Gaussian Noise Generator
Core Specification

Guangxi Liu

guangxi.liu@opencores.org

Revision 1.0

Jan. 29, 2015

Revision History

Revision	Date	Author	Description
0.1	2014/8/4	Guangxi Liu	First Draft.
1.0	2015/1/29	Guangxi Liu	Revision 1.0

Contents

1 Introduction

1.1 Description

The *Gaussian Noise Generator* (GNG) core generates white Gaussian noise of standard normal distribution, which can be used to measure BER to extremely low BER levels ($\sim10^{-15}$). The core uses a 64-bit combined Tausworthe generator and an approximation of the inverse normal cumulative distribution function, which obtains a PDF that is Gaussian to up to 9.1σ.

The core was designed using synthesizable Verilog code and can be delivered as a soft-IP targeted for any FPGA device and ASIC technology. C/MATLAB models and corresponding test benches are also available.

1.2 Features

- Period of generated noise sequence is about 2^{176}
- Random distribution in the range of $\pm9.1\sigma$
- Noise is quantized to 16 bits with 5 bits of integer and11 bits of fraction
- Internal 64-bit uniform random number generator with configurable initial seeds
- Based on a piecewise polynomial approximation of the inverse normal cumulative distribution function
- High throughput, over 300 MHz clock rate and output sample rate in advanced FPGA
- Fully synchronous design using single clock
- Design optimized for Xilinx & Altera FPGA technology

1.3 Applications

- Communication system requiring accurate emulation of an AWGN channel
- Bit error rate measurement system

2 Algorithm

2.1 Overview

Most digital methods for generating Gaussian random variables are based on the transformation of uniformly distributed random variables. The most important methods for the design of hardware GNG are summarized below.

CLT

The central limit theorem (CLT) states that the sum of a sufficiently large number of independent uniformly distributed random variables will be approximately normally distributed. High accuracy GNGs need to sum up a large number of samples and the CLT method is not suitable for high-speed applications.

B-M Method

The Box–Muller (B-M) method has been widely used to generate Gaussian noise samples. This method is based on the transformation of two independent uniformly distributed random numbers, i.e., U(0,1), using elementary functions (sqrt(), ln(), and cos()/sin()).

Rejection–Acceptance Methods

The rejection–acceptance methods, basically the Polar method and the Ziggurat method, are the most important methods in software random number generation. Their main drawback is the use of conditional statements that lead to a nonconstant output rate.

Wallace Method

The Wallace method is based on the fact that linear combinations of Gaussian-distributed random numbers are also Gaussian distributed. This method avoids the evaluation of transcendental functions.

ICDF

The inversion method is based on the use of the ICDF of the Gaussian distribution to transform a uniformly distributed random variable x into a Gaussian variable y through y = ICDF(x). Using this method, it is possible to generate random samples for arbitrary distributions. This method is used as the design presented here.

2.2 CTG

Although traditional linear feedback shift registers (LFSRs) are often sufficient as a uniform random number generator (URNG), Tausworthe URNGs are fast and occupy less area. Furthermore, they provide superior randomness when evaluated using the Diehard random number test suite.

The combined Tausworthe generator (CTG) is used here follows the algorithm presented by [3][4], which generates a 64-bit uniform random number per clock and has a large period of $2^{176}(\approx 10^{53})$.

The parameters of generator are as follows (see Table 2 in [4]):

$$L = 64,\ J = 3,\ k = 176$$

$$(k1, k2, k3) = (63, 58, 55)$$

$$(q1, q2, q3) = (5, 19, 24)$$

$$(s1, s2, s3) = (24, 13, 7)$$

The algorithm is described in [3], and special process is used in generating valid initial state of generator. See relative codes of design.

2.3 ICDF

The inversion method generates Gaussian samples via the ICDF of the Gaussian distribution. A uniformly distributed random sample $x \in \mathrm{U}(0,1)$ is transformed into a sample y with the desired probability density function applying y = ICDF(x). The definition of ICDF is

$$\mathrm{ICDF}(x) = \sqrt{2}\,\mathrm{erf}^{-1}(2x-1)$$

The curve of it is shown in Figure 2.1.

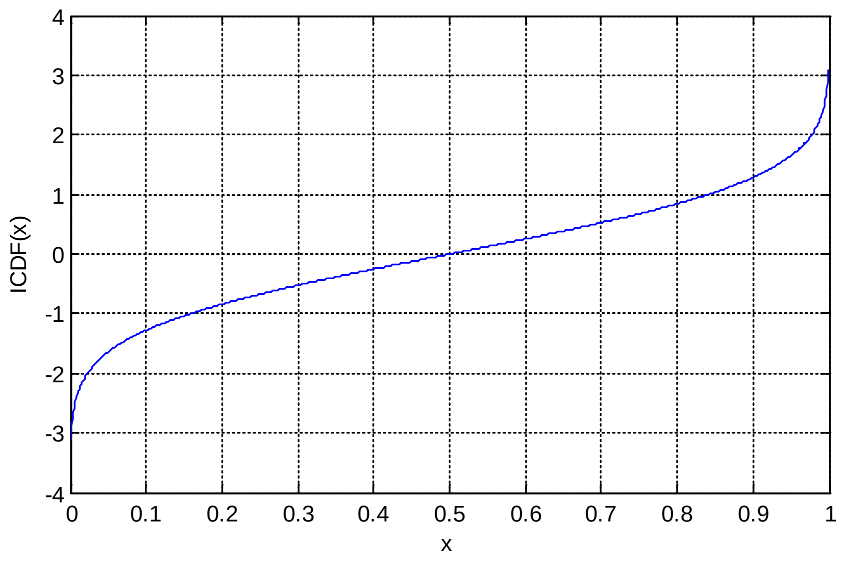

Figure 2.1 Curve of function ICDF(x)

It is highly nonlinear when x approaches 0 or 1. Besides, the curve is center symmetric with x = 0.5. So only the part of x > 0.5 is considered except sign.

The design uses a nonuniform segmentation scheme for the ICDF approximation. Typically, the most significant bits (MSBs) of the URNG output determine the segment and from the rest of the bits, the most significant ones are used as the input for interpolation inside the selected segment. So, we need to compute ICDF(u), u

being the value obtained if the bits of the URNG output are interpreted as an unsigned binary representation (64 bits here).

The architecture of nonuniform segmentation scheme for the ICDF approximation is similar to the one proposed in [6] for the hierarchical segmentation. The segmentation of the function leads to two basic blocks in the ICDF computation: segment selection and interpolation inside the selected segment. For interval (0, 0.5], P2S$_L$ segmentation schemes is used for the first pass and then US for the second pass.

The hardware computes ICDF(*u*'), *u*' being another uniform variable, which is different from *u*, created also from the URNG output bits. ICDF(*u*') can be computed more efficiently than ICDF(*u*), both being uniformly distributed variables with the same word length and, therefore, leading to the same output precision[1].

For each inner segment, a second order polynomial evaluation is performed, that is

$$y = (C_2 x + C_1) x + C_0$$

And we need to find the best coefficients C_0, C_1 and C_2 making the approximation error minimum. For real hardware implementation, these coefficients required further converted to fixed-point.

2.4 Performance

The error plot of each segment is shown in Figure 2.2. For each segment, only the maximum error between fixed-point and ideal result is shown.

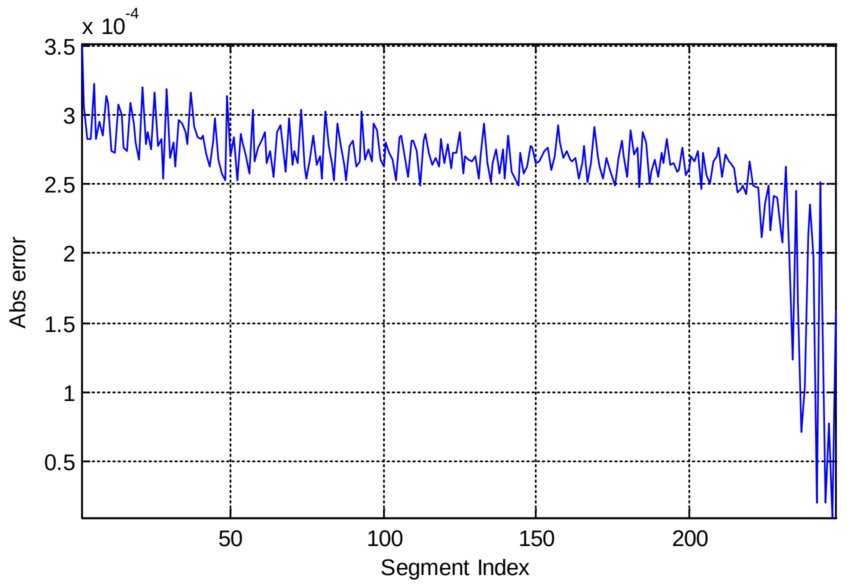

Figure 2.2 Error plot for each segment

From above plot, the maximum absolute error is about 0.00035, which is about 0.72 ulp (1 ulp = 2^{-11}). The result satisfies system accuracy requirement.

Figure 2.3 below shows the PDF of generated samples for a population of 10 million. The black solid line indicates the ideal Gaussian PDF.

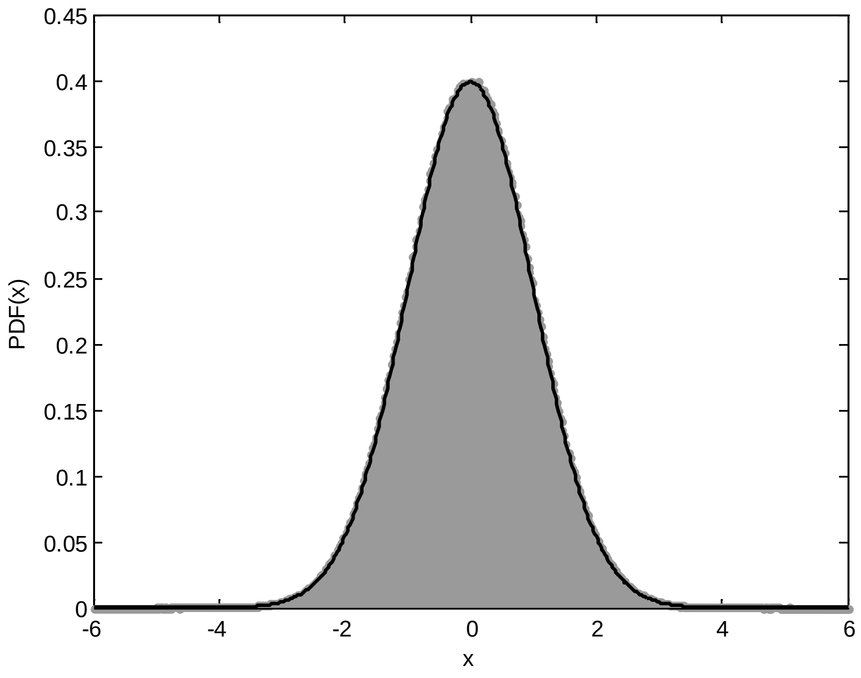

Figure 2.3 PDF of the generated samples for 10 million samples

We can see that the generated samples closely follow the true Gaussian PDF.

3 Implementation

3.1 Schematic Symbol

The schematic symbol for the core is shown in Figure 3.1.

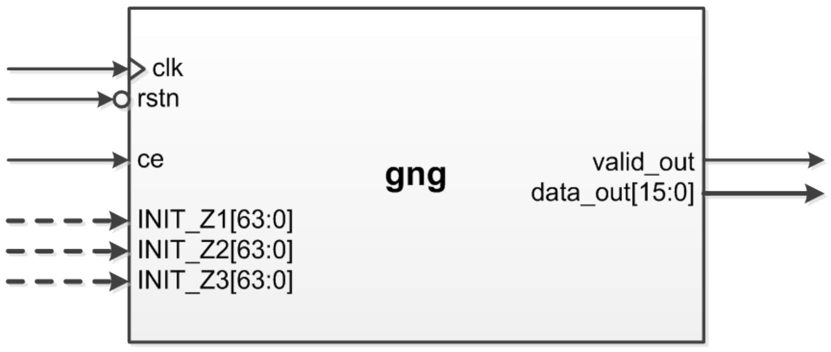

Figure 3.1 Core schematic symbol

3.2 Parameters

The parameters for the core are shown in Table 3.1.

Table 3.1 Core parameters

Name	Width	Description
INIT_Z1	64	Initial state value for CTG sub module 1.
INIT_Z2	64	Initial state value for CTG sub module 2.
INIT_Z3	64	Initial state value for CTG sub module 3.

3.3 Ports

The ports for the core are shown in Table 3.2.

Table 3.2 Core ports

Name	Width	Direction	Description
clk	1	Input	System clock.
rstn	1	Input	System synchronous reset, active low.
ce	1	Input	Clock enable.
valid_out	1	Output	Output data valid.
data_out	16	Output	Output data, fixed point format s<16,11>.

3.4 Structure

3.4.1 Top Module

The top module comprises module CTG and ICDF, as shown in Figure 3.2.

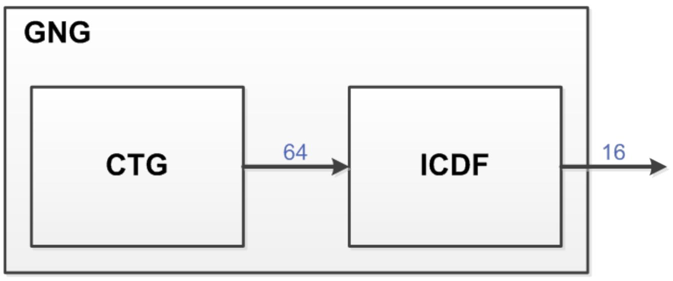

Figure 3.2 Top module architecture

3.4.2 CTG

The architecture of CTG is depicted in Figure 3.3. For simplicity the control signals are not presented.

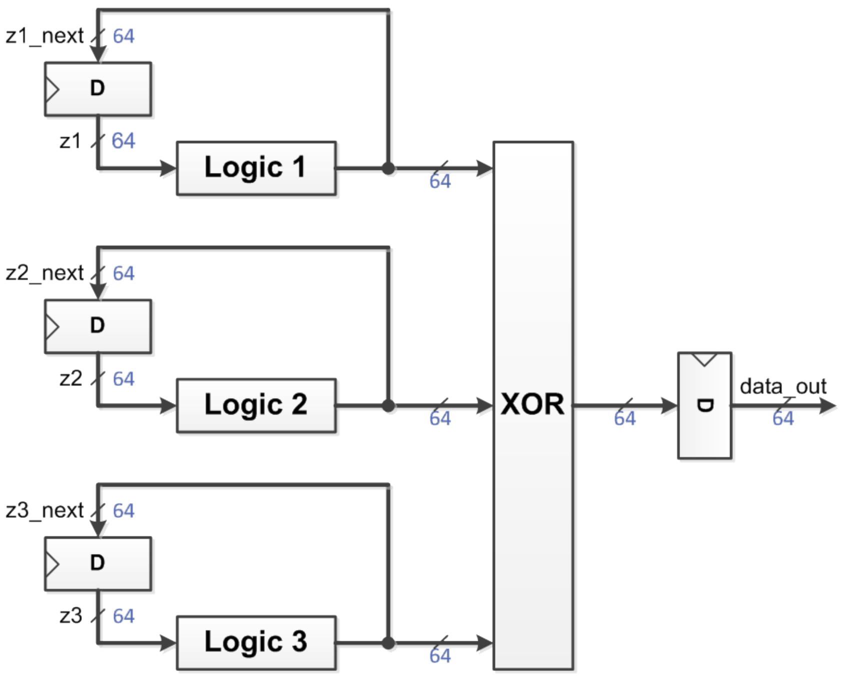

Figure 3.3 Architecture of CTG

The initial values of registers z1, z2 and z3 are precalculated. And combinatorial logic for each branch is (symbol $\oplus$ denotes the logical exclusive-or operator)

$$z1_{next} = \left(z1[39:1], z1[58:34] \oplus z1[63:39]\right)$$
$$z2_{next} = \left(z2[50:6], z2[44:26] \oplus z2[63:45]\right)$$
$$z3_{next} = \left(z3[56:9], z3[39:24] \oplus z3[63:48]\right)$$

3.4.3 ICDF

The architecture of ICDF is depicted in Figure 3.4. For simplicity the control signals and registers are not presented.

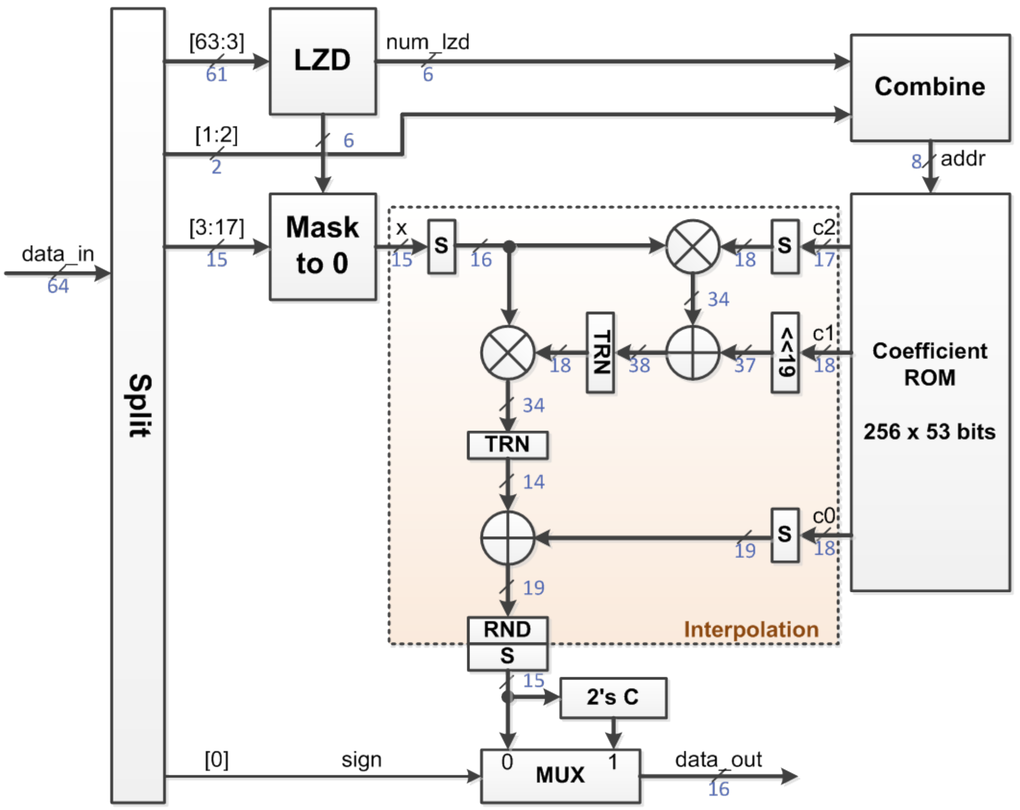

Figure 3.4 Architecture of ICDF

Where submodule S is sign bit extension, TRN is data truncation, and RND is data round. All multipliers and adders are signed operation.

Submodule LZD is to count number of leading zeros of 61-bit number. Here we choose the methodology proposed in [5]. The architecture of LZD is shown in Figure 3.5.

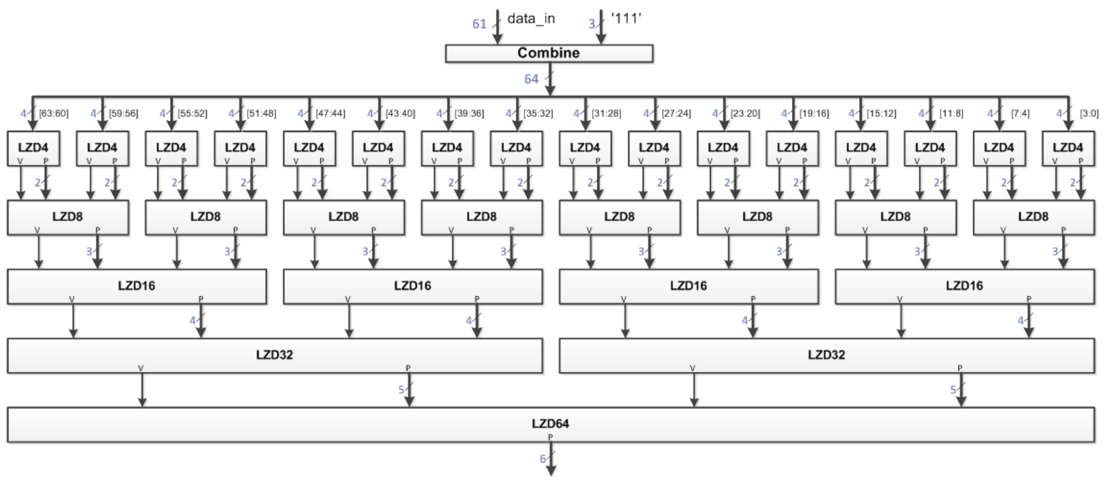

Figure 3.5 Architecture of LZD

Where the logic of LZD4 is as below

$$V = d[0] \mid d[1] \mid d[2] \mid d[3]$$

$$P = \{\sim(d[2] \mid d[3]), ((d[2] \mid d[3]) \; ? \sim d[3] : \sim d[1])\}$$

And the logic of LZD8 is

$$V = V[0] \mid V[1]$$

$$P = \{\sim V[1], ((V[1]) \; ? \; P[1] : P[0])\}$$

And so on. Note that in the last level LZD64 only *P* is used.

3.5 Timing

The typical timing diagram is in Figure 3.6.

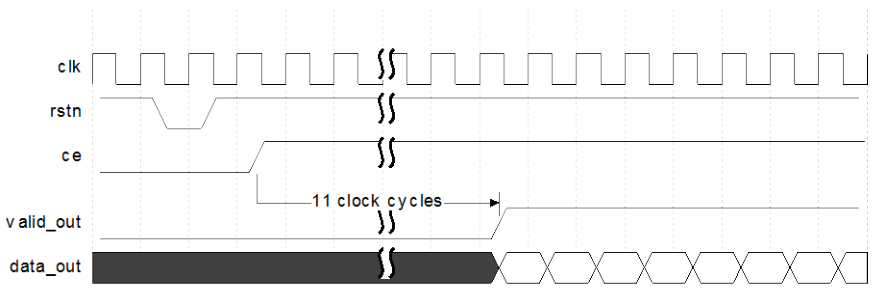

Figure 3.6 Core timing diagram

The signal *valid_out* asserts after 11 clock cycles when signal *ce* asserts. If *ce* is tied high, *data_out* is output every clock.

4 C/MATLAB Model

4.1 C codes

We use C codes to build the functional model. The file names and their descriptions are listed in Table 4.1 below.

Table 4.1 C codes list

File Name	Description
taus176.h	Header file for maximally equidistributed combined Tausworthe generator.
taus176.c	Implementation file for maximally equidistributed combined Tausworthe generator.
icdf.h	Header file for piecewise polynomial approximation of inverse of the normal cumulative distribution function.
icdf.c	Implementation file for piecewise polynomial approximation of inverse of the normal cumulative distribution function.

These codes are also used as C MEX codes in MATLAB model (See next section).

4.2 C MEX and MATLAB codes

To accelerate MATLAB simulation speed, C MEX method is used. The file names and their descriptions are listed in Table 4.2 below.

Table 4.2 C MEX/MATLAB codes list

File Name	Description
ctg_seed.c	C MEX file for generate Combined Tausworthe Generator seed.
ctg_seed.m	MATLAB helper file for ctg_seed function.
ctg_gen.c	C MEX file for generate Combined Tausworthe number.
ctg_gen.m	MATLAB helper file for ctg_gen function.
icdf_gen.c	C MEX file for generate inverse of the normal cumulative distribution function.
icdf_gen.m	MATLAB helper file for icdf_gen function.
build_mex.m	Build above C MEX files.
test_gng.m	Test Gaussian noise generator.

Note that 64-bit integer data type is used in C MEX file, and VC++ should be used as compile in MATLAB mex setup. Typical versions are MATLAB R2011b and Visual C++ 2010.

5 Synthesis

5.1 Xilinx FPGA

FPGA Device is Virtex-6 XC6VLX240T-2ff1156 and implementation tool is Xilinx ISE 14.7. Results are shown in table 5.1, and are slightly better than [2].

Table 5.1 Implementation results (place and route) for Xilinx FPGA

Number of occupied Slices	97
Number of RAMB36E1	1
Number of DSP48E1s	2
Maximum frequency	311.8 MHz

5.2 Altera FPGA

FPGA Device is Stratix IV GX EP4SGX230KF40C3 and implementation tool is Altera Quartus II 11.1. Results are shown in table 5.2, and are slightly better than [2].

Table 5.2 Implementation results (place and route) for Altera FPGA

Total LABs	34
M9K blocks	2
DSP block 18-bit elements	4
Maximum frequency	376.8 MHz

5.3 ASIC

ASIC technology library is SMIC 55nm LL and synthesis tool is Synopsys Design Compiler 2012.06-SP3. Results are shown in table 5.3.

Table 5.3 Implementation results for SMIC library

Area	16739.52 μm^2
Equivalent gates	13078
Target frequency	400.0 MHz
Power	4.2133 mW

The smallest area (1.28μm^2) of an NAND2 gate is used as the base in equivalent gates calculation. Notice that the results are not the best due to the core is not specially optimized for ASIC.

6 Simulation

6.1 Test Bench

To verify the correction of the core, a SystemVerilog code (tb_gng.sv) is written to do it. First, we use `ctg_seed(1)` command in MATLAB to generate the core parameters *INIT_Z1*/ *INIT_Z1*/ *INIT_Z1*, which are just the default parameters in code gng.v.

The design gng is instantiated as a design under test (DUT) in test bench. The basic function of test bench is: generate N = 1000000 active signal *ce* to the DUT, and record its output data to a data file (gng_data_out.txt).

6.2 Simulation Result

By running the simulation script file run.do in ModelSim, the result data file gng_data_out.txt can be generated. Meanwhile by running the m file test_gng.m in MATLAB, the variable x is gotten. Then Import the data in gng_data_out.txt into MATLAB and compare it with x. The comparison result should be all equal, which means the function of RTL exactly matching that of MATLAB model.

References

[1] R. Gutierrez, V. Torres, and J. Valls, “Hardware architecture of a Gaussian noise generator based on inversion method,” IEEE Trans. Circuits Syst. II, vol. 59, no. 8, pp. 501–505, 2012.

[2] DiComLab, “IP cores: AWGN Generator,” http://www.gised.upv.es/awgn.html.

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

[4] P. L’Ecuyer, “Tables of Maximally Equidistributed Combined LFSR Generators,” Math. Computation, vol. 68, no. 225, pp. 261-269, 1999.

[5] V. Oklobdzija, “An Algorithmic and Novel Design of a Leading Zero Detector Circuit: Comparison with Logic Synthesis,” IEEE Trans. VLSI Systems, vol. 2, no. 1, pp. 124-128, 1994.

[6] R. C. C. Cheung, D. Lee,W. Luk, and J. Villasenor, “Hardware generation of arbitrary random number distributions from uniform distributions via the inversion method,” IEEE Trans. VLSI Systems, vol. 15, no. 8, pp. 952–962, 2007.