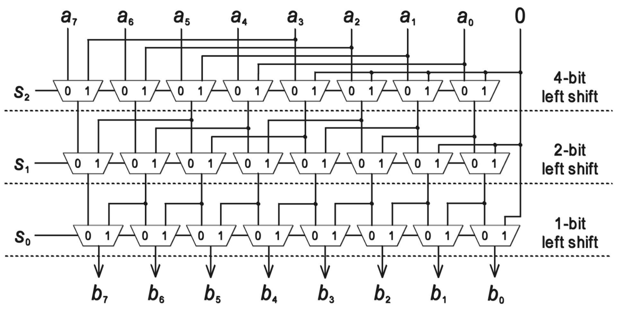

Fig. 9. An 8-bit logical left barrel shifter: a is the input data, s is the number of bits to shift, and b is the shifted data.

Another important component of our design is the barrel shifter, which is required at the range reduction step of the logarithm unit and at the range reduction and reconstruction steps of the square root unit. We employ the logical barrel shifter described in Pillmeier et al.'s survey paper [28]. An example of an 8-bit logical left barrel shifter is shown in Fig. 9. As an example, the 48-bit left barrel shifter used in the logarithm unit occupies 130 slices and has two pipeline stages.

In Section 6, we determined the table sizes of the three function evaluation units to be 3,968, 4,096, and 16,640 bits for sin/cos, square root, and logarithm, respectively, resulting in a total memory requirement of 24,704 bits. A Virtex-4 block RAM is reported to be 18Kb, but 18Kb can only be utilized for certain memory width and depth combinations. Due to this constraint, we are unable to fit the logarithm table into a single block RAM. We adopt a three block RAM structure to store the tables, as illustrated in Fig. 10. In block RAM 0, the $C2_e$ and $C1_e$ coefficients of the logarithm unit are stored. They are concatenated to form a single long word. In block RAM 1, the upper 256 locations store the $C0_e$ coefficients of the logarithm unit and the lower 128 locations store the two square root unit tables. This RAM is dual-ported since the logarithm and square root unit need to access their coefficients every cycle. Block RAM 2 stores the coefficients for the sin/cos unit. It is

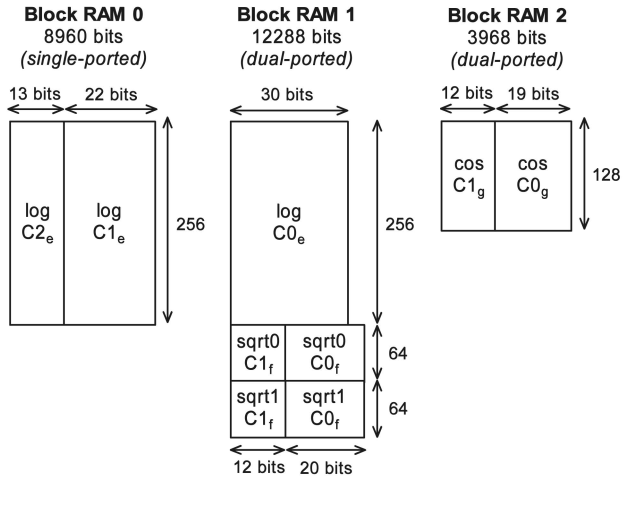

Fig. 10. Structures of the three block RAMs used.

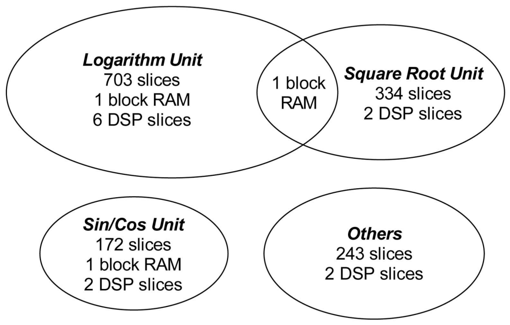

Fig. 11. Hardware area comparisons of the various units in our Box-Muller architecture.

dual-ported to allow simultaneous reads for sin and cos evaluations.

Our Box-Muller architecture is mapped into FPGA implementations using Xilinx System Generator 7.1. The implementations are heavily pipelined to maximize the throughput/area ratio. Synplicity Synplify Pro 8.1 is used for synthesis and Xilinx ISE 7.1.03i is used for place-and-route with maximum effort level. An implementation on a Xilinx Virtex-4 XC4VLX100-12 FPGA occupies 1,452 slices, three block RAMs, and 12 DSP slices. Fig. 11 illustrates hardware area comparisons of the various units in the architecture. It is capable of a maximum clock speed of 375 MHz, one of the rounding circuitry in the logarithm unit being the critical path. Since we can generate two samples per clock, a throughput of 750 million noise samples per second is attainable. There are 53 pipeline stages, hence valid noise samples will appear 53 clock cycles after the reset signal is set from high to low. In addition, we have used the noise generator for channel code simulations on an FPGA platform, where the hardware utilization, quality, and performance were confirmed.

Higher throughputs can be obtained by exploiting parallelism. We are able to fit 37 instances of the noise generator on a Xilinx Virtex-II Pro XC2VP100-7 device, which, in fact, has more area resources than the largest Virtex-4 device. They occupy 45,077 slices, 84 block RAMs, and 336 MULT18X18s (embedded multipliers). Due to routing congestion on the chip, we are only able to achieve a clock speed of 95 MHz. Despite the significantly lower clock speed, with the parallelism of the multiple instances, we are able to achieve a throughput of seven billion noise samples per second.

8 Evaluation and Results

This section describes how we verify the accuracy of our noise samples and compare the performance of our design against other hardware and software implementations.

In order to test the accuracy of the noise samples, we compare 10 billion noise samples from our noise generator against the ones generated from IEEE double-precision floating-point arithmetic. As anticipated, the ulp error of all samples are found to be less than 1 ulp for all samples. To verify the accuracy of the samples in the high σ regions, we test 10 billion noise samples over the range of sigma multiples $[-7, -4]$ and $[4, 7]$ and, again, all samples are

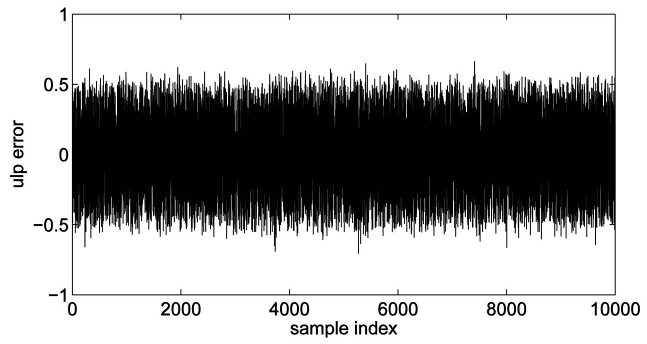

Fig. 12. Error plot for 10,000 randomly generated samples from our noise generator compared against IEEE double-precision floating-point arithmetic. The noise samples have 11 fraction bits, hence the ulp is 2^{-11}. Over 95 percent of the samples are exactly rounded.

found to be accurate to 1 ulp. Throughout the tests, 95 percent of the samples are observed to be accurate to 1/2 ulp (i.e., exactly rounded), demonstrating the sheer quality of our noise samples. Fig. 12 shows a ulp error plot of 10,000 samples. We see that all samples are accurate to 1 ulp and most are accurate to 1/2 ulp.

Statistical tests, such as the χ^2 test or the Anderson-Darling test [6] are not necessary since 1) we know that the derivation of the original Box-Muller algorithm itself is correct and 2) we generate the samples accurately within the 16 bits resolution. Fig. 13 shows the PDF of our noise samples for a population of 10 million, while Fig. 14 shows the PDF between 7σ and 8.2σ for a population of 10,000. In both cases, our noise samples closely follow the true Gaussian PDF.

Various hardware implementations of our Box-Muller architecture are compared against several software implementations based on the Wallace [9], Ziggurat [7], polar, and Box-Muller methods [6], which are known to be the fastest methods for generating Gaussian noise on instruction processors. For the Wallace and Ziggurat methods, FastNorm3, available in [29], with maximum quality setting and rnorrexp available in [7] are used. The software implementations are run on an Intel Pentium-4 3 GHz PC and an AMD Athlon-64 3000+ 1.8 GHz PC,

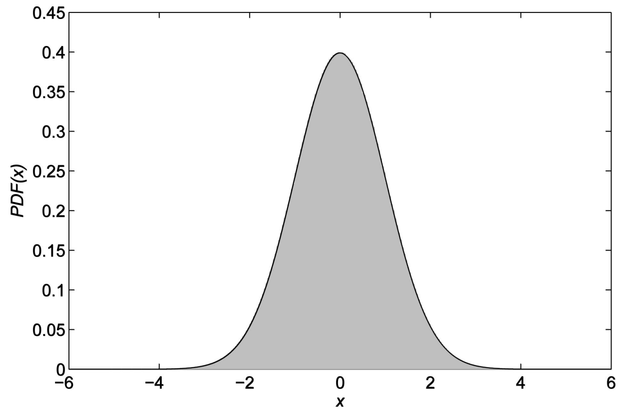

Fig. 13. PDF of the generated noise from our design for a population of 10 million samples. The black solid line indicates the ideal Gaussian PDF.

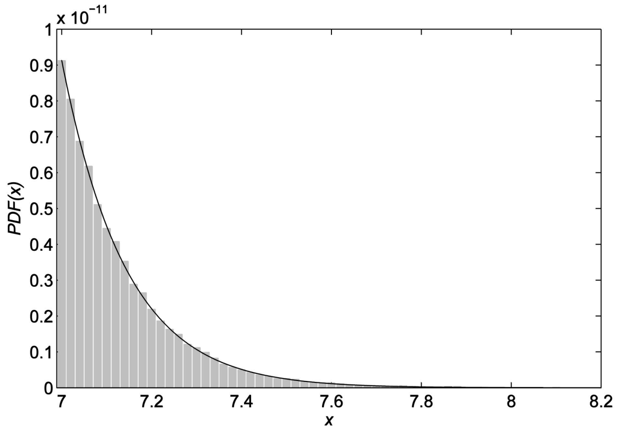

Fig. 14. PDF of the generated noise from our design for a population of 10,000 samples between 7σ and 8.2σ. The black solid line indicates the ideal Gaussian PDF.

both equipped with 2 GB DDR-SDRAM. They are written in ANSI C and compiled with the GNU gcc 3.3.3 compiler with -O3 optimization, generating IEEE double-precision floating-point numbers. In order to make a fair comparison, we use the Tausworthe URNG for all implementations. The Tausworthe URNG can generate 110 million 32-bit uniform random numbers per second on an Intel Pentium-4 3 GHz PC. The comparisons are shown in Table 1. We see that our hardware designs are faster than software implementations by 11-5,408 times, depending on the device used and the resource utilization. It is also important to note that the software implementations could certainly be made somewhat faster by utilizing assembler-level optimizations. That said, the FPGA implementations are approximately two to three orders of magnitude faster in terms of throughput than the software implementations

TABLE 1
Throughput Comparisons of Various Hardware and Software Gaussian Noise Generators

platform	speed [MHz]	method	throughput [million/sec]
XC2VP100-7 FPGA	95	37 inst	7030.0
XC4VLX100-12 FPGA	375	1 inst	750.0
XC2V4000-6 FPGA	233	1 inst	466.0
XC3S5000-5 FPGA	181	1 inst	362.0
Intel Pentium-4 PC	3000	Wallace	33.3
AMD Athlon-64 PC	1800	Wallace	30.3
Intel Pentium-4 PC	3000	Ziggurat	27.8
AMD Athlon-64 PC	1800	Ziggurat	26.7
Intel Pentium-4 PC	3000	Polar	9.1
AMD Athlon-64 PC	1800	Polar	7.7
Intel Pentium-4 PC	3000	Box-Muller	2.1
AMD Athlon-64 PC	1800	Box-Muller	1.3

The XC2VP100-7 FPGA belongs to the Xilinx Virtex-II Pro family, XC4VLX100-12 FPGA belongs to the Xilinx Virtex-4 LX family, the XC2V4000-6 belongs to the Xilinx Virtex-II family, while the XC3S5000-5 belongs to the low-cost Xilinx Spartan-III family.

TABLE 2
Comparisons of Different Hardware Gaussian Noise Generators Implemented on a Xilinx Virtex-II XC2V4000-6 FPGA

	Xilinx [17]	Zhang [16]	Lee [15]	Lee [14]	this work
method	Box-Muller & CLT	Ziggurat	Box-Muller & CLT	Wallace	Box-Muller
slices	653	891	2514	770	1528
block RAMs	4	4	2	6	3
MULT18X18s	8	2	8	4	12
clock speed [MHz]	168	170	133	155	233
samples / clock	1	0.993	1	1	2
million samples / sec	168	168	133	155	466
max σ	4.8	unknown	6.7	unknown	8.2
quality	low	high	high	high	very high
noise bit-width	16	32	32	24	16
ulp accuracy guarantee	no	no	no	no	yes

"CLT" refers to the central limit theorem.

and even the best assembler-level coding would leave a large performance gap. Thus, the message of Table 1 is that, as one would expect, the dedicated hardware implementation is significantly faster than what is achievable with programmable software platforms.

Table 2 shows comparisons of our design against four other designs: the Gaussian noise generator block available in Xilinx System Generator 7.1 [17], the Ziggurat design in [16], our previous Box-Muller design [15], and our Wallace design [14]. Note that the Xilinx block is based on Xilinx's AWGN core 1.0 architecture [12]. In order to make the comparisons fair, all designs are placed-and-routed on a Xilinx Virtex-II XC2V4000-6 FPGA and hand placement-and-routing is not performed. We observe that our design has the best noise quality, the maximum obtainable σ multiple, and the best throughput/area ratio. The clock speed of our design is higher than others due to more aggressive pipelining.

9 Conclusions

We have presented a hardware Gaussian noise generator using the Box-Muller method to aid simulations involving large numbers of samples. The architecture involves a series of elementary function evaluations, which are computed using fixed-point arithmetic. In order to obtain minimal signal bit-widths while respecting the accuracy requirements, we perform error analysis based on the MiniBit framework [25]. Two 16-bit noise samples are generated every clock and, due to the accurate error analysis, every sample is analytically guaranteed to be accurate to one unit in the last place. The noise generator accurately models the true Gaussian PDF out to 8.2σ.

The design has been realized in FPGA technology. An implementation on a Xilinx Virtex-4 XC4VLX100-12 FPGA occupies 1,452 slices, three block RAMs, and 12 DSP slices and is capable of generating 750 million samples per second at a clock speed of 375 MHz. Further performance improvements are obtained through exploiting parallelism: 37 instances of the noise generator on a Xilinx Virtex-II Pro XC2VP100-7 FPGA can generate seven billion noise samples per second, which is over 200 times faster than an Intel Pentium-4 3 GHz PC. The noise generator is currently being used at the Jet Propulsion Laboratory, NASA to evaluate the performance of low-density parity-check codes for deep-space communications.

Future work includes applying our design methodology to other distributions, including Cauchy, exponential, and Weibull. These can be generated by evaluating the inverse cumulative distribution function (CDF) of the distribution [8], which can be approximated via polynomials.

Acknowledgments

The authors thank Ray C.C. Cheung and Altaf Abdul Gaffar for their assistance. The support of the Jet Propulsion Laboratory, the US Office of Naval Research, Xilinx Inc., the UK Engineering and Physical Sciences Research Council (Grant number EP/C509625/1, EP/C549481/1, and GR/R 31409), and the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK4333/02E) is gratefully acknowledged.

References

[1] R. Gallager, "Low-Density Parity-Check Codes," *IEEE Trans. Information Theory,* vol. 8, pp. 21-28, 1962.
[2] B. Jung, H. Lenhof, P. Müller, and C. Rüb, "Langevin Dynamics Simulations of Macromolecules on Parallel Computers," *Macromolecular Theory and Simulations,* pp. 507-521, 1997.
[3] A. Brace, D. Gatarek, and M. Musiela, "The Market Model of Interest Rate Dynamics," *Math. Finance,* vol. 7, no. 2, pp. 127-155, 1997.
[4] C. Berrou, A. Glavieuxand, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes," *Proc. IEEE Int'l Conf. Comm.,* pp. 1064-1070, 1993.
[5] K. Tsoi, K. Leung, and P. Leong, "Compact FPGA-Based True and Pseudo Random Number Generators," *Proc. IEEE Symp. Field-Programmable Custom Computing Machines,* pp. 51-61, 2003.
[6] D. Knuth, *Seminumerical Algorithms,* third ed., vol. 2. Addison-Wesley, 1997.
[7] G. Marsaglia and W. Tsang, "The Ziggurat Method for Generating Random Variables," *J. Statistical Software,* vol. 5, no. 8, pp. 1-7, 2000.
[8] W. Hörmann and J. Leydold, "Continuous Random Variate Generation by Fast Numerical Inversion," *ACM Trans. Modeling and Computer Simulation,* vol. 13, no. 4, pp. 347-362, 2003.

[9] C. Wallace, "Fast Pseudorandom Generators for Normal and Exponential Variates," *ACM Trans. Math. Software,* vol. 22, no. 1, pp. 119-127, 1996.
[10] G. Box and M. Muller, "A Note on the Generation of Random Normal Deviates," *Annals Math. Statistics,* vol. 29, pp. 610-611, 1958.
[11] E. Boutillon, J. Danger, and A. Gazel, "Design of High Speed AWGN Communication Channel Emulator," *Analog Integrated Circuits and Signal Processing,* vol. 34, no. 2, pp. 133-142, 2003.
[12] *Additive White Gaussian Noise (AWGN) Core v1.0,* Xilinx Inc., 2002, http://www.xilinx.com.
[13] E. Fung, K. Leung, N. Parimi, M. Purnaprajna, and V. Gaudet, "ASIC Implementation of a High Speed WGNG for Communication Channel Emulation," *Proc. IEEE Workshop Signal Processing Systems,* pp. 304-409, 2004.
[14] D. Lee, W. Luk, G. Zhang, P. Leong, and J. Villasenor, "A Hardware Gaussian Noise Generator Using the Wallace Method," *IEEE Trans. VLSI Systems,* vol. 13, no. 8, pp. 911-920, 2005.
[15] D. Lee, W. Luk, J. Villasenor, and P. Cheung, "A Gaussian Noise Generator for Hardware-Based Simulations," *IEEE Trans. Computers,* vol. 53, no. 12, pp. 1523-1534, Dec. 2004.
[16] G. Zhang, P. Leong, D. Lee, J. Villasenor, R. Cheung, and W. Luk, "Ziggurat-Based Hardware Gaussian Random Number Generator," *Proc. IEEE Int'l Conf. Field-Programmable Logic and Its Applications,* pp. 275-280, 2005.
[17] *Xilinx System Generator User Guide v7.1,* Xilinx Inc., 2005, http://www.xilinx.com.
[18] R.C. Tausworthe, "Random Numbers Generated by Linear Recurrence Modulo Two," *Math. and Computation,* vol. 19, pp. 201-209, 1965.
[19] G. Marsaglia, *Diehard: A Battery of Tests of Randomness,* 1997, http://stat.fsu.edu/~geo/diehard.html.
[20] P. L'Ecuyer, "Maximally Equidistributed Combined Tausworthe Generators," *Math. Computation,* vol. 65, no. 213, pp. 203-213, 1996.
[21] V. Lefevre, J. Muller, and A. Tisserand, "Toward Correctly Rounded Transcendentals," *IEEE Trans. Computers,* vol. 47, no. 11, pp. 1235-1243, Nov. 1998.
[22] M.J. Schulte and E.E. Swartzlander Jr., "Hardware Designs for Exactly Rounded Elementary Functions," *IEEE Trans. Computers,* vol. 43, no. 8, pp. 964-973, Aug. 1994.
[23] J. Muller, *Elementary Functions: Algorithms and Implementation.* Birkhauser Verlag AG, 1997.
[24] J. Walther, "A Unified Algorithm for Elementary Functions," *Proc. AFIPS Spring Joint Computer Conf.,* pp. 379-385, 1971.
[25] D. Lee, A. Abdul Gaffar, O. Mencer, and W. Luk, "MiniBit: Bit-Width Optimization via Affine Arithmetic," *Proc. ACM/IEEE Design Automation Conf.,* pp. 837-840, 2005.
[26] L. de Figueiredo and J. Stolfi, "Self-Validated Numerical Methods and Applications," *Brazilian Math. Colloquium Monograph,* IMPA, Brazil, 1997.
[27] V. Oklobdzija, "An Algorithmic and Novel Design of a Leading Zero Detector Circuit: Comparison with Logic Synthesis," *IEEE Trans. VLSI Systems,* vol. 2, no. 1, pp. 124-128, 1994.
[28] M. Pillmeier, M. Schulte, and E. Walters III, "Design Alternatives for Barrel Shifters," *Proc. SPIE Advanced Signal Processing Algorithms, Architectures, and Implementations,* pp. 436-447, 2002.
[29] C. Wallace, "MDMC Software—Random Number Generators," 2003, http://www.datamining.monash.edu.au/software/random.

Dong-U Lee received the BEng degree in information systems engineering and the PhD degree in computing, both from Imperial College London, in 2001 and 2004, respectively. He is currently a postdoctoral researcher in 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. He is a member of the IEEE.

John D. Villasenor received the BS degree in 1985 from the University of Virginia, the MS degree in 1986 from Stanford University, and the PhD degree in 1989 from Stanford, all in electrical engineering. From 1990 to 1992, he was with the Radar Science and Engineering Section of the Jet Propulsion Laboratory in Pasadena, California, where he developed methods for imaging the earth from space. He joined the Electrical Engineering Department at the University of California, Los Angeles (UCLA) in 1992 and is currently a professor. He served as vice chair of the department from 1996 to 2002. At UCLA, his research efforts lie in communications, computing, imaging and video compression, and networking. He is a senior member of the IEEE.

Wayne Luk received the MA, MSc, and DPhil degrees in engineering and computer science from the University of Oxford, Oxford, United Kingdom. He is a professor of computer engineering, Department of Computing, Imperial College London and leads the Custom Computing Group there. 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. He is a member of the IEEE.

Philip H.W. Leong received the BSc, BE, and PhD degrees from the University of Sydney in 1986, 1988, and 1993, respectively. In 1989, he was a research engineer at AWA Research Laboratory, Sydney, Australia. From 1990 to 1993, he was a postgraduate student and research assistant at the University of Sydney, where he worked on low-power analogue VLSI circuits for arrhythmia classification. In 1993, he was a consultant to SGS Thomson Microelectronics in Milan, Italy. He was a lecturer in the Department of Electrical Engineering, University of Sydney from 1994-1996. He is currently a professor in the Department of Computer Science and Engineering at the Chinese University of Hong Kong and the director of the Custom Computing Laboratory. He is the author of more than 70 technical papers and five patents. His research interests include reconfigurable computing, digital systems, parallel computing, cryptography, and signal processing. He is a senior member of the IEEE.