An Efficient Hardware Implementation of High Quality AWGN Generator Using Box-Muller Method

Jamshaid Sarwar Malik
School of ICT
Royal Institute of
Technology, Sweden
jamsjaid@kth.se

Jameel Nawaz Malik
National University of
Sciences and Technology
SEECS, Pakistan
muhammad.jameel@seecs.edu.pk

Ahmed Hemani
School of ICT
Royal Institute of
Technology, Sweden
hemani@kth.se

N. D. Gohar
National University of
Sciences and Technology
SEECS, Pakistan
nasir.gohar@seecs.edu.pk

***Abstract*—Box Muller (BM) algorithm is extensively used for generation of high quality Gaussian Random Numbers (GRNs) in hardware. Most efficient published implementation of BM method utilizes transformation of 32-bit data path to 16 bits and use of first degree piece-wise polynomial approximation to compute logarithmic and square root functions. In this work, we have performed extensive error analysis to show that coefficient memory for polynomial approximation can be reduced by more than 35 percent without compromising on quality of generated Gaussian samples. This also reduces complexity of corresponding address generator, which requires most hardware resources. We have also used more efficient and statistically accurate skip-ahead Linear Feedback Shift Registers to generate uniformly distributed numbers for the BM algorithm. Complete hardware implementation utilizes only 407 slices, 03 DSP blocks and 1.5 memory blocks on Xilinx Virtex-4 XC4VLX15 operating at 230 MHz while providing a tail accuracy of 6.6σ. This is better in terms of accuracy and hardware utilization than any of the previously reported architecture.**

Index Terms*—*Random Number Generator, Gaussian, Normal, AWGN, Box Muller Algorithm

I. Introduction

Fast, compact and high quality Gaussian Random Number (GRN) generation is a key capability for simulations across a wide range of disciplines. For example, channel code evaluation, Monte-Carlo (MC) simulations, financial modeling, molecular dynamics simulation and product failure simulations etc. Most of the processes in nature are Gaussian distributed since they tend to be a balanced sum of many unobserved random events and by virtue of Central Limit Theorem [1], sum of sufficiently large random numbers tend to become Gaussian distributed. In simulations, where probability of occurrence of an event is very low, high tail accuracy GRNs are required. One such example is BER simulation of wireless radio standards, some of which have maximum allowable BER of less than 10^{10} for specified SNRs. Other important examples are turbo codes and Low-Density Parity-Check (LDPC) codes which are currently the focus of an intense research due to their ability to approach Shannon bound very closely and with only moderate decoding complexity [2].

Simulating occurrence of such events, requires a tail accuracy of at least 6σ. It takes about 2.5 hours to generate 10^9 Gaussian samples on a modern core i7 machine using an optimized software such as Matlab [3]. This means that it will take more than a hundred days to generate 10^{12} Gaussian samples in order to exhibit a tail accuracy of 6σ. Clearly, software simulations cannot provide an adequate solution to examine the behavior of such codes at very low BER.

In this work, we have improved an existing hardware implementation [3] of BM algorithm, thereby reducing both logic and memory. Key contributions of this work include:

- An extensive error analysis of implementation of [3] that results in reduction of memory by more than 35 percent and a less complicated address generator.
- Introduction of skip-ahead LFSRs [4] instead of Combined Tausworthe Generators (CTGs) [3] which provides uncorrelated uniformly distributed pseudo random numbers at a lower hardware cost.
- Real-time hardware implementation of BM algorithm on Xilinx Virtex-4 XC4VLX15 FPGA which is better in terms of logic, speed and performance than any of the previously reported implementation (Section-III, Table-IV).

A. Paper Layout

This paper has been divided into four Sections. Section-I provides a brief introduction, followed by an overview of the related published literature to date. Section-II discusses the Box Muller Algorithm, its hardware implementation and an extensive error analysis that results in a reduced hardware cost while not compromising on accuracy. In Section-III, we have provided statistical accuracy of the GRNs produced by our proposed design, its synthesis results and a comparison with previously reported designs. Section-V provides conclusions and suggests some further improvement possibilities.

B. Related Prior Work

Gaussian Random Numbers (GRNs) are usually generated by use of transformation of uniformly distributed numbers [1]. Ziggurat algorithm [6] and Polar method [7] fall under the class of rejection-acceptance methods. These algorithms share a common disadvantage that their output rate is not constant due to conditional *if-then-else* statements. Therefore, these methods are not considered well suited for efficient hardware implementations of GRN generators. However, in [6], the authors present the hardware implementation of Ziggurat method with a constant output rate that involves the computation of some complex functions such as natural logarithm and exponential function. Also, the Polar method in [7] gives GRNs at constant output rate using first-in and first-out (FIFO) buffer.

The inversion method includes CDF inversion that transforms uniform random numbers to generate Gaussian samples by approximating the Inverse Gaussian Cumulative Distribution Function (IGCDF). In [8], an efficient hardware implementation of inversion method is proposed that employs non-uniform segmentation scheme to accurately approximate the IGCDF, especially in low probability region.

Wallace method belongs to the class of recursion methods [9]. The algorithm applies linear transformations to a pool of previously generated Gaussian samples to generate new Gaussian outputs. The Wallace method suffers from correlations between successive samples due to inherent feedback path in its architecture [1]. In [9], hardware implementation of Wallace method is proposed that minimizes correlation effects by proper parameter selection.

Boutillon, in [10], proposed the hardware implementation of GRN generator based on Box-Muller (BM) method. The Central Limit theorem (CLT) is employed in the given technique to minimize the approximation errors that occur due to evaluation of mathematical functions involved in BM method.

In [11], a more efficient hardware implementation of Box-Muller method is reported that uses degree one piecewise polynomial approximation to the mathematical functions involved in BM Method. Also, a non-uniform segmentation scheme is introduced that minimizes the approximation errors in evaluating the mathematical functions. To further reduce the approximation and quantization errors, CLT is used to improve the noise quality. The provided output rate is one sample per clock cycle and highest attainable tail accuracy is 6.6σ.

In [12], Dong-U Lee presented an accurate error analysis and bit width optimization techniques for evaluating mathematical functions used in Box-Muller method. The provided output rate is two samples per clock cycle as CLT is no longer used. The highest attainable standard deviation value is 8.2σ. The proposed method is efficient both in terms of hardware resources and throughput compared to previously reported hardware implementations of BM Method.

The authors in [3] presented a fast and compact Gaussian noise generator based on Box-Muller method. The work provides the most efficient hardware implementation of BM Method and maximum attainable standard deviation values larger than previously published designs. The algorithm uses polynomial curve fitting with hybrid segmentation and scaling scheme to approximate the mathematical functions involved in Box-Muller method. Xilinx [13] had released an IP core based on Box-Muller algorithm. ASIC chip for Box-Muller Method is given by Fung[14].

II. Hardware Implementation of BM Algorithm

In this section we will provide a detailed error analysis of the hardware implementation of BM algorithm.

A. The Box Muller Algorithm

Proposed by George Edward Pelham Box and Mervin Edgar Muller in 1958 [15], BM algorithm is a transformation of two Uniformly distributed random numbers to generate a pair of independent Gaussian distributed random numbers. Suppose U_1 and U_2 are independent random variables that are uniformly distributed in the interval $(0, 1)$. BM transformation involves two new variables Z_1 and Z_2 which are Gaussian distributed with unit variance such that:

$$Z_1 = \sqrt{-2lnU_1}.sin(2\pi U_2) = f(.)sin(2\pi U_2) \tag{1}$$

$$Z_2 = \sqrt{-2lnU_1}.cos(2\pi U_2) = f(.)cos(2\pi U_2) \tag{2}$$

where $f(.) = \sqrt{-2lnU_1}$. There are three advantages of BM transform that makes it suitable for hardware implementation:

- It involves direct conversion of 2 variables to 2 new variables. Unlike *Inversion*, *Wallace* and *CLT* methods, no memory is needed to store previous samples.
- Data path is fixed with no conditional statements. Hence, a constant output rate is maintained unlike *Ziggurat* method.
- Unlike *CLT* and *CDF Inversion* methods, BM is an *exact* method which means that it is analytically possible to guarantee an ideal Gaussian distribution of the output.

Consequently, the BM transformation have been reported in a number of publications on many GRN hardware implementations [3],[11],[12].

B. Error in approximated $f_{(.)}$ function

To date, most efficient hardware implementation of BM algorithm has been reported in [3]. To provide a tail accuracy of 6.6σ, U_1 has to be at least 32 bit wide [8], which translates to a data path of same width. However, [3] provides a methodology to transform U_1 and corresponding data path to 16 bits without loosing accuracy of the $f_{(.)}$ function. For the sake of completion, the methodology in [3] is briefly explained here:

$f_{(.)}$ function is implemented using a hybrid segmentation scheme with both logarithmic and uniform segmentations. First, the domain $(0, 1)$ of U_1 is divided into two subintervals $r_0 \in (0, 0.5)$ and $r_1 \in (0.5, 1)$ with 31 non-uniform segments $s_{r,w}$ where $r \in (r_0, r_1)$ and $w \in (0, 1, 2, ..., 30)$. Then, each segment is subdivided uniformly into $L = 2^l$ subsegments, where each subsegment is denoted by $s_{r,w,l}$. This is shown in Figure-1

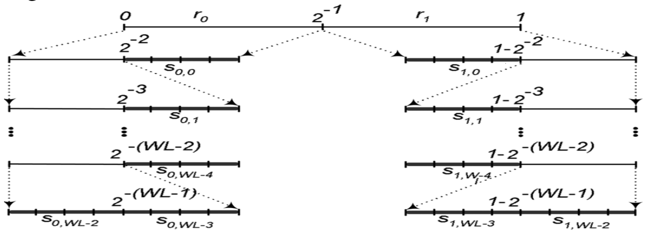

Fig. 1. Hybrid (logarithmic and uniform) segmentation (Courtesy [3])

After segmentation, the function $f_{(.)}$ is approximated using first order polynomial approximation. An important point to note is that as U_1 approaches 0 or 1, the slope tends toward infinity and, therefore, the coefficients become very large. To avoid this, U_1 and the polynomial coefficients are transformed from 32 to 16 bit using following relations:

When U_1 lies in the region $(0, 0.5)$

$$\begin{aligned} U_{1_{tr}} &= 2^w U_1 \\ a_{tr} &= 2^{-w} a \\ b_{tr} &= b \end{aligned} \tag{3}$$

Where a and b are the coefficients of the first order polynomial. a_{tr}, b_tr and U_tr are 16-bit transformed variables for a, b and U respectively. Similarly, when U_1 lies in the region $(0.5, 1)$:

$$\begin{aligned} U_{1_{tr}} &= 2^w(1 - U_1) \\ a_{tr} &= -2^{-w} a \\ b_{tr} &= b + a \end{aligned} \tag{4}$$

These transformations result in a data path where the input ($U_{1_{tr}}$) and polynomial coefficients (a_{tr} and b_{tr}) are all 16-bit wide, thereby drastically reducing the computational hardware complexity.

C. *Reducing error in approximated $f_{(.)}$ function*

It is apparent that increasing the number of sub-segments L will improve accuracy of the approximated function. This is shown in Figure-2 where approximation error is plotted for $L = 4, 8$ and 16 respectively.

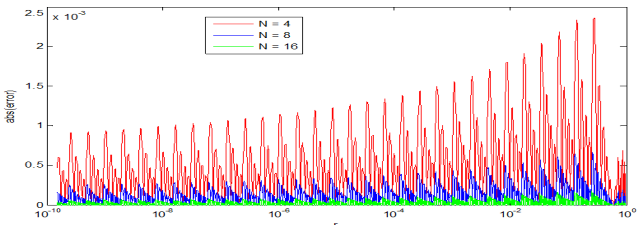

Fig. 2. Approximation error for $L = 4, 8$ and 16

This improvement, however, comes at that cost of increased coefficient memory. Table-1 shows total coefficient memory requirement for different values of L.

The address generator for addressing the non-uniform segments $s_{r,w}$ is is a leading ones detector (LOD) circuit as shown in Figure-3 [12]. The summation block at the bottom

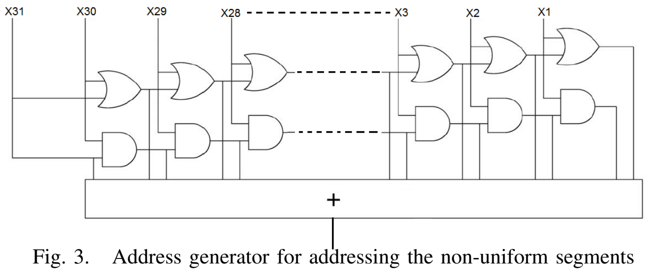

Fig. 3. Address generator for addressing the non-uniform segments

of address generator provides address of 62 non-uniform segments. It can be observed that the complexity of the address generator can be reduced by reducing total number of non-uniform segments. In [3], it is shown that for $L >= 8$, the degradation in the accuracy of $f_{(.)}$ function is negligible. In the following text, however, we will show that using a few optimizations, it is possible to reduce the peak error for $L = 4$ by 65 percent, thereby making it approximately equal to the case where $L = 8$.

Optimization-1:

A tail accuracy of $\pm 8\sigma$ can be exhibited with a fixed point format of 5.11 (05 whole bits and 11 fraction bits). This corresponds to maximum quantization error of 2^{-11} or $MQE = 4.88e - 4$. Analyzing the function $f_{(.)}$, we can see that as U_1 approaches one, $f_{(.)}$ becomes close to zero. In fact, we can prove that when $w >= 15$ in the region r_1, approximation error is less than MQE. Hence, we dont need to go beyond level-15 when r goes from 0.5 to 1. In other words we can straightaway save 25 percent coefficient memory for a word length of 32-bit by reducing the required segments from 62 to 48. It also results in reducing the complexity of corresponding address generator (Figure-3).

Optimization-2:

Spacing between sub-segments in [3] have been kept equal. Since $f_{(.)}$ is a non-linear function, this equal spacing causes varying approximation error for different sub-segments which consequently results in large peak error. It is possible to choose the spacing between sub-segments in such a way that approximation error is distributed evenly between all the sub-segments. This is shown in Figure-4 where we have reduced the peak approximation error by 30 percent by *tuning* the spacing between sub-segments.

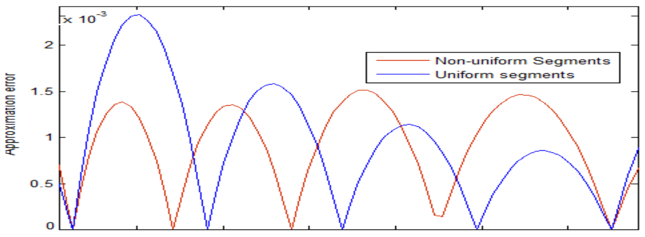

Fig. 4. Peak error reduction with non-uniform segmentation

Optimization-3:

Figure-5 shows the approximation error curve for $L = 4$. We can see the error grows in positive direction in region $r_0 \in (0, 0.5)$ and and in negative direction for $r_1 \in (0.5, 1)$. Hence, by changing second polynomial coefficient b (offset) we can *force* the error to grow both in positive and negative direction, thereby reducing both maximum and average approximation error.

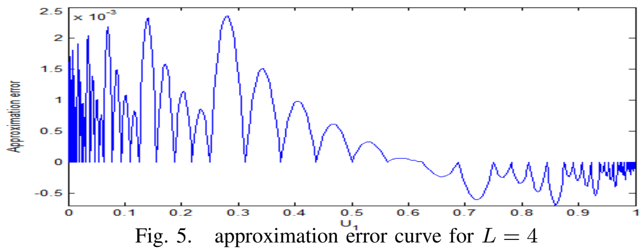

Fig. 5. approximation error curve for $L = 4$

Figure-6 and Table-1 summarizes the results of above mentioned optimizations. Finally, $f_{(.)}$ can be approximated by only 48 segments and 04 subsegments with a maximum approximation error of 7.54e-4 and mean error of 3.21e-4. This is significantly less than MQE (4.88e-4). Hence, the coefficient memory has been reduced from *996 x 16* to *384 x 16* (more than 60 percent) with negligible degradation in approximation error.

D. *Pseudo-random Number Generators*

Pseudo-Random Number (PRN) Generators are required to produce uniformly distributed U_1 and U_2 inputs. In simplest

TABLE I
REQUIRED COEFFICIENT MEMORY FOR CHANGING L

L	Max. Error	Mean. Error	Memory req. x16
16	2.35e-4	1.419e-5	1992
8	6.22e-4	6.40e-5	996
4	2.36e-3	2.65e-4	498
4 (optimized)	7.54e-4	6.03e-6	384

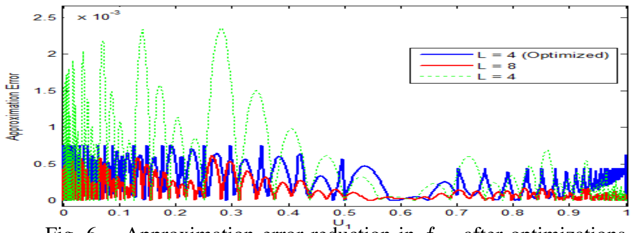

Fig. 6. Approximation error reduction in $f_{(.)}$ after optimizations

form, Linear Feedback Shift Registers (LFSRs) can be used for this purpose. However, these LFSRs suffer from unwanted correlations in adjacent samples which results in a low-pass filter like spectrum. This can also be seen as a lattice structure in 2-D scatter plot of the LFSR output. This problem is usually addressed by use of Combined Tausworthe Generators (CTGs) [3] which require more hardware resources as compared to LFSRs.

In this work, we have used skip-ahead LFSRs that follow the algorithm presented by Leonard Colvito in [4]. This technique uses a transition matrix A^k which can advance the LFSR to k steps ahead. An LFSR can be represented by:

$$q^{t+k} = A^k.q^t \tag{5}$$

where

$$A = \begin{bmatrix} a_{n-1} & a_{n-2} & \dots & a_1 & a_0 \\ 1 & 0 & & & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix}$$

and

$$q = \begin{bmatrix} q_{n-1} \\ q_{n-2} \\ \vdots \\ q_1 \\ q_0 \end{bmatrix}$$

In our design, we advanced the LFSR by $k = 16$ to ensure that successive samples are not correlated. While being much more hardware efficient as compared to CTGs, these modified LFSRs do not suffer from correlations since they skip n number of bits at every sample. As a result, there are no correlations in adjacent samples of the output.

Another advantage of skip-ahead LFSRs is high period length ρ with minimal hardware resources. It is generally recommended to have ρ to be much larger than the number of random numbers required by the simulations [3]. CTGs require huge hardware resources when ρ is increased. For example, a CTG implemented in [3] with $\rho = 2^{133}$ requires 114 slices. In contrast, we designed two skip-ahead LFSR with $\rho = 2^{127}$ and $\rho = 2^{128}$ respectively where each module takes only 33 slices. This provides an effective period length (ρ_{eff}) of 2^{255} (Section-III, Table-IV).

E. Implementation of Trigonometric functions

$sin(2\pi U_2)$ and $cos(2\pi U_2)$ functions can be implemented in a variety of ways including polynomial approximation, Look-up-tables (LUTs) and CORDIC algorithm [12]. We have used

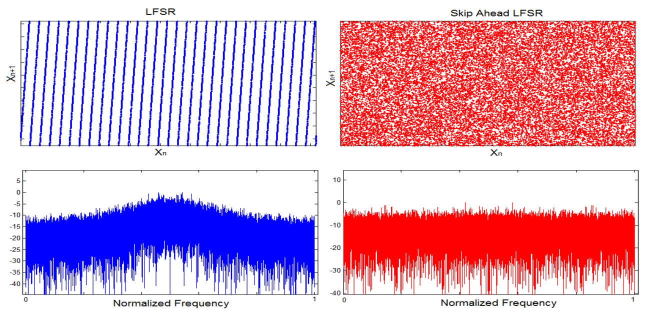

Fig. 7. 2-D Scatter and Spectral plots of LFSR and Skip Ahead LFSR

the implementation used in [3] which utilizes LUTs. Both functions can be implemented with a precision of 12-bits by using a 1024 words dual port LUT, which stores precomputed values for a quarter cycle of $sine$ function in fixed point format of 1.15. Symmetry between the sine and cosine functions is then exploited to compute both the functions simultaneously. The scheme results in a quantization error of 2^{-11} in both sine and cosine functions. Figure-8 shows the plot, maximum and mean absolute error in $sin(2\pi U_2)$ when a 1024 words LUT is used.

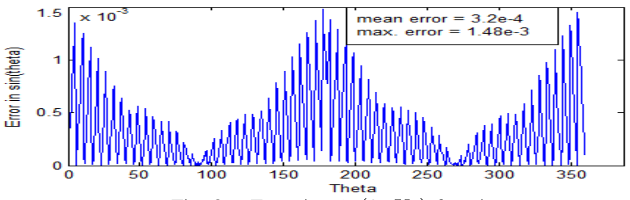

Fig. 8. Error in $sin(2\pi U_2)$ function

F. Error analysis of implemented BM algorithm

Equations 1 and 2 show that transformed Gaussian variables Z_1 and Z_2 are simultaneously dependent on $f_{(.)}$ and trigonometric functions. Hence, any error in these functions will translate to an error in the transformed variables Z_1 and Z_2. In following text, we will first analyze, and then quantify these errors. Due to limited space we will only analyze error in Equation-1 and skip Equation-2 which has similar error due to symmetry between sine and cosine functions.

For the sake of simplicity, lets represent $sin(2\pi U_2)$ and $cos(2\pi U_2)$ by $g_{(.)}$. If ξ_f, ξ_g and ξ_{Z_1} are the errors in functions $f_{(.)}$, $g_{(.)}$ and Z_1 respectively then:

$$Z_1 + \xi_{Z_1} = (f_{(.)} + \xi_f)(g_{(.)} + \xi_g) \tag{6}$$

From Equations 1 and 6, we can extract the expression for the error in Z_1 as

$$\xi_{Z_1} = (f_{(.)}\xi_g) + (g_{(.)}\xi_f) + (\xi_f\xi_g) \tag{7}$$

Maximum and mean values of $f_{(.)}$ are 6.6 and 2.254, while for $g_{(.)}$ these values are 1 and 0 respectively. Maximum and mean value of ξ_f and ξ_g have already been calculated in Table-I and Figure-8. Using equation-7, we can compute mean and maximum error in Z_1 for different values of L, This is summarized in Table-II. Figure-9 provides a better visual perception of these values by graphically plotting the

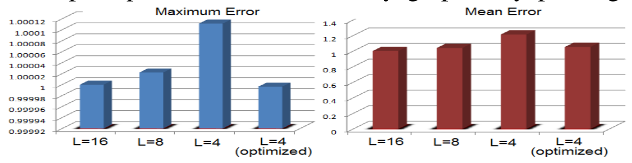

Fig. 9. Normalized Mean and Maximum error in Z_1

TABLE II

MAXIMUM AND MEAN ERROR IN U_1

L	Max. Error	Mean. Error
16	0.010004	7.212845e-4
8	0.010390	7.213004e-4
4	0.01212	7.213648e-4
4 (optimized)	0.010523	7.212819e-4

G. 3-D Error plots of implemented BM algorithm

Figures 6 and 8 provide error due to individual $f_{(.)}$ and $g_{(.)}$ functions. Overall error in Z_1 and Z_2 is, however, a product of these two errors as predicted by Equation-7. In order to find this overall error, we swept the values of U_1 and U_2 from 0 to 1 and 0 to 2π respectively and compared the error in Z_1 and Z_2 with the Matlab double precision floating point.

Figure-10 shows the 3-D error plots of Z_1 and Z_2 with $L = 8$ and $L = 4_{(optimized)}$ respectively. It can be seen that the maximum and mean error in all the four cases is very closed to the predicted values in Table-II. Also a keen observer may note that maximum error occurs when $U_1 \Rightarrow 0$. This is because according to Equation-7, maximum error is a function of $f_{(.)}$ which gives highest value (6.66) when U_1 is minimum.

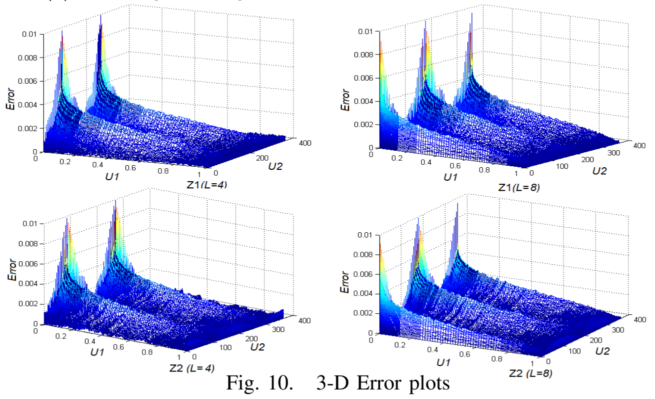

Fig. 10. 3-D Error plots

III. RESULTS

In Section-II-C, we showed that Maximum Quantization Error (MQE) for the fixed point format of Z_1 and Z_2 is $4.88e-4$. This is less than both maximum and mean error in Table-II and Figure-10. Hence, the implementation of BM algorithm does not analytically guarantee arbitrarily accurate Gaussian numbers. Therefore, it is necessary to apply standard tests to measure statistical accuracy of the generated numbers. These tests include Probability Density Function (PDF) plots, Chi Square tests, Scatter plots and Autocorrelation function.

A. PDF plots

At least 10^{12} samples are required to observe tail accuracy till 6.6σ [2]. At such high values of σ, the probability is so low ($< 10^{-9}$) that it is not observable at linear scale. Using the architecture described in Section-III, we generated 10^{12} random number samples. Since, it is not possible to store such a huge number of samples in memory, we built a simple framework in FPGA that divides all possible values Z_1 and Z_2 into 64 equal bins. All the bins were initialized with a zero value. Whenever, a GRN is generated, the framework computes the address of bin belonging to generated number. This bin is then incremented by one. Running at 230 MHz, it took approximately 70 minutes to compute PDF of 10^{12} samples . Figure-11 shows PDF plots of the generated GRNs on logarithmic scale as compared to ideal Gaussian PDF .

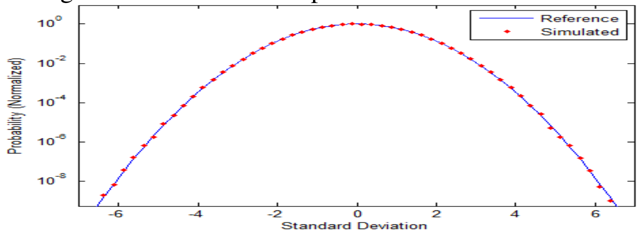

Fig. 11. PDF plot of 10^{12} generated samples (U_1)

B. Chi-Square Tests

Pearson's chi-square test, also known as the chi-square goodness-of-fit test for independence, is a statistical hypothesis test in which the sampling distribution is a Chi-Square [15] when the null hypothesis is true. A set of observed samples (observed frequencies) is compared against the expected distribution (expected frequencies). The test is performed by dividing a given sample of random numbers data into bins. Using more bins gives higher resolution with respect to the different input values, but reduces the expected number in each bin [16]. For each bin, observed and expected counts are calculated. The Chi-Square test is then computed using the formula:

$$\chi^2_{\alpha,\gamma-1} = \sum_{i=1}^{\gamma} \frac{[O(i) - E(i)]^2}{E(i)} \quad (8)$$

Where $O(.)$ is the observed counts, $E(.)$ is the expected count according to normal distribution [18], α is the desired significance level, γ is total number of bins in which the distribution is divided into and $\gamma - 1$' is the degree of freedom. Since the normal distribution is completely characterized by two parameters, the mean and the standard deviation, the degree of freedom is thus reduced by 2 from $\gamma - 1$ to $\gamma - 3$.

To perform chi-square test, the horizontal axis (σ axis) is divided into three regions from 0σ to 6.6σ as shown in Table-III and each interval is segmented into 100, 50 and 12 bins respectively. To eliminate statistical inaccuracy, it has been ensured that at least 50 samples fall into last bin. Chi square test is considered to *pass* for a given value of α if the observed value (calculated by Equation-8) is less than its corresponding theoretical value. Multiple tests with different seed value of LFSRs were performed to validate the consistency of the results.

TABLE III

CHI-SQUARE TEST RESULTS

Range	χ^2_{obs}	$\alpha = 0.1$	$\alpha = .05$	$\alpha = .01$
$0 \leq \sigma \leq 3.0$ *(100-bins)*	57	114 *(pass)*	121 *(pass)*	132 *(pass)*
$3.0 \leq \sigma \leq 5.0$ *(50-bins)*	24	60 *(pass)*	64 *(pass)*	73 *(pass)*
$5.0 \leq \sigma \leq 6.6$ *(12-bins)*	15	16 *(pass)*	18 *(pass)*	23 *(pass)*

The results obtained, as shown in Table-III, indicate that the proposed GRN generator successfully passes the chi-square test over the entire range of $0 - 6.6\sigma$ within 10 percent significance level (α).

C. Scatter plot and Autocorrelation function

Any correlation between neighboring numbers can be seen as a regular lattice structure in the scatter plot [3]. This is an unwanted property in any random number sequence. Right side of Figure-12 shows a $2-D$ scatter plot of the generated GRNs with no visible lattice structure. This can also be seen in left side of Figure-12 which shows the Autocorrelation function over a range of $\pm 10^5$ lags. Correlation for all non-zero lags is extremely low.

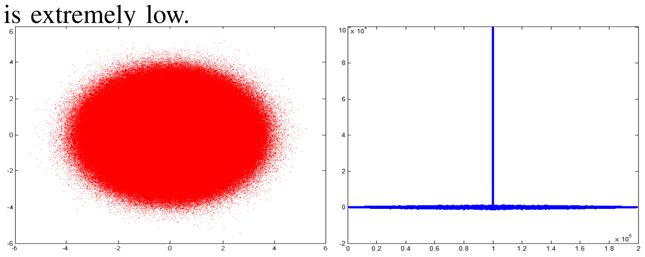

Fig. 12. Scatter plot autocorrelation function of generated GRNs

D. Synthesis Results and Comparison with Prior Work

The architecture with optimizations described in Section-II was implemented in VHDL. Top level schematics is shown in Figure-13. The design was synthesized using a relatively older FPGA family (Xilinx XC4VLX15 Virtex-4 device). This is because later versions of FPGAs (for example Virtex-5 and Virtex-6) are faster and have individual logic slices with more capacity. Hence, for the sake of fair comparison with the previously reported work, we used FPGA from an older family. Table-IV summarizes the comparison of synthesized

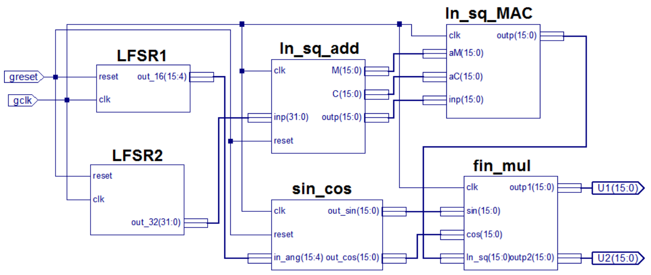

Fig. 13. Top-level schematics of the implemented GRN generator

architecture with previously reported work. The design required only 407 Slices, 3 DSP blocks and 1.5 block RAM while operating at a frequency of 230 MHz. We can see that resource utilization is better than any of the previously reported hardware implementation of GRN generator.

TABLE IV
COMPARISON WITH PUBLISHED WORK

Design	**[10]**	**[13]**	**[3]**	**[12]**	**[9]**	**[6]**	**This work**
Method Used	BM	BM	BM	BM	Wlc.	Zgrt.	**CLT**
Logic Cells	437	480	534	1528	770	891	**407**
Memory Blocks	0.5	5	2	3	6	2	**1.5**
Multi-pliers	N/A	5	3	12	4	2	**3**
GRN Bitwidth	12	16	16	16	24	32	**16**
Repetition period	2^{18}	2^{190}	2^{88}	2^{64}	2^{32}	2^{88}	2^{255}
Tail Accuracy	4σ	4.8σ	6.6σ	8σ	7σ	N/A	6.6σ
Speed (M samples/s)	25	245	440	468	155	168	**460**

IV. Conclusion and Future Work

In this work, we have improved the previously reported best hardware implementation of BM algorithm in terms of speed, logic and memory. We used extensive error analysis to show that coefficient memory for polynomial approximation can be reduce by more than 35 percent without compromising on quality of generated numbers. We used more efficient and statistically accurate skip-ahead Linear Feedback Shift Registers (LFSRs) to generate uniformly distributed numbers for the BM algorithm. All these optimizations resulted in a hardware implementation which is better in terms of accuracy and hardware utilization than any of the published architecture.

Proposed architecture can further be improved in two dimensions. First, in current scheme, the output is not analytically guaranteed to be correct since maximum error in generated GRNs is greater than one LSB (Equation - 7). LUT memory of both $f_{(.)}$ and $g_{(.)}$ has to be increased so that maximum error is less than one LSB. Secondly, limitation in tail accuracy is due to size of LFSR-2 (32-bits) [8]. A 64-bit LFSR will double the complexity of address generator and corresponding LUT memory while providing a tail accuracy of 9.4σ.

Acknowledgment

- The authors of this work are extremely gratified to ICT R and D Fund, Pakistan, for their breed financial support.

References

[1] Thomas D, Luk. W, Leong P, Villasenor J, *" Gaussian random number generators* ACM Comput. Surv. 39, 4, Article 11 (October 2007)

[2] D. Lee, J. Villasenor, W. Luk, P. Leong , *"A hardware Gaussian noise generator using the Box-Muller method and its error analysis"* , IEEE Trans. Comput., vol. 55, no. 6, Jun. 2006.

[3] A Alimohammad, Saeed F. Fard, Bruce F. Cockburn, Christian Schlegel, *"A Compact and Accurate Gaussian Variate Generator"*, IEEE Transactions on VLSI Systems, Vol. 16, No. 5, MAY 2008.

[4] L. Colavito and D. Silage, *"Efficient PGA LFSR Implementation Whitens Pseudo random Numbers"*, ICRCF, 2009.

[5] M.A Shami, A Hemani, *"Partially Reconfigurable Interconnection Network for Dynamically Reprogrammable Resource Array"*, ASICON 2009.

[6] G. Zhang, W. Leong, D. Lee, J. D. Villasenor, R.Cheung, W. Luk, *"Ziggurat-based hardware Gaussian random number generator "*, Proc. IEEE Int. Conf. Field Program. Logic It's Appl, 2005.

[7] Y. Fan, Z. Zilic, M. W. Chiang, *"A versatile high speed bit error rate testing scheme "*, in Proc. IEEE Int. Symp. Quality Electron. Des., 2004.

[8] D. Lee, R. Cheung, J. Villasenor, W. Luk, *"Inversion based hardware Gaussian random number generator: A case study of function evaluation via hierarchical segmentation "*in Proc. IEEE Int.Conf. Field-Program. Technol., 2006.

[9] D.-U. Lee et al.,*"A hardware Gaussian noise generator using the Wallace method "*, IEEE Trans. Very Large Scale Integr. (VLSI) Syst., Aug. 2005.

[10] E. Boutillon, J. Danger, A. Gazel, *"Design of High Speed AWGN Communication Channel Emulator"*, AICSP, 2003.

[11] D. Lee, W. Luk, J. Villasenor, P. Cheung, *"A Gaussian Noise Generator for Hardware-Based Simulations"* IEEE Trans. Computers, Dec. 2004.

[12] D. Lee, J. Villasenor, W. Luk, P. Leong , *"A hardware Gaussian noise generator using the Box-Muller method and its error analysis"* , IEEE Trans. Comput, Jun. 2006.

[13] *"Additive White Gaussian Noise (AWGN) Core"*, v1.0, Xilinx Inc., 2002.

[14] E. Fung, K. Leung, N. Parimi, M. Purnaprajna, V. Gaudet, *"ASIC Implementation of a High Speed WGNG for Communication Channel Emulation "*, Proc. IEEE Workshop Signal Processing Systems, pp. 304-409, 2004.

[15] M. E. Muller,*"A comparison of methods for generating normal deviates on digital computers"*, J. ACM, vol. 6, no. 3, pp. 376-383, 1959.

[16] Ralph B. D'Agostino , Michael A. Stephens, *"Goodness-of-fit-techniques (Statistics: a Series of Textbooks and Monographs, Vol. 68)"*, New York: Marcel Dekker, 1986.