AWGN Using Box-Muller Method

Features:

- Generates Gaussian 16-bit noise samples accurate up to 8.2σ which models true Gaussian PDF accurately for simulation size of over 10^15 samples.
- Here 16 bit noise sample here are not represented using 2's complement form rather MSB bit represents the sign of the noise sample and next 4 bits are represents Integer bits and the last 11 bits represents fractional bits.
- FPGA's are used to generate the Noise samples
- AWGN IP core is designed using Verilog and the test bench is also written in Verilog which can record all the Noise samples.
- AWGN IP core is based on Box-Muller method.
- Bit true MATLAB emulation model is also provided
- Max of 625Mhz routable frequency can be used for Virtex-7 device

Applications:

- Used for Channel code evaluation
- Measuring BER performance
- FPGA based AWGN generator is much efficient than software generated noise samples by speeding the simulations by several orders of magnitude

Design Configurations:

AWGN IP Specifications	
Supports family devices	Supports Xilinx FPGA's that has large I/O's greater than 210
Tools Used	Xilinx Vivado 2015.1 MATLAB 2016a
Resources utilized on FPGA	FF's - 993 LUT's – 1216 Memory LUT 's – 43 IO – 210 BRAM - 2 DSP48 - 7 BUFG - 1

Functional Description:

Box-Muller architecture is realized based on these equations:

$$e = -2\ln(u_0),$$

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

$$g_0 = \sin(2\pi u_1),$$

$$g_1 = \cos(2\pi u_1),$$

$$x_0 = f * g_0,$$

$$x_1 = f * g_1,$$

Variables u_0 and u_1 should be over interval [0, 1) to produce random Gaussian samples of x_0 andx_1. These are produced by Tausworthe Uniform Random Number Generator (URNG) which generates 32 bit width random variables with a period of ~10^{25}. Since we need two random variables u_0 and u_1 we use two URNG's.

The above equations has to be designed in hardware (FPGA's) to produce noise samples. All the above equations are represented by fixed point arithmetic which is realized by mini bit width optimization which quantizes to finite precision. Here I used chopping off bits to optimize bit widths.

By applying minimum bit width optimization methods the above equations we get the following bit widths:

1. Bit-width of u_0 is 48 bit of which fractional bit width of 48
2. Bit-width of u_1 is 16 bit of which fractional bit width of 16
3. Bit-width of f is 17 bit of which fractional bit width of f is 13.
4. Bit-width of e is 31 bit of which fractional bit width of e is 24.
5. Bit-width of g_0 and g_1 is 16 bit of which fractional bit width of g_0 and g_1 is 15.
6. Bit-width of x_0 and x_1 is 16 bit of which fractional bit width of x_0 and x_1 is 11, integer bit width of 4 and sign bit of 1.

Bit-width u_0 of 48 bits are obtained by concatenating 16 MSB bits of other URNG and Bit-width u_1 of 16 bits is obtained by remaining 16 bits of URNG.

Practical Implementation:

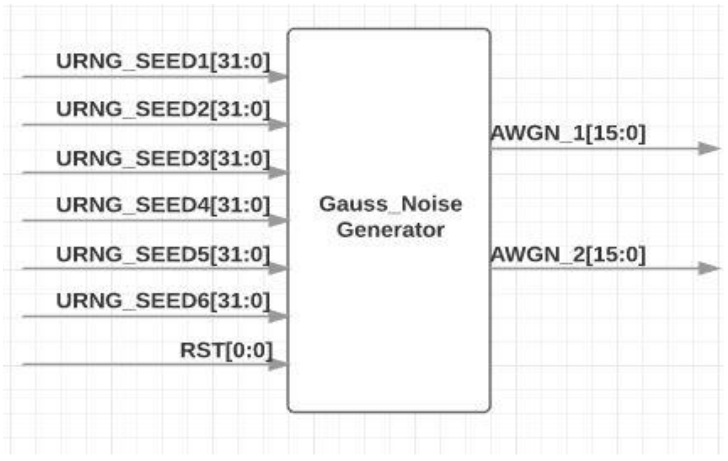

Fig 1. AWGN IP block diagram

In the Initial step of design AWGN IP first we have to develop URNG's based on the algorithm given. URNG's gives out random numbers in range [0, 1).

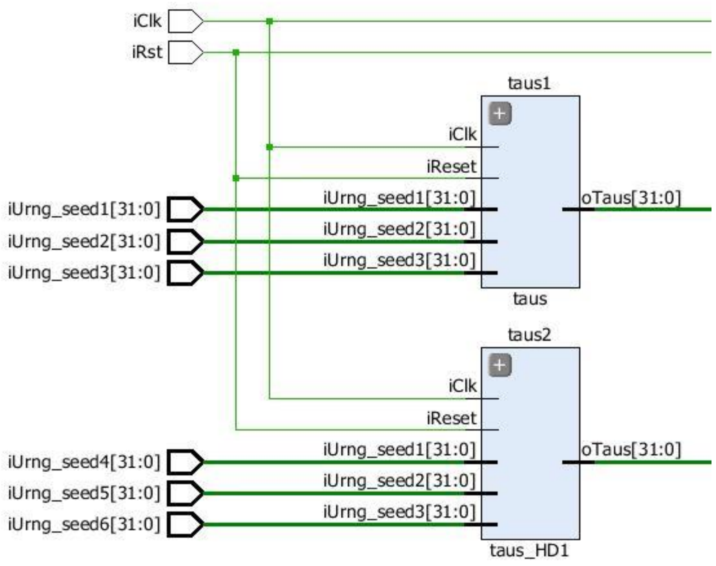

Fig 2. URNG

After developing random u_0 and u_1 now the next step is to find $e = -2 * \ln(u_0)$. You can't find a direct solution like in MATLAB in hardware. We have to find the solution $y_e = -\ln(x_e)$ where x_e should be in the range [1, 2). So range of e should be reconstructed to the range [1, 2) and it is represented by x_e. Bit width of x_e is 49 bits of which 48 bits are fractional bits and 1 bit of integer bits.

$$y_e = -\ln(x_e)$$

Function is approximated by using piece wise polynomials with uniform segments. These polynomial coefficients are found using "`polyfit`" MATLAB function which provides constants of required range. From the IEEE paper [1] we came to know that for natural log function, we need degree 2 polynomial function with uniform segments of about 256. All these information is found out from the MATLAB and saved in "block ram" of FPGA devices using ".mif" extension files which configures the memory in FPGA. The obtained result is in range [1, 2) but this has to be transformed/ reconstructed to the original range which is given the pseudo code line 17 to 20 [1].

In next step we have to find the square root of the logarithmic function:

$$f = \sqrt{e}$$

Initial step is to reduce the range of the function in between the interval [1, 4) and this procedure is known from the IEEE paper pseudo code lines 24 to 27 [1]. Now we have approximate the function which is in the range [1, 4) and from [1] we came to know that we need about 64 uniform segments with a polynomial degree of 1 to approximate. After approximation now we need to reconstruct the function according to lines 33 to 35 according to pseudo code [1].

As of now we have found f let us concentrate on finding g_0 and g_1 to get x_1 and x_0. The equations of g_0 and g_1 are evaluated as below:

$$g_0 = \sin(2\pi u_1),$$

$$g_1 = \cos(2\pi u_1),$$

The 16 bit value of u_1 gets from LSB of one URNG initially range reduction is done according to lines 33 to 35 according to pseudo code [1]. After that approximate the cosine and sine function over the linear range of cosine function from 0 to $\pi/2$. From pseudo code [1] lines from 45 to 49 we need to approximate cosine and sine functions. After approximation we need to reconstruct the cosine and sine functions from range 0 to $\pi/2$ to range of 0 to 2π.

As of now we have f, g_0 and g_1, we can implement the final following equations:

$$x_0 = f * g_0 \text{ and } x_1 = f * g_1$$

Evaluating Hardware design with Bit-width optimized MATLAB model:

I have designed a MATLAB model which emulates exactly the way hardware/Verilog has to behave. So lets us compare the results by taking the outputs at a single clock cycle of Verilog design with MATLAB bit width model.

Verilog	**MATLAB**
I/P: URNG Seed1 :1999 URNG Seed2 :2995 URNG Seed3 :3666 URNG Seed4 :3658 URNG Seed5 :1564 URNG Seed6 :4578 O/P: URNG output1: 484203280 URNG output2: 589611434	I/P: URNG Seed1 :1999 URNG Seed2 :2995 URNG Seed3 :3666 URNG Seed4 :3658 URNG Seed5 :1564 URNG Seed6 :4578 O/P: URNG output1: 484203280 URNG output2: 589611434
U0 [47:0]: 000111001101110001011011000100000010001100100100 U1 [15:0]: 1100000110101010	U0 [47:0]: 000111001101110001011011000100000010001100100100 U1 [15:0]: 1100000110101010
`e = -2ln(u0)`	`e = -2ln(u0)`
Range reduction: x_e [48:0]: 1110011011100010110110001000000100011001001000000	Range reduction: x_e [48:0]: 1110011011100010110110001000000100011001001000000
Value of e [30:0]: 0000100.010111100101101011000001	Value of e [30:0]: 0000100.010111100101110010010001
`f = sqrt(e)`	`f = sqrt(e)`
Value of f [16:0]: 0010.0001011001110	Value of f [16:0]: 0010.0001011100001

`g0 =sin(2*pi*u1)`
`g1 =cos(2*pi*u1)`
	`g0 =sin(2*pi*u1)`
	`g1 =cos(2*pi*u1)`

g0 [16:0] = 10.111111111101001	g0 [16:0] = 10.111111111101001
g1 [16:0] = 00.000011001010000	g1[16:0] = 00.000011001001000

`x0 = f*g0`
`x1 = f*g1`
	`x0 = f*g0`
	`x1 = f*g1`

AWGN_0: 10010.00010110000 AWGN_1: 00000.00011010010	AWGN_0: 10010.00010110101 AWGN_1: 00000.00011010010

AWGN_0 (decimal):
-2.0859375
AWGN_1 (decimal):
0.1025390625
	AWGN_0 (decimal):
	-2.08837890625
	AWGN_1 (decimal):
	0.1025390625

Comparison of the Verilog and bit width optimization methods with real time data:

$$e = -2\ln(u_0),$$

$$e = 4.3654,$$

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

$$f = 2.0894,$$

$$g_0 = \sin(2\pi u_1),$$

$$g_0 = -0.9992$$

$$g_1 = \cos(2\pi u_1),$$

$$g_1 = 0.0408,$$

$$x_0 = f * g_0,$$

$$x_0 = -2.0876,$$

$$x_1 = f * g_1,$$

$$x_1 = 0.0853$$

From these results we can say that our design is correct and implemented fine while generating noise samples.

Conclusion:

10,000 Noise samples are recorded with the help of test bench and compared with bit width Matlab model. In MATLAB 10,000 samples are generated from the input vectors generated from Verilog.

References:

[1] Dong-U Lee, Member, John D. Villasenor Wayne Luk, Member and Philip H.W. Leong "A Hardware Gaussian Noise Generator Using the Box-Muller Method and its Error Analysis," *IEEE Trans*, computers, vol. 55, no 6, Jun. 2006