A Short Survey on Uniform Random Number Generator and Gaussian Ramdom Number Generator with Implementation on FPGA

***Abstract*—This short survey can be considerd as a brief summary of my investigation and research process recently. I've focused primarily on the algorithm of uniform random number generator (RNG) and Gaussian random number generator (GRNG) along with the implementation design and scheme on FPGA. The general structure of this survey follows as some basic konwledge and introduction of pseudo random number generation, hardware implementation designs of uniform random number generator, Gaussian random number generator along with its FPGA implementation.**

***Index Terms*—Ramdom Number Generator, Gaussian Random Number Generator, Hardware Implementaion, Survey Paper**

I. Introduction and Background

Random Number Generation (RNG) has a wide range of applications within various fields including game developments, statistical simulations, security protocols and encrypting documents. A series of physical methods for generating random numbers include thermal noise, clock jitter, radioactive decay, etc. Random numbers generated by computational methods are actually pseudo random numbers which means the same number sequence might return after a period of certain time. Our summer research project also focuses on the uniform pseudo random number generator and Gaussian random number generator. Algorithm of random number generation has always been a hot topic among the theoretical research literature. Universally used algorithms include linear congruential generator, Mersenne Twister, middle-square method, xorshift, etc. Linear Congruential Generator (LCG) gives an iterative equation as Eq. (1).The specific value of a, b and m must be carefully chosen for a maximum length sequence.

$$X_{n+1} = (a \cdot X_n + b) \bmod m. \tag{1}$$

A great amount of hardware implementation methods for random number generation are available through the hardware field. Since Verilog has a build-in random number generator, we could readily provide a quick way to generate random numbers as shown in Figure 1 .Being nice but only available for testbenches, a version for hardware implementation is needed. Normally, a pseudo random number generator in digital hardware consists of linear feedback shift registers (LFSR). We could specify an LFSR by means of the feedback polynomial as shown in Figure 2. Knowing the specific polynomial could be enough to draw the LFSR. Polynomials generating very long state sequences are called maximal-length LFSR which are potentially necessary for our project. Eq. (2) could be an

```
module random(q);
  output [0:31] q;
  reg [0:31] q;

  initial
    r_seed = 2;

  always
    #10 q = $random(r_seed);
endmodule
```

Fig. 1. Build-in random number generator in Verilog

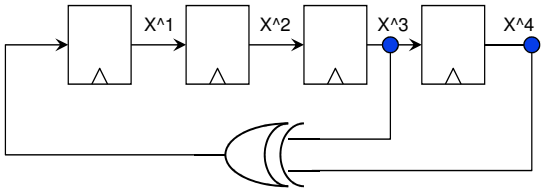

Fig. 2. Diagram of feedback polynomial

instance of the maximum-length feedback polynomial whose state machine owns a total of $2^{153} - 1$ states. By the way, the LFSR could be equivalently regarded as a finite state machine logically.

$$P_x = x^{153} + x^{152} + 1. \tag{2}$$

Through my survey, abundant resources of implementing RNG (Random Number Generator) by LFSR in Verilog could be referred to in the regarding literature which could potentially be a basis of our implementation of a uniform RNG (Random Number Generator). Whereas, RNG (Random Number Generator) could have a variety of flaws including bias (a certain number occurs more often than others) and predictability. Actually, LFSR has a small bias because the all-zero state never appears. However, for a very long LFSR, the bias becomes negligible. For a pseudo RNG (Random Number Generator), what generates could be a highly deterministic sequence. It could be proved that given an N-bit LFSR with

unknown feedback pattern, then only 2N bits are needed to predict bit 2N + 1. Some solutions can be provided like a maximal-length polynomial and a non-linear combination generator which need to be taken into consideration for our project especially when a long cycle is necessarily emphasized.

Apart from traditional hardware design of RNG (Random Number Generator), tremendous feasible improvement schemes have been proposed such as FPGA-optimised uniform random number generators using LUTs and shift registers [1]. FPGA-optimised Random Number Generators (RNGs) are more resource efficient than software-optimised RNGs, as they can take advantage of bit-wise operations and FPGA-specific features. However, it is difficult to concisely describe FPGA-optimised RNGs, so they are not commonly used in real-world designs. A LUT-SR RNG takes advantage of bit-wise XOR operations and the ability to turn LUTs into shift-registers of varying lengths. This provides a good resource-quality balance compared to previous FPGA-optimised generators, between the previous high-resource high-quality LUT-FIFO RNGs and low-resource low-quality LUT-OPT RNGs.

II. Gaussian Random Number Generator

A. *Gaussian Random Number Generation*

The standardised continuous uni-variate Gaussian Probability Distribution (PDF) is defined as:

$$\phi(x) = \frac{1}{\sqrt{2\pi}} exp(-\frac{x^2}{2}). \tag{3}$$

with the Cumulative Distribution Function (CDF):

$$\Phi(x) = \int_{-\infty}^{x} \phi(t)dt = \frac{1}{2}(1 + erf(x/\sqrt{2})). \tag{4}$$

Gaussian distributions with different means and variances can be created through basic scaling and translation of the standardised distribution, so normally only the standardised version is considered. In software, the continuous Gaussian distribution is the target distribution for GRNGs, as single-precision floating-point numbers can be considered a good approximation to continuous variates. However, in FPGA architectures it is common to use fixed-point numbers, with the number of fractional bits being chosen to minimise resource usage while maintaining the numerical and statistical properties required in the target application.

There exist a set of orthodox algorithms for generating GRN (Gaussian Random Numbers) :Box-Muller algorithms [2], [3], polarization decision algorithm [3], [4], central limit algorithm [3], [5], etc. The Box-Muller transform is one of the earliest exact transformation methods [3]. It produces a pair of Gaussian random numbers from a couple of uniform numbers. The polar method is an exact method related to the Box-Muller transform and has a closely related two-dimensional graphical interpretation, but uses a different approach to get the 2D Gaussian distribution [3], [4]. The probability density function describing the sum of multiple uniform random numbers is obtained by convolving the constituent probability density function. Thus, by the central limit theorem, the probability density function of the sum of K uniform random numbers over the range (0, 1) will approximate a Gaussian distribution. During the current research period, we choose to focus on the Central Limit Algorithm primarily. The central limit algorithm is based on the central limit theorem which states that when a sufficiently large number of samples drawn from independent random variables (i.e., uniform distributions), the arithmetic mean of their distributions will have a normal distribution. Thus, the central limit algorithm is an extremely efficient method in GRNs generation, since it simply samples sufficient amount of identical and independent uniform distributions.

More formally, assume there are n independent and identically distributed uniform numbers $U_i \sim U(0,1)$. Then we can approximate results of the sum of U_i as S=$\sum_{i=1}^{n} U_i$.The cumulative distribution function of S can be approximated as:

$$F_S(s) = \Phi(\frac{s - n\mu}{\sqrt{n\sigma^2}}). \tag{5}$$

where Φ represents the cumulative distribution function of a Gaussian distribution. For a uniform random variable $U_i \sim U(0,1)$, the mean μ and variance σ are given by $\frac{1}{2}$ and $\frac{1}{12}$ respectively. Thus if we choose variable z = $\frac{s-\frac{n}{2}}{\frac{1}{12}\sqrt{n}}$ the distribution function of z is Gaussian distribution:

$$f_Z(z) = \frac{n}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}. \tag{6}$$

After normalization, a standard Gaussian distribution is obtained. Central limit algorithm can be used to transform uniform random numbers to Gaussian with a very low hardware cost. However, it deserves attention that the GRNs (Gaussian Random Numbers) produced by central limit algorithm is not highly accurate in tail region [3].

B. *Hardware Implementation*

For hardware implementation, we also focus on the Central Limit Algorithm version. Figure 3 shows one instance of the design diagram of the Gaussian random generator, using central limit algorithm. The uniform random numbers $U_1, U_2 ... U_n$ are generated by MSRG (Multi-return Shift Register Generator) module. All the random numbers $U_1, U_2 ... U_n$ and indispensable constants are converted into single-precision floating point numbers. The new variable S = $\sum_{i=1}^{n} U_i$ is obtained by an adder and the square root is calculated by module 'SQRT'. The division function is achieved by modules 'DIV'. Tow external multipliers and one adder are used in the design. As a result, one set of Gaussian random numbers is obtained, i.e., α. Besides, some other feasible design schemes for GRNG (Gaussian Random Number Generator) are also available for referring to during our research process [6].

III. Conclusion

In this survey report, we give a brief introduction of the pseudo random number generation and focus on the uniform random number generator (URNG), Gaussian random number generator (GRNG) and the hardware implementation designs. A series of different versions of algorithms are provided

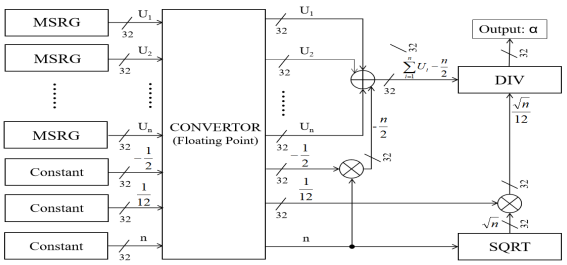

Fig. 3. Gaussian random number generator using central limit algorithm

for implementing the two kinds of RNG (Random Number Generator). Abundant resources including published papers and open-source projects could be referred to during our process of designing the uniform random number generator (URNG) and Gaussian random number generator (GRNG).

REFERENCES

[1] D. B. Thomas and W. Luk, "Fpga-optimised uniform random number generators using luts and shift registers," *International Conference on Field Programmable Logic and Applications*, 2010.

[2] B. GEP and M. ME, "A note on the generation of random normal deviates," *Ann math statist*, 1958 06;29(2):610-611.

[3] D. B. Thomas, W. Luk, P. Leong, and J. Villasenor., "Gaussian random number generators," *ACM Computing Surveys*, 2007;39(4):11.

[4] J. Bell, "Normal random deviates," *Commun ACM*, 1968 Jul;11(7):498.

[5] D. Knuth, "Seminumerical algorithms," *The art of computer programming, Volume 2 (3rd Ed)*, 1997.

[6] D. B. Thomas, "Fpga gaussian random number generators with guaranteed statistical accuracy," *IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, 2014.