FPGA Gaussian Random Number Generator Based on Quintic Hermite Interpolation Inversion

Pedro Echeverría, Marisa López-Vallejo
Department of Electronic Engineering
Universidad Politécnica de Madrid (Spain)
Email: {petxebe, marisa}@die.upm.es

***Abstract*— In this work we present a very accurate floating point FPGA implementation of a Gaussian random number generator (GRNG) based on the inversion method. The inverse Gaussian cumulative distribution function ($GCDF^{-1}$) is approximated using a quintic degree segment interpolation with Hermite coefficients and an accuracy-adaptative segmentation which divides the $GCDF^{-1}$ into several non-uniform segments. Our architecture generates simple floating point samples of 32 bits with an accuracy of 20 bits of mantissa, achieving a 185 MHz speed and a troughput of one sample per cycle on a Xilinx Virtex-II FPGA.**

I. Introduction

Many computationally intensive simulations require of a high quality Gaussian Random Number Generator (GRNG) such as physics, Monte Carlo or communication systems simulations. The software complexity of this type of generators makes them ideal candidates for hardware acceleration, taking advantage of the excellent performance of nowadays FPGAs.

There are several implementations of GRNG on FPGA which differ mainly on the generation method selected: Ziggurat [1] Wallace [2] or Box-Muller [3], [4]. Unlikely, none of these generators is compatible with some variance reduction techniques [5] used in very intensive simulations like stratified sampling or latin hypercube. Fortunately there is another generation method which is compatible with these techniques, the inversion method [6], as it transforms sequences of uniform random numbers into gaussian random numbers keeping the structural properties of the uniform sequence.

The inversion method is a general method to generate any probability distribution using the inverse function of the corresponding cumulative distribution function (CDF) and uniform variables. These variables correspond to values of the cumulative probability (axis y in the GCDF graphic in Figure 1). Then the variables of the desired distribution are obtained calculating the x values that generate that probability with the inverse function.

Direct implementation of the Gaussian CDF^{-1} function on a hardware platform is unworkable due to its great complexity which involves logarithms, exponentials and square roots [7]. This way, the inverse function has to be approximated with numerical algorithms. One type of algorithms that suits hardware constraints is the splines approximation, where the function range is divided into segments and, in each segment, the function is approximated by a polynomial.

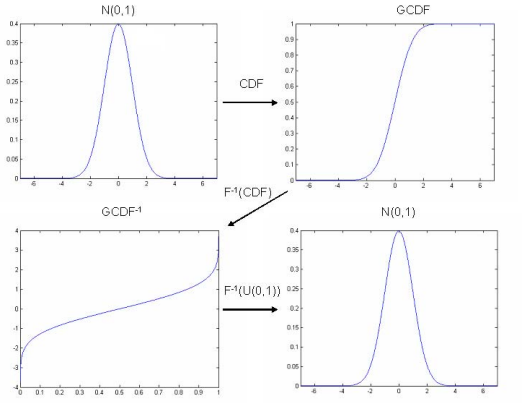

Fig. 1. Inversion Method

In [8] an inversion based GRNG is presented with splines approximation of polynomials of degree two with Chebyshev coefficients and a hierarchical segmentation of the function range. However this implementation does not ensure very high accuracy due to the usage of a fixed point arithmetic of 16 bits and the selected interpolation degree.

In this work we present a floating point GRNG with a very high accuracy and single precision floating point (f.p.) variables. This accuracy is required by some kind of simulations as is the case of financial Monte Carlo. This implementation is based on a high quality approximation with splines of degree five and Hermite coefficients [9]. An adaptative segmentation algorithm divides the function range into non-uniform segments ensuring, with the f.p. arithmetic, very high accuracy of the inversion.

This paper is composed as follows: in section II Hermite interpolation and the segmentation algorithm are introduced, while the architecture of the GRNG is exposed in section III. Section IV details the experimental results achieved and finally some conclusions are drawn.

II. Gaussian CDF^{-1} with Hermite Interpolation

A software implementation of a GRNG with Hermite interpolation has been previously studied by Hörmann et al in [10] and can be found in [11]. Hermite interpolation has been selected as interpolation method due to two facts, the better results that it obtains for the same interpolation degree with respect to other methods and, since it is a local approximation, its accuracy can be improved just by introducing more

1-4244-1176-9/07/$25.00 ©2007 IEEE.

segmentation points where needed.

For each segment of the GCDF [p_i,p_{i+1}] where $u_i = CDF(p_i)$, the inverse function is interpolated with a polynomial. In [10] several polynomial degrees of Hermite interpolation (linear, cubic and quintic) are studied. The quintic interpolation obtains the best results in terms of accuracy (almost exact) and lower number of segments.

So the general equation of the interpolation polynomial for each segment is the 5 degree polynomial:

$$H_i^5(\bar{u}) = a_{i0} + a_{i1}\bar{u} + a_{i2}\bar{u}^2 + a_{i3}\bar{u}^3 + a_{i4}\bar{u}^4 + a_{i5}\bar{u}^5$$

where $\bar{u} = u - u_i$ being u the uniform variable to invert.

For each segment the value of the coefficients is [9]:

$$a_{i0} = p_i \ ; \ a_{i1} = \frac{1}{f_i} \ ; \ a_{i2} = -\frac{f_i'}{2f_i^3}$$

$$a_{i3} = \frac{3f_i'/f_i^3 - f_{i+1}'/f_{i+1}^3}{2\Delta u} + \frac{10\Delta s - 6/f_i - 4/f_{i+1}}{\Delta u^2}$$

$$a_{i4} = \frac{-3f_i'/f_i^3 + 2f_{i+1}'/f_{i+1}^3}{2\Delta u^2} + \frac{-15\Delta s + 8/f_i + 7/f_{i+1}}{\Delta u^3}$$

$$a_{i5} = \frac{f_i'/f_i^3 - f_{i+1}'/f_{i+1}^3}{2\Delta u^3} + \frac{6\Delta s - 3/f_i - 3/f_{i+1}}{\Delta u^4}$$

where $f_i = f(p_i)$, $f_i' = f'(p_i)$, $\Delta u = u_{i+1} - u_i$, $\Delta p = p_{i+1} - p_i$, $\Delta s = \frac{\Delta p}{\Delta u}$ and it has been used $(F^{-1}(u_i))'' = \frac{-f'(p_i)}{f(p_i)^3}$

The segmentation of the function range and the calculation of the coefficients for each segment can be done in a setup stage previously to start with the generation of Gaussian variables. Once the setup stage is finished, the generation of the each Gaussian random variable consists on:

1) Generation of a uniform random variable u.
2) Search for the segment corresponding to u.
3) Extraction of the coefficients of that segment.
4) Calculation of GCDF$^{-1}(u)$ applying $H_i^5(\bar{u})$.

This calculation methodology fits very well with a hardware architecture. While no changes are done in the segmentation policy or in the type of the interpolation used, the setup stage is always the same and can be realized in a software platform. Its results (the coefficients and the starting points of each segments u_i) can be stored in tables. So, the hardware architecture only needs memory tables with those setup results.

A. Accuracy-adaptative Segmentation Policy

Given a desired accuracy, the range of a function can be segmented in two different ways according to two objectives, to obtain the smallest number of segments or to achieve an efficient segmentation in terms of the segment search. Segment search can compromise the performance of the GRNG taken that usually search algorithms are multicycle. To avoid this problem, segmentation points can be chosen in such a way that segment search of the uniform variable is restricted to analyze the value of some of its bits. One example of this segmentation is the Hierarchical segmentation used in [8].

However, selecting the segmentation points based on their search easiness has the negative effect of increasing the number of segments needed to achieve the desired accuracy, and for very high accuracy the number of segments is too high.

To obtain as less as possible number of segments, an accuracy-adaptative segmentation method, based on the iterative one employed on [11], has been used. For calculating each segment, the whole function range that has not been segmented yet, is taken as a segment. Then the coefficients for that segment are calculated, and accuracy is measured in the middle point of the segment (where the highest approximation error is expected) on the uniform range. If the obtained error is bigger than the desired accuracy the segment is divided by its middle point. The measuring and dividing process is repeated until a segment that achieves the desired accuracy is obtained. Then the next segments are calculated with the same method until all the function range is segmented.

Our method introduced several modifications to the original one. First, we have adapted the method to the uniform values that we are going to use in the hardware architecture, 32 bits values (from H"00000000" to "FFFFFFFF"), instead of double floating points values. So segments initial points always will match with a uniform of the hardware range and will allow a search algorithm based on integer search. Second, we have changed the measuring in the middle point from an absolute error on the uniform axis, to measure accuracy as a relative error in the gaussian axis using direct inversion [7] to measure the error. This way, the true error is measured and not approximated as before, and using a relative error we can ensure that, for all segments, all f.p. gaussian variables generated have the same number of mantissa bits of accuracy.

With this method we obtain non-uniform, non hierarchical segments adapted to the desired accuracy (number of accurate bits of mantissa). This type of segments makes difficult the segment search and an adaptation of a multicycle software algorithm is needed, as described in section III-B.

B. Segmentation and Coefficient Analysis

The analysis of the results obtained for both the original segmentation method and ours, has been realized with a double f.p. arithmetic and different searched error bounds. This analysis has focused on three aspects: accuracy, number of segments and value of the coefficients. Respect to the two first aspects, the analysis proved that the final accuracy obtained (for a reasonable number of segments) is limited by the interpolation in the extremes of the GCDF^{-1}, where the function tends to infinity and its value is in the order of single f.p. arithmetic. So single f.p. arithmetic can be adopted without significant accuracy loss. In these extremes GCDF^{-1} is hardly interpolated and the last segments are very small, only containing one point of the possible uniform values.

Regarding the coefficient values, it has to be considered that GCDF^{-1} is an odd function around 0.5 so the generated Gaussian Random variables obtained from u and 1-u will only differ in the sign. This way, it is only needed to segment half of the GCDF^{-1} to implement the inversion. The coefficient values obtained can be clearly differentiated between the segments before and after 0.5. All segments before 0.5 have

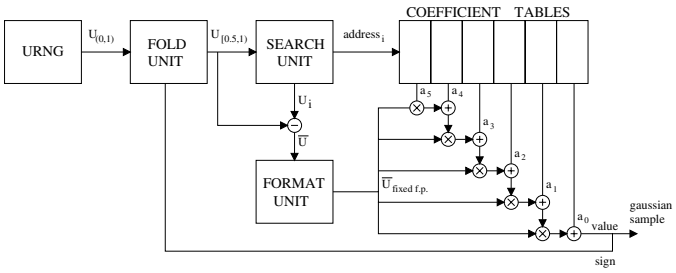

Fig. 2. GRNG architecture

negative coefficients, while for the segments after 0.5, only the segment starting with 0.5 (only one coefficient and very close to zero) and the segments of the extreme (those containing only one possible uniform values) have negative coefficients.

Additionally, considering the single f.p. arithmetic, all of the coefficients are normalized numbers and their multiplication by the minimum $\bar{u}$ value (not considering zero) 2^{-32} are also normalized numbers.

Negative coefficients imply subtractions in the polynomial and an overhead in the hardware architecture, see section III-A. Thus the upper half of GCDF^{-1} has been selected to implement the inversion and its negative coefficients have been eliminated. Replacing the negative coefficient of the segment starting with 0.5 by a zero has no consequences on the global accuracy obtained (due that its value is very close to zero and much smaller than the other segment coefficients) while segmentation in the extreme has been recalculated with linear interpolations including two uniform values. This way, like GCDF^{-1} is monotonically increasing, the interpolations of these segments have only positive values (a_{i0} and a_{i1} and the rest zeros) and the accuracy on the extreme is improved while the number of segments is reduced.

III. Hardware Architecture

A pipelined hardware architecture has been designed according to the previous analysis and using as base uniform random number generator a 32 bits one. In Figure 2, the implemented architecture is depicted (without the pipeline registers). As can be observed, the 5 degree polynomial is calculated with Horner's rule[1] reducing the calculation of the polynomial to five multiplications and additions.

Apart from the polynomial calculation, the architecture is composed of four main units. The URNG generates the base uniform variables of 32 bits in the range (0,1) that are transformed to the range [0.5,1) in the Fold Unit. The calculation segment for the uniform variable is obtained in a search unit and finally a Format Unit transforms the fixed arithmetic of $\bar{u}$ into fixed f.p. (see Section III-A).

A. Tailored Floating Point Arithmetic

The use of a standard f.p. arithmetic instead of a fixed one introduces a very heavy computational overhead because of the five types of numbers than can be managed in f.p. arithmetic: normalized, denormalized, zeros, infinities and NaN (not a number). To handle with those five types f.p. arithmetic units are composed of three subunits:

- Operands Prenormalization Unit
- Computation Unit
- Result Normalization Unit

being prenormalization and normalization units the ones that introduce the most significant overhead.

However, in our GRNG not all the five types of numbers are needed. Infinities and NaN can never be produced due to the obtained coefficients and the operations done. Additionally, the elimination of negative numbers (so there are no subtractions) in combination with the use of Horner's rule (alternating multiplications and additions) and the coefficient values obtained, implies that denormalized numbers are not generated as well.

So, our f.p. arithmetic is greatly simplified because it only needs to handle normalized numbers and zeros (handling denormalized numbers composes the most part of the prenormalization and normalization) and adders do not need the logic requiered for subtraction.

The tailored f.p. arithmetic is also based on the arithmetic used for $\bar{u} = u - u_i$. $\bar{u}$ is one of the operands for the five multipliers so how it is represented affects to the resources used by the pipeline and to the logic needed in the multipliers. This way we have selected a fixed f.p. format according to the range of values of $\bar{u}$ (maximum value is determined by the largest segment while its minimum value is zero). Using a fixed f.p. format overcomes the possible problem of resolution that can appear with a single f.p. format (depending on the maximum value of $\bar{u}$, we can need more than 23 mantissa bits) and simplifies the logic of the multipliers.

B. Search Unit. Search Algorithm

One of the key factors of this work is the search algorithm developed. Non-uniform, non-hierarchical segmentation makes necessary some kind of hardware adapted search algorithm to overcome the multycle search of software algorithms.

GCDF^{-1} is monotonically increasing and so will be the resulting segmentation. Consequently, ordered search methods are the ones that best fit with our search. Specifically, we have selected an indexed searching method (that uses index and search tables) to take advantage of the characteristics of nowadays FPGAs, with dual port RAM memory blocks.

The index search method has been extended to ensure that the pointer obtained from an index table always points to the correct segment or the segment immediately below in the search table. This way each search can be finished with just one access to the indexed table to obtain the pointer and another one to the search table reading simultaneously segments starting points at pointer and pointer+1. A subsequent comparison of the searched value with the segment starting point at pointer+1 will determine to which of the two segments belongs the searched value.

To obtain the desired index table (in the setup stage), a local search scheme based on fixed arithmetic and multiple local index tables was developed (searched variables are 32 bits fixed variables). The value range of searched variables is

[1] $p(x) = a_0 + x(a_1 + x(a_2 + x(...)))$

TABLE I
IMPLEMENTATION RESULTS IN XILINX VIRTEX-II XC2V4000-6 FPGA

	Slices	Block RAMs	18x18 Mult	Speed [MHz]	Troughtput [sample/s]	Max σ	Absolute Accuracy	Relative Accuracy	Sample format
Lee et al [8]	585	1	4	231	231	8.2	$0.3*2^{-11}$	30%	fixed 16 bits
Ours	2022	5	15	185	185	6.23	$0.5*2^{-18}$	0.000047%	32bits single f.p.

divided into several parts corresponding to each part a local index table where one access search is ensured. The addresses for the index table, pointing to the corresponding local tables, are generated with a bit comparison scheme and attending to the range of values on input variables for each local table.

IV. EXPERIMENTAL RESULTS

The architecture for the GRNG has been developed on a Xilinx Virtex-II XC2V4000-6 FPGA and Table I resumes the performance of our floating point implementation compared to the previous fixed-arithmetic one [8].

The details of the implementation are the following:

- Uniform RNG: 32 bits Tausworthe with zero detection and substitution $=>$ Uniform range(2^{-32} to $1-2^{-32}$)
- Uniform and Gaussian RNG periodicity of 2^{88}.
- Searched accuracy on segmentation: 21 bits of mantissa.
- 256 segments corresponding to the upper half of CDF^{-1}
- Highly deep pipeline implementation of 55 stages
- One sample of 32 bits single f.p. per cycle.

The selection of a 32 bits URNG determines the maximum σ as GCDF$^{-1}(2^{-32}) = -6.23$.

A. Accuracy

The searched accuracy on the setup stage has been of 21 bits of mantissa (22 bits of accuracy counting the hidden bit of f.p. arithmetic), obtaining a segmentation of 202 segments. The rest of the segments, up to the total number of 256 comes from the setting of a maximum segment size (to minimize the number of bits needed to represent $\bar{u}$) and the use of linear interpolations in the extreme.

However, the achived accuracy is reduced on one bit due to two facts. The first one is that the accuracy-segmentation algorithm measures accuracy in the middle point of a segment, but the error approximation can be slightly bigger in points close to the middle. And the second fact is that a rounding truncation scheme has been used in f.p. units.

So, the hardware accuracy obtained has been of 20 bits of mantissa. As can be seen in Table I this means an improvement of seven more bits of absolute accuracy[2] respect to [8] for our worst case. And in terms of relative accuracy[3] the improvement achieved is of more than five orders of magnitude.

B. Performance

The improved accuracy comes at the cost of increasing the number of resources: the single f.p. arithmetic used instead of the fixed one, the higher interpolation degree and sample generation of 32 bits instead of 16 are the reasons why we need more resources than [8].

[2]Maximum error value between GCDF$^-$1 and the polynomial interpolation.
[3]Maximum percentage error in a Gaussian variable

The use of samples with more bits also explains the lose of speed and throughput (of a 20%). However this is a very small performance penalty taking into account that f.p. arithmetic has been used. The reason is that f.p. overhead is handled efficiently with a very deep pipelened architecture of 55 stages.

V. CONCLUSIONS

In this work we have presented a GRNG architecture that generates very accurate samples in a single floating point format with the inversion method. GDD^{-1} is approximated with a high quality five degree Hermite interpolation that ensures high accuracy with a non-uniform, non-hierarchical segmentation generating Gaussian samples with 20 bits of mantissa of accuracy. A highly deep pipelined architecture handles the overhead due to the use sf single f.p. arithmetic achieving a 185 Mhz speed and a throughput of one sample per cycle on a Xilinx Virtex-II FPGA.

ACKNOWLEDGEMENTS

This work has been partly funded by BBVA under contract P060920579 and by thr Spanish Ministry of Education and Science through the project TEC2006-00739.

REFERENCES

[1] G. Zhang, P. H. Leong, D.-U. Lee, J. Villasenor, R. C. Cheung, and W. Luk, "Ziggurat-based hardware gaussian random number generator," in *IEEE International Conference Field-Programmable Logic and its Applications*, 2005, pp. 275–280.

[2] D.-U. Lee, W. Luk, G. Zhang, P. H. Leong, and J. Villasenor, "A hardware gaussian noise generator using the Wallace method," *IEEE Transactions on VLSI Systems*, vol. 13, pp. 911–920, 2005.

[3] D.-U. Lee, W. Luk, J. Villasenor, and P. Y. Cheung, "A gaussian noise generator for hardware-based simulations," *IEEE Transactions on Computers*, vol. 53, pp. 1523–1533, 2004.

[4] ——, "A hardware gaussian noise generator using the Box-Muller method and its error analysis," *IEEE Transactions on Computers*, vol. 55, pp. 659–671, 2006.

[5] J. E. Gentle, *Random Number Generation and Monte Carlo Methods*. Springer-Verlag, 1998.

[6] P. Bratley, B. L. Fox, and L. E. Schrage, *A Guide to Simulation*. Springer-Verlag, 1983.

[7] M. Mori, "A method for evaluation of the error function of real and complex variable with high relative accuracy," *Publ. Res. Inst. Math. Sci.*, vol. 19, pp. 1081–1094, 1983.

[8] D.-U. Lee, R. C. Cheung, J. Villasenor, and W. Luk, "Inversion-based hardware gaussian random number generator: A case study of function evaluation via hierarchical segmentation," in *Field Programmable Technology*, 2006, pp. 33–39.

[9] R. L. Dougherty, A. Edelman, and J. M. Hyman, "Nonnegativity-, monotonicity-, or convexity-preserving cubic and quintic hermite interpolation," *Mathematics of Computation*, vol. 52, pp. 471–494, 1989.

[10] W. Hörmann and J. Leydold, "Continuous random variate generation by fast numerical inversion," *ACM Transactions on Modeling and Computer Simulation*, vol. 13, p. 347362, 2003.

[11] J. Leydold and W. Hörmann, "Unu.ran - a library for non-uniform universal random variate generation," Institut für Statistick, WU Wien, Austria, A-1090, Tech. Rep., 2002.