BoxMuller AWGN Signal Generator For FPGA

zhanxn@gmail.com

2018/08/26

Acknowledgement

Box-Muller AWGN signal generator implemented for FPGA/ASIC is presented. Mainly refer to [6] and [7], thanks for the authors.

Features

- Based on the Box-Muller algorithm with no central limit theorem required
- Piecewise polynomial based Chebyshev function approximation units with range reduction
- Period of generated noise sequence is 2^88 (about 10^25)
- 18-bit noise samples with 5 bits of integer and 13 bits of fraction, accurate to one unit in the last place (ulp) up to 8.1573σ, which models the true Gaussian PDF accurately for a simulation size of over 10^{15} samples
- Core can be reset to its initial state and work on clock enable signal
- Bit-true MATLAB and C mex programs included
- Maximum clock rate and output sample rate of 320MHz on Xilinx VCU108 Board

Box-Muller Algorithm

The **Box–Muller transform**, by George Edward Pelham Box and Mervin Edgar Muller [1], is a pseudo-random number sampling method for generating pairs of independent, standard, normally distributed (zero expectation, unit variance) random numbers, given a source of uniformly distributed random numbers.

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).

$$x_0 = \sqrt{-2\ln(u_0)}\,\sin(2\pi u_1) \tag{1}$$

$$x_1 = \sqrt{-2\ln(u_0)}\,\cos(2\pi u_1) \tag{2}$$

Split the procedure into three steps for hardware implementation:

$$e = -2\ln(u_0) \tag{3}$$

$$f = \sqrt{e} \tag{4}$$

$$g0 = \sin(2\pi u_1),\ g1 = \cos(2\pi u_1) \tag{5}$$

Fixed Point Method

Generally, when evaluating channel codes, one needs 100 to 1000 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}.
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 period of at least 10^{15}.
By examining the normal distribution N(0,1) using *norminv* function in MATLAB, 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).

Bit-width for u0 and u1

Let

$$\max(\sqrt{-2\ln(u_0)}) = 8.1$$

Calculate minimum u0:

$$\min(\mathrm{u}_0) = 5.6620\mathrm{e}^{-15}$$

So u_0 should be represented with no less than 48 bit width with 1ups of $1/2^{48} = 3.5527\mathrm{e}^{-15}$.

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.

Bit-width for e, f and g

After generating the two 32-bit Tausworthe URNGs numbers u0 and u1. The implementation of (3)~(5) in hardware utilizes look up tables where the coefficients were found using Chebyshev series approximations [5].
Moreover, function (3) and (4) processing need to split into three steps: range reduction, function approximation and range reconstruction [6][7]. These steps are utilized because it facilitates the implementation by lowering the complexity to only approximating the function on a smaller input interval. Truncation is used here to further decrease hardware complexity.
Chebyshev coefficients for (3) (4) and (5) approximation are generated with MATLAB codes, refer to “chebv_approx.m” file. For example, the Degree-2 Chebyshev approximation coefficients for –ln(x) calculation is like this:

```
%% -ln(x) approximation
k = 8;
n = 3; %Degree+1, degree of Chebyshev app
ir = 1; %integer offset, 1 means range [1,2)
t = cos(((2*(0:n-1)+1)*pi)/(2*n)); %chebyshev
intv_num = 2^k;
for intv=1:intv_num
    %input range of current interval
    a = ir+(intv-1)/intv_num;
    b = ir+intv/intv_num;

    xi = (t*(b-a)+(b+a))/2; %change range to [a,b)
    fln = -log(xi);

    Cn0(3,intv) =
      fln(1)/((xi(1)-xi(2))*(xi(1)-xi(3))) ...
    + fln(2)/((xi(2)-xi(1))*(xi(2)-xi(3))) ...
    + fln(3)/((xi(3)-xi(1))*(xi(3)-xi(2)));

    Cn0(2,intv) =
      fln(1)*(-xi(2)-xi(3))/((xi(1)-xi(2))*(xi(1)-xi(3))) ...
    + fln(2)*(-xi(1)-xi(3))/((xi(2)-xi(1))*(xi(2)-xi(3))) ...
    + fln(3)*(-xi(2)-xi(1))/((xi(3)-xi(1))*(xi(3)-xi(2)));

    Cn0(1,intv) =
        fln(1)*(xi(2)*xi(3))/((xi(1)-xi(2))*(xi(1)-xi(3))) ...
      + fln(2)*(xi(1)*xi(3))/((xi(2)-xi(1))*(xi(2)-xi(3))) ...
      + fln(3)*(xi(2)*xi(1))/((xi(3)-xi(1))*(xi(3)-xi(2)));

    xm = (intv - 1)/intv_num;
    Ce(3,intv) = (Cn0(3,intv))*2^(-2*k);
    Ce(2,intv) = (Cn0(3,intv)*(2+2*xm)+Cn0(2,intv))*2^(-k);
    Ce(1,intv) =  Cn0(3,intv)*(xm+1)^2+Cn0(2,intv)*(xm+1) +  ...
                  Cn0(1,intv);
end
```

The error analysis starts at the end and makes its way back to the beginning because the main goal is for the output's error to meet the faithful rounding requirement. Please refer to [6] and [7] for detail analysis description. The result of fixed point from [6] is as Figure 1 (f and g width expanded).

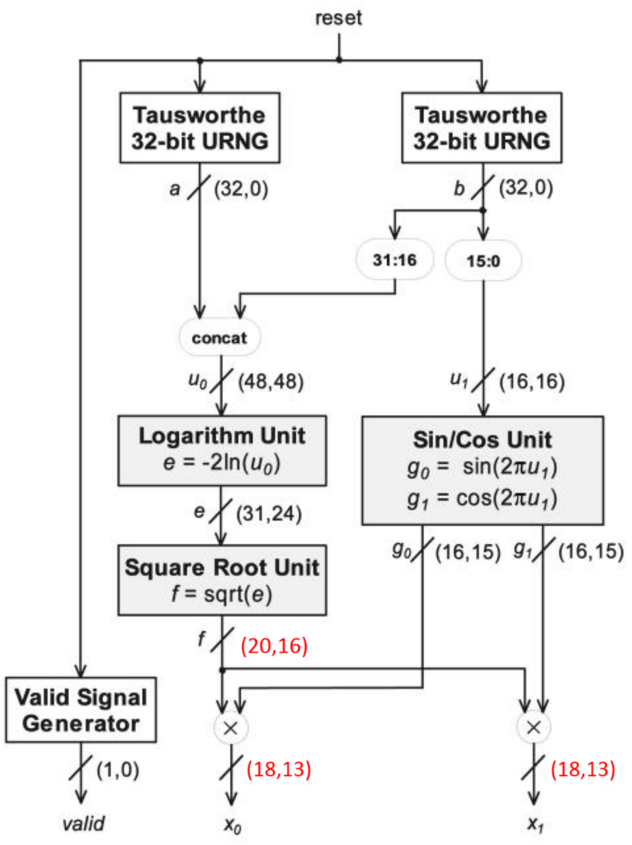


Figure 1 Hardware Computation procedure diagram from [6]

Hardware Implementation

Tausworthe URNG Generation

Although traditional linear feedback shift registers (LFSRs) are often sufficient as a uniform random number generator (URNG), Tausworthe URNGs [3] are fast and occupy less area. Tausworthe URNG follows the algorithm presented by L'Ecuyer [4], 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} (about 10^{25}).

```
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;
}
```

Figure 2 Tausworthe URNG generation codes

Normal Distribution Performance Test

Test Environment

Taus_URNG, seeds are 2846420573 2846420573 2846420573
Taus_URNG, seeds are 912462866 912462866 912462866
Test Generated samples number for analysis: 1e7

Fixed point Accuracy Evaluation

Fixed point output is compared to float point model (with same URNG input):

	Floor truncation	Round
e max error value (abs)	6.3224e-08	3.4341e-08
f max error value (abs)	3.1588e-05	2.3977e-05
g0/g1 max error value (abs)	1.0174e-04 1.0174 e-04	1.1467e-04 1.1467 e-04
x0/x1 max error value (abs)	4.8891e-04 4.8123e-04	5.5051e-04 6.1058e-04
x0/x1 mean value	-5.6479e-04 8.4952e-05	-5.6477e-04 8.4947e-05
x0/x1 variance value	0.99978 1.0001	0.99982 1.0001

The results show that accuracy loss caused by truncation can be neglected. So truncation method is used in hardware.

Anderson-Darling test result

adtest result (vs. MATLAB *randn()* function):

	H	P	adsta	cv
This design	0	0.7810	0.2398	0.7519
MATLAB randn()	0	0.3270	0.4209	0.7519

Distribution Figure

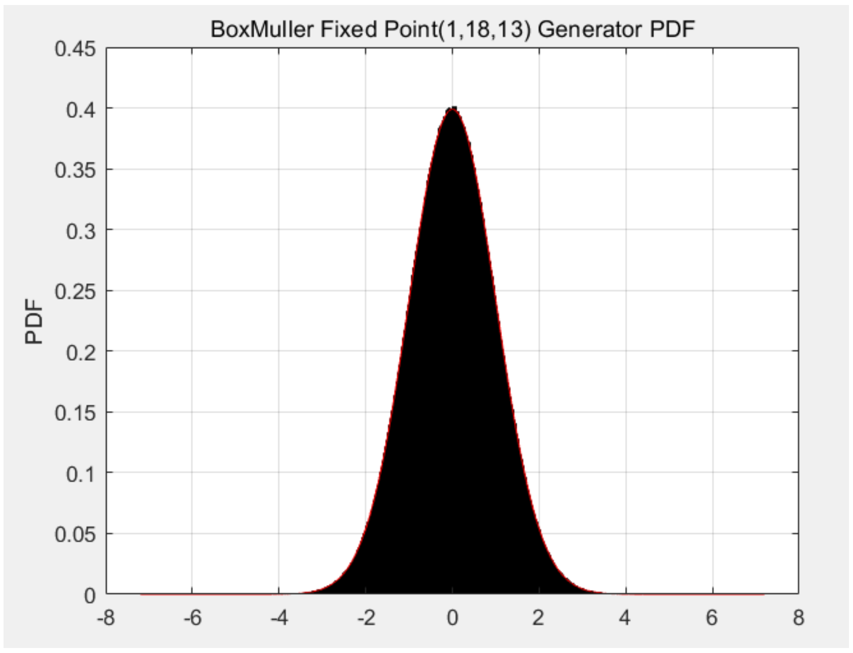
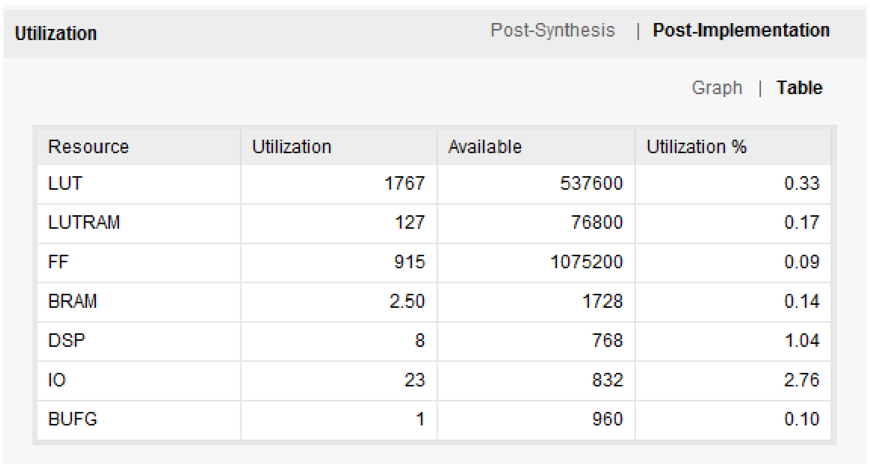


Figure 3. PDF Distribution of generated 1e7 samples

Resource Utilization and Throughput

Xilinx VCU108 FPGA Evaluation Board
Fmax : 330MHz (can be improved further more)
Latency : 16 cycles



Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	1767	537600	0.33
LUTRAM	127	76800	0.17
FF	915	1075200	0.09
BRAM	2.50	1728	0.14
DSP	8	768	1.04
IO	23	832	2.76
BUFG	1	960	0.10

Figure 4. Resource Utilization

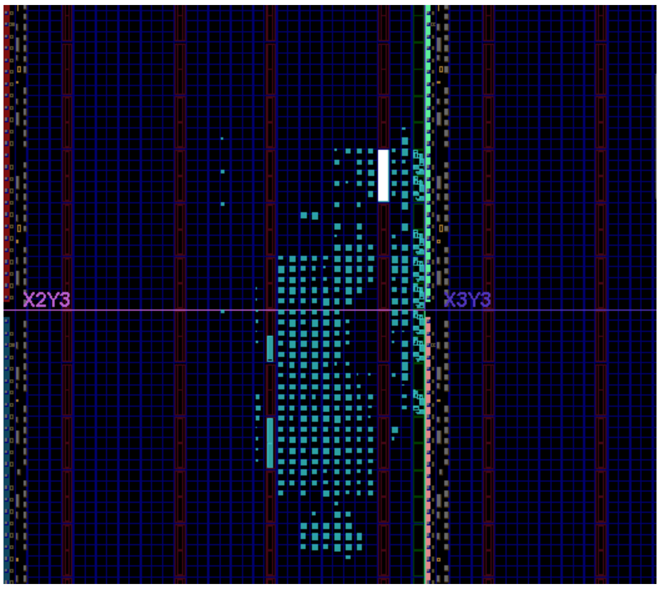


Figure 5. After Place and Route Photograph on Xilinx FPGA

More Methods to generate AWGN

Wallace method

D.-U. Lee, W. Luk, J. D. Villasenor, G. Zhang, and P. H. W. Leong, "A hardware gaussian noise generator using the wallace method," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 13, no. 8, pp. 911–920, Aug 2005.

Ziggurat methods

W. Hörmann and J. Leydold, "Continuous random variate generation by fast numerical inversion," ACM Trans. Model. Comput. Simul., vol. 13, no. 4, pp.347–362, Oct. 2003. [Online]. Available

Inversion method

R. Gutierrez, V. Torres, and J. Valls, "Hardware architecture of a gaussian noise generator based on the inversion method," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 59, no. 8, pp. 501–505, Aug 2012.

R. C. C. Cheung, D. U. Lee, W. Luk, and J. D. Villasenor, "Hardware generation of arbitrary random number distributions from uniform distributions via the inversion method," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 15, no. 8, pp. 952–962, Aug 2007.

CORDIC use

J. S. Malik, A. Hemani, and N. D. Gohar, "Unifying cordic and box-muller algorithms: An accurate and efcient gaussian random number generator," in 2013 IEEE 24th International Conference on Application-Specifc Systems, Architectures and Processors, June 2013, pp. 277–280.

Y. Wang and Z. Bie, "A novel hardware gaussian noise generator using box-muller and cordic," in 2014 Sixth International Conference on Wireless Communications and Signal Processing (WCSP), Oct 2014, pp. 1–6.

Reference

[1] G. E. P. Box and Mervin E. Muller, *A Note on the Generation of Random Normal Deviates*, The Annals of Mathematical Statistics (1958), Vol. 29, No. 2 pp. 610–611 doi:10.1214/aoms/1177706645, JSTOR 2237361

[2] Box-Muller Transform wiki page https://en.wikipedia.org/wiki/Box%E2%80%93Muller_transform#cite_note-1

[3] R.C. Tausworthe, “Random Numbers Generated by Linear Recurrence Modulo Two,” Math. and Computation, vol. 19, pp. 201-209, 1965.

[4] P. L’Ecuyer, “Maximally Equidistributed Combined Tausworthe Generators,” Math. Computation, vol. 65, no. 213, pp. 203-213, 1996.

[5] M. J. Schulte and E. E. Swartzlander, “Hardware designs for exactly rounded elementary functions,” IEEE Transactions on Computers, vol. 43, no. 8, pp. 964–973,Aug 1994.

[6] D. U. Lee, J. D. Villasenor, W. Luk, and P. H. W. Leong, “A hardware Gaussian noise generator using the box-muller method and its error analysis,” IEEE Transactions on Computers, vol. 55, no. 6, pp. 659–671, June 2006.

[7] Lincoln Glauser, “The Design of a 24-bit Hardware Gaussian Noise Generator via the Box-Muller Method and its Error Analysis”, (2017). Tesis. Rochester Institute of Technology.