

# **Inexpensive Parallel Random Number Generator for Configurable Hardware**

**Mustapha Abdulai**

**Arizona State University, Tempe Arizona**

**Summer Undergraduate Program in Engineering Research at Berkeley**

**(SUPERB) 2003**

**Faculty Mentor: Prof. Bob Brodersen**

**Graduate Mentor: Danijela Čabrić**

**Berkeley Wireless Research Center, College of Engineering**

**University of California, Berkeley**

## **Abstract**

**In this paper we study a hardware efficient implementation of a uniform random number generator based on Cellular Automata. This random number generator can be used to create Additive Gaussian White Noise and other noise sources in wireless application simulations. The algorithm used to generate random numbers can be realized using simple digital circuits and memory, and we implemented it on an emulation platform based on large FPGA array. Our goal was to investigate periodicity and correlation properties of the generator as a function of initialization sequence. We have developed the testing strategy that detects the period and locking states of the generator with a simulation time at least two orders of magnitude shorter than in a software implementation.**

*“The generation of random numbers  
is too important to be left to chance”*

*R. Coveyou*

## **1.0 Introduction:**

Random number sequences can be found in a large number of applications that include cryptography, unbiased sampling in Monte Carlo simulations, and imitating stochastic natural processes. Good random number generators are required to provide true source of randomness in applications where one has to model a physical process. For example, in a simulation of a wireless communication system there are several noise sources that need to be included. The most commonly used is Additive White Gaussian Noise, but there other analog impairments like phase noise, clock jitter, etc. that in a complete system analysis must be taken into account. In general, a communication system is characterized through a bit error rate (BER) vs. signal to noise ratio (SNR) curves, which require long and highly computational extensive Monte Carlo simulations. Therefore, it is desirable to have a fast preferably hardware implementation of a system that can produce BER vs. SNR curves in real time. The type of random number generator needed for wireless applications simulations must be very fast but furthermore highly implementation efficient. In this project we study the random number generators that can be implemented on a FPGA using simple logic blocks and have a satisfactory performance for simulation of wireless communication systems.

A wide variety of ingenious methods have been designed to generate random numbers. Most generators are software based and generally fall into one of these three categories:

- ❖ Linear Congruential Generator – Usually of the form  $X_i = (aX_{i-1} + b) \bmod m$ , where  $a$ ,  $b$ , and  $m$  are constants. This random

number generator requires integer recursion thus it is expensive in hardware. Also, this generator is efficiently predictable when the constants  $a$ ,  $b$  and  $m$  are known [1].

- ❖ Lagged Fibonacci Generator – Generally of the form  $X_i = (X_{i-r} \otimes X_{i-s}) \bmod m$ , where  $r$ ,  $s$  and  $m$  are constants,  $r > s$ , and  $\otimes$  could be any of the following binary operators,  $+$ ,  $-$ ,  $\times$ ,  $xor$ . This generator requires the initial data set  $X_1, X_2, \dots, X_r$  and depending on the choice of the binary operator might require integer recursion.
- ❖ Linear Feedback Shift Register Generator – They are based on the theory of primitive polynomials in the form  $X^p + X^q + 1$ . Given such a primitive polynomial and  $p$  binary digits,  $X_0, X_1, X_2, \dots, X_{p-1}$  then  $X_k = X_{k-p} \oplus X_{k-p+q}$ , where  $\oplus$  is the XOR operator. This generator has been shown to exhibit lattice structures in the random number sequence generated [1].

On the other hand, there are other non-software based ways of generating random numbers. Some examples are:

- ❖ Physical devices – Random number generators based on physical devices usually sample some physical event (like vibrations or temperature) and generate the random numbers using A/D converters. The random number sequences generated by physical devices are non reproducible unless they are stored in memory.
- ❖ Cellular Automata Random Number Generator – Based on linear finite state machine each consisting of one-dimensional array of cells, each cell

is allowed to communicate only with its immediate neighbor based on a rule.

In general one considers a sequence of numbers random if no general pattern can be discerned, no prediction can be made about it, and no description can be found [2]. However, on a computer, random numbers are generated by the successive iteration of a definite transformation. Thus for number sequences generated by a computer the most random sequence are those that are least predictable.

### **1.1 Organization of This Paper**

The remainder of this paper is organized as follows. Section 2 provides implementation details of the random number generator on the Berkeley Emulation Engine (BEE) FPGA and Simulink ® system simulation. Section 3 discusses one application of the random number generator, the Additive White Gaussian Noise Generator. Section 4 discusses the test performed on the random number sequences generated to check for any correlation effects. Section 5 presents the results of our project in relation to the periodicity of the random number generator and its locking states. Section 6 concludes with the status of this project and future considerations.

### **2.0 Implementation**

Random number sequences are generated either by a deterministic rule on a computer (pseudo-random numbers) or by sampling certain random natural properties (like noise in a room, etc). For our application the best random number sequence is one that is very uniform, has very little correlation effects, requires minimal hardware, easy to use, and above all the “random sequence” must be completely reproducible.

Recent studies have shown that Cellular Automata is a promising technique for generating uniform random numbers. Cellular Automata can be viewed as computational systems, whose evolution process information is contained in their initial configurations. It can also be considered as a discrete approximation of partial differential equations [2]. The generation of random numbers using Cellular Automata has been proven to be at least as random as the more established random number generators [2,3]. Moreover, this generator has the advantage of being highly parallel and thus easily scalable at relatively very little hardware cost.

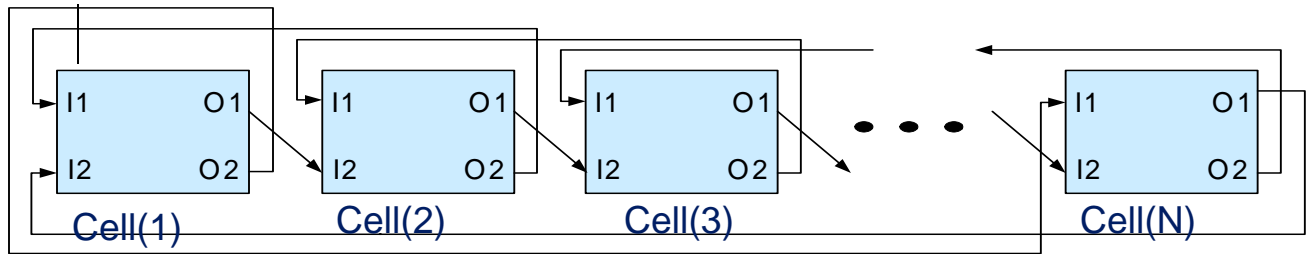
For random sequences produced by Cellular Automata there are no known mathematical shortcuts to predicting the values and the computational power needed to discern a pattern is usually far greater than the power used to generate the sequences [2].

The Random number generator chosen for this study is based on a one dimensional cellular automaton with the following connecting rule:

$$\begin{aligned}
 &I1[1] = O2[2] \\
 &I2[1] = O1[N] \\
 &\text{for } (k = 2, k < N, k++) \\
 &\{ \\
 &\quad I1[k] = O2[k + 1] \\
 &\quad I2[k] = O1[k - 1] \\
 &\}
 \end{aligned}$$

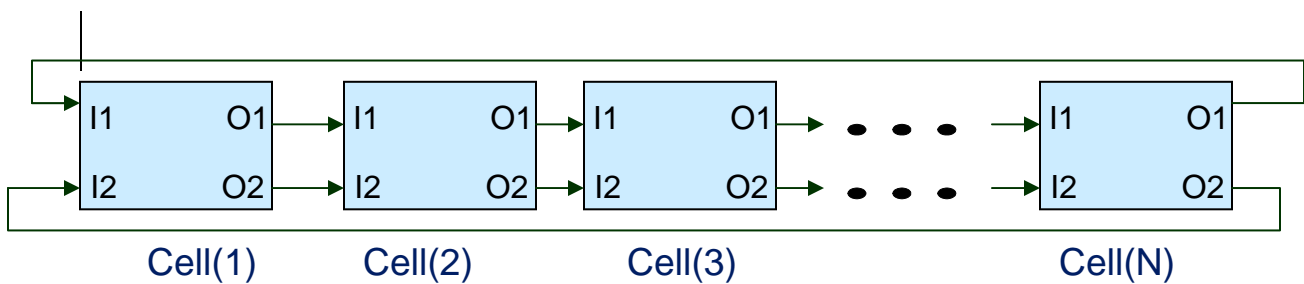
Where O1 and O2 are the outputs and I1 and I2 are the inputs. Each random number-generating cell can then be connected with N-1 other cells where N is the desired precision of the output. One way of to connect the random number generating cell is to connect the first output O1 from the first cell to the second input I2 or the second cell then connect the second input O2 from the first cell to the first input I1 of the previous cell (i.e. the Nth cell) as shown in Fig. 1. Another way of connecting the cells is to connect the outputs O1 and O2 from cell (1) to

the inputs I1 and I2 of cell (2) respectively as shown in Fig. 2. Other ways of connecting the cells are possible.



Basic Cell Architecture (a)

Fig. 1. First method for connecting RNG cells. Output O1 is connected to input I2 of the adjacent cell whiles output O2 is connected to the input I1 of the previous cell.



Basic Cell Architecture (b)

Fig. 2 Second method for connecting RNG Cell. Outputs O1 and O2 are connected to inputs I1 and I2 of the next cell.

On FPGA array, different sites or groups of sites in the cellular automaton are assigned to different processors. They are then updated independently (though synchronously), using the same instructions, and with only local communications, this ensures that this random number generator is at least twice as fast as a software-implemented version.

## 2.1 The Random Number Generator

As shown in fig. 1, the random number generator is implemented using XORs, registers and a RAM. One of the inputs, I1, is XORed with the output from the

RAM and the second output bit, O2 then delayed one clock cycle to form the first output, O1. The second input is XORed with the output from the RAM and delayed one clock cycle to form the second output bit, O2. The first output is feedback into the RAM.

The outputs from eight RNG cells can be concatenated to form an eight-bit output on every clock cycle. The precision of an N-bit RNG can be easily changed by adding or removing RNG cells.

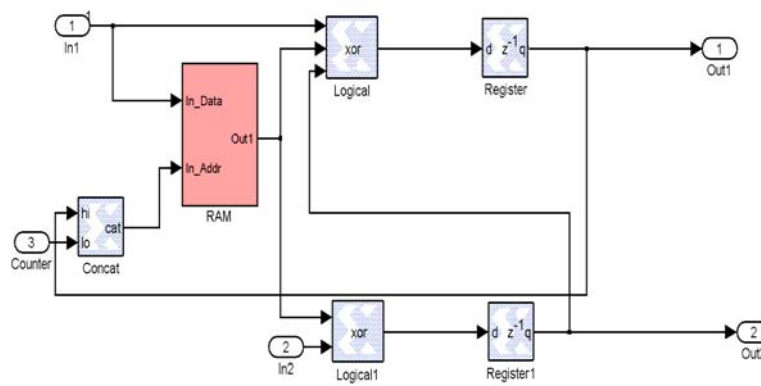


Fig.3 Random Number Generating Cell, the outputs Out1 and Out2 are stored in the 1X32 RAM. The output from the RAM is XORed with the input and delayed one clock cycle to form a new output.

The configuration used to produce two 8-bit random numbers per clock cycle in this project is shown in fig. 4, however other arrangements of the RNG cells are possible.

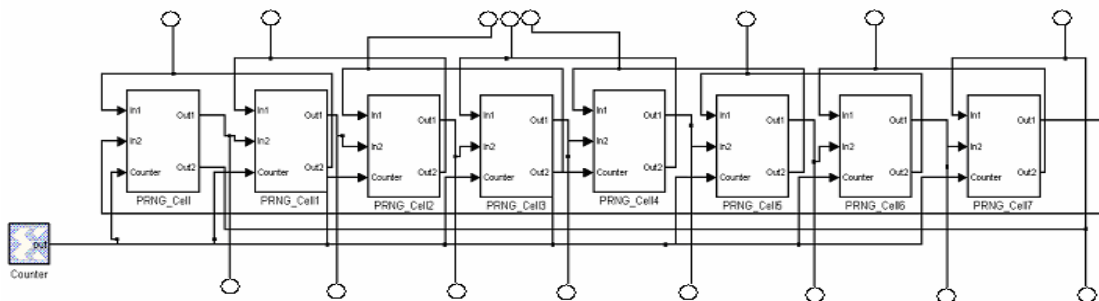


Fig. 4. Cascading of 8 random number generating cells to form an 8-bit random number generator. The outputs at the top and bottom can be connected to a bus either serially or in parallel (parallel being more efficient in terms of hardware cost) depending on the application for which the random number generator is needed

## 2.2 RAM

The RAM used in this implementation is a 5-bit address and 1 bit wide RAM cell, the MSB of the address into the RAM is the previous output bit from O2 and the remaining four bits are generated by a 4-bit counter. The RAM cell contains 32 registers in the addressable register. The initial state of these registers serves as the seed for the random number generator.

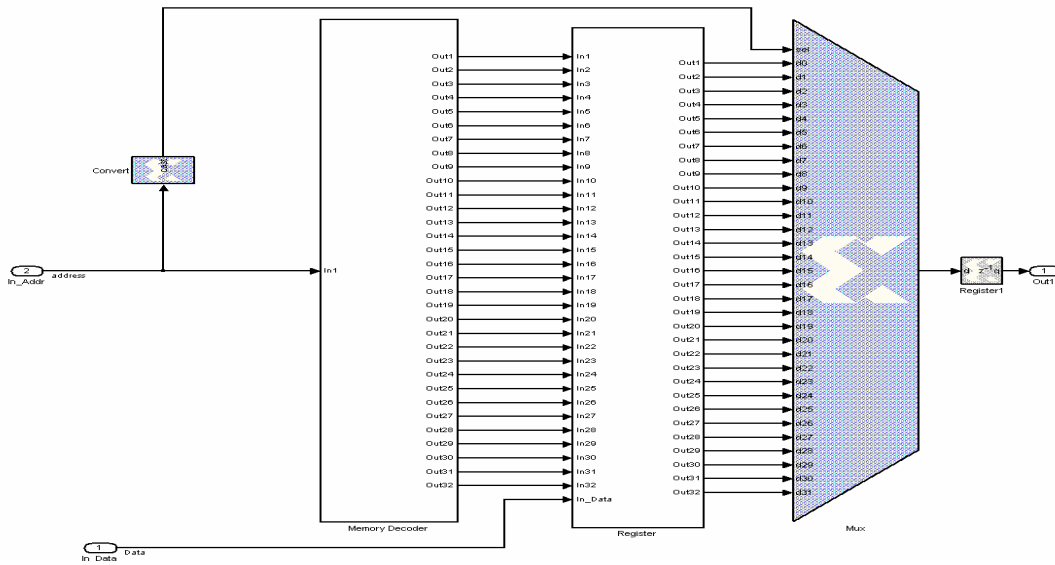


Fig. 5 RAM cell that can store 1-bit in one of 32 address spaces. The address line into the RAM cell is a 4-bit counter concatenated with the previous output of the random number generator as the MSB.

## 3.0 Application – Additive White Gaussian Noise Generator

In wireless communication system simulations it is often necessary to add analog impairment models to help simulate the effect noises found in real wireless communication systems. One main source of noise in wireless communication systems is Additive White Gaussian Noise (AWGN). To create the AWGN generator 8 RNG cells are connected together to produce two 8-bit random numbers. The outputs are then manipulated according to the formula shown below to produce Gaussian noise.

$$\begin{aligned}
 f(x_1) &= \sqrt{-2 \ln(x_1)} \\
 g_1(x_2) &= \cos(2\pi x_2) \\
 g_2(x_2) &= \sin(2\pi x_2)
 \end{aligned}
 \longrightarrow
 \begin{aligned}
 n_1 &= f(x_1)g_1(x_2) \\
 n_2 &= f(x_1)g_2(x_2)
 \end{aligned}$$

Where  $x_1$  and  $x_2$  are the outputs from the RNG and  $n_1$  and  $n_2$  are the AWGN generator outputs. Below in fig. 3 is the BEE FPGA implementation of the AWGN.

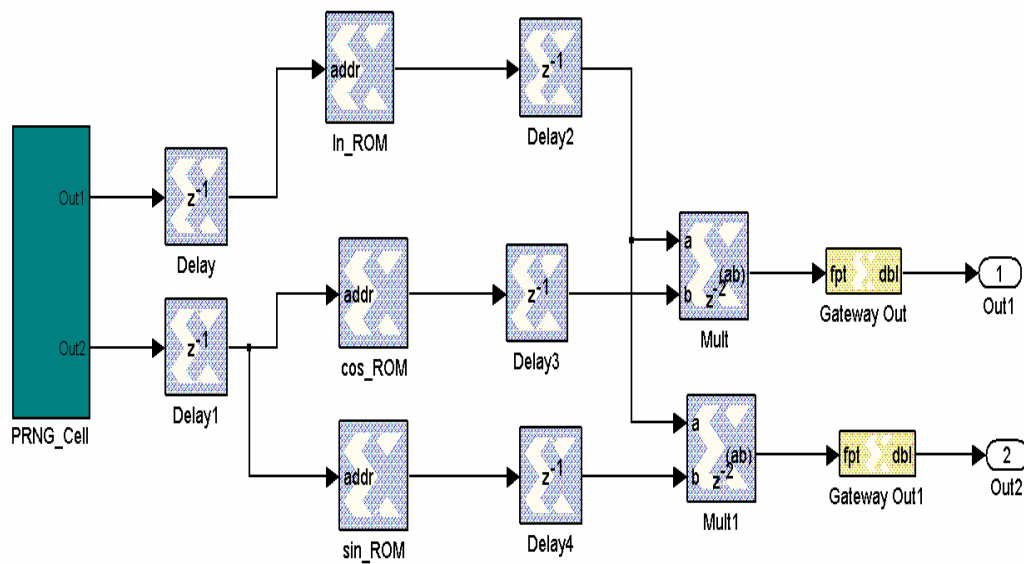


Fig.6. Implementation of Additive White Gaussian Noise Generator in BEE. Two 8-bit outputs from the RNG are used to create two set of Gaussian noise to be used in wireless communication systems simulations.

#### 4.0 Testing

There are no real tests to determine if a sequence of numbers is truly random, however there are several test that we can use to convince ourselves that a sequence might not be random enough a particular application. The type of random numbers sequences needed to create Additive White Gaussian Noise for wireless application simulations needed to be uniformly distributed, have a relatively flat spectrum and also have very little correlation effects.

**4.1 Correlation Effect** – In most sequences generated by a definite transformation correlation effects exists between the individual numbers in the sequence or between the sequences. To test for correlation effects we analyzed the autocorrelation coefficients. Let  $\{r_i, i = 0,1,\dots, N\}$  be a sequence of random numbers. Then the autocorrelation coefficients

$$\rho_k = \frac{\sum_{i=k}^N (r_i - \mu)(r_{i-k} - \mu)}{(\sum_{i=k}^N (r_i - \mu)^2 \sum_{i=k}^N (r_{i-k} - \mu)^2)^{\frac{1}{2}}}$$

for  $k = 1,2,\dots, N$  and  $\mu = \sum_{i=0}^N r_i / N$  measures the statistical independence of the sequences  $\{r_i, i = 0,1,\dots, N - k\}$  and  $\{r_{i-k}, i = k, k + 1,\dots, N\}$ . Simple correlations can be easily detected when the correlation coefficients differ significantly from zero [4].

**4.2 Spectral Analysis** – One very important characteristic of the random number generator needed for the Additive White Gaussian Noise Generator is for it to produce random numbers that have a relatively flat spectrum in the frequency domain. To check for this characteristic the graph of the discrete Fourier transform of the sequence is plotted against the power. The graph is then visually checked for spikes in the frequency, too many such frequency spikes would make the random number generator unsuitable for use in this application.

**5.0 Results** –Our approach was to compare this RNG with some standardized software based RNGs. We choose to use the Linear Congruential Generator as a basis for our comparison since it is the most widely used software based random number generator. The LCG was implemented in using 69069, 1, 256, and 667790 for  $a, b, m$  and  $X_0$  respectively. As shown in fig. 7 the 8-bit random numbers produced by the Cellular Automata had significantly fewer correlation effects.

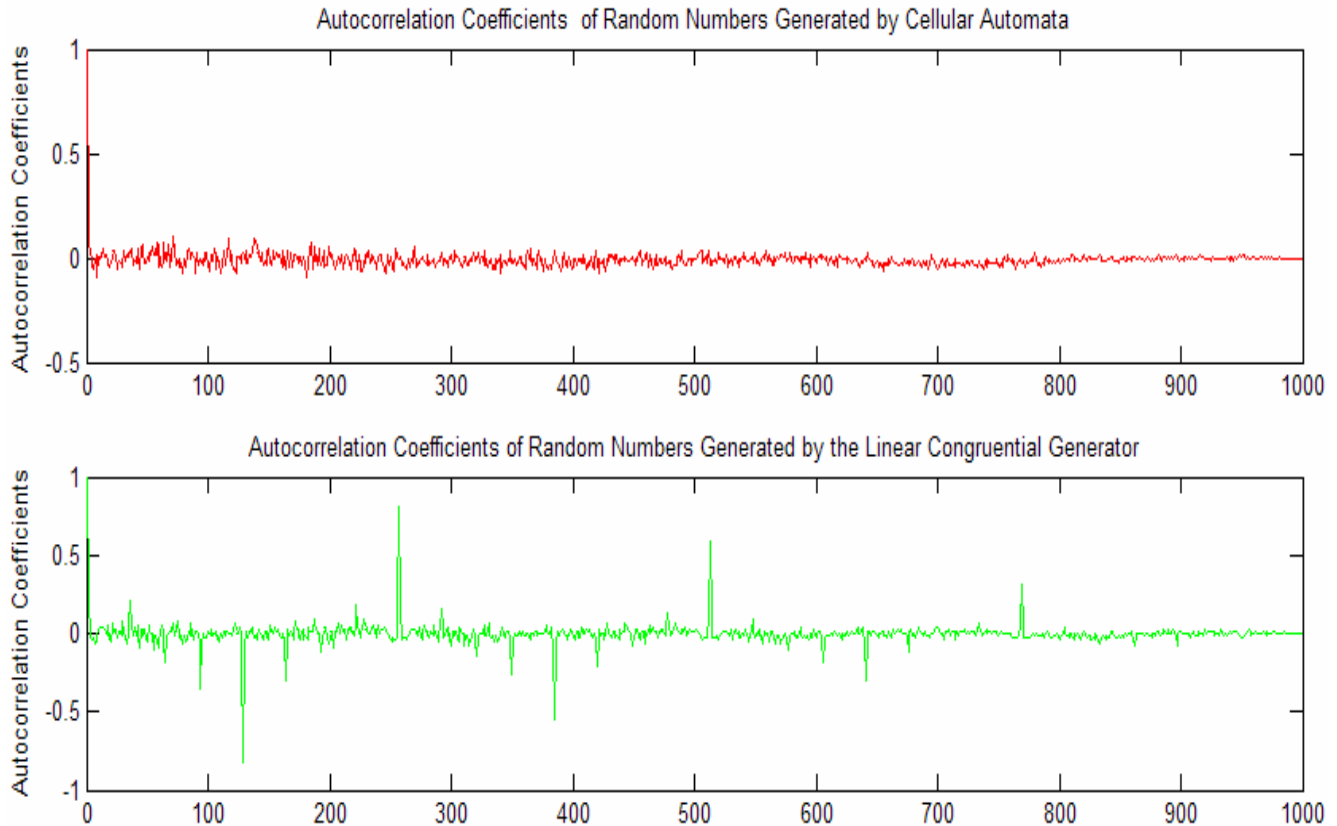


Fig. 7. Graph showing autocorrelation coefficients for random 8-bit random numbers generated using Cellular Automata (top graph) and 8-bit random numbers generated using the Linear Congruential Generator (lower graph). Significant deviations from zero in the autocorrelation coefficients suggest correlation effects between the numbers in the sequence.

The cellular automata based RNG was implanted and tested on the large FPGA array based emulation engine called Berkeley Emulation Engine (BEE). Broad variety of initial states is used for the registers in the RAM. We observed the following results. When the initial states were all zeroes the RNG did not produce any random numbers. When the registers were initialized with all ones the RNG produced random sequences with period of  $\sim 2^{10}$ . When the registers were initialized with alternating ones and zeroes the random numbers generated had a period of  $2^{25}$ , and when the registers were initialized randomly, the random sequence produced had a period of  $\sim 2^{28}$ .

The Implementation of the RNG on the BEE FPGA array was found to be at least 100 times faster than the software based Linear Congruential Generator. The

Cellular Automata based generator produced  $2^{28}$  random number in less than 6 seconds (running at 50 MHz), while the software based Linear Congruential generator produced (implemented in Matlab on a 2000 MHz WinXP machine with 512 MB RAM)  $2^{28}$  random numbers in  $\sim 1216$  seconds.

On the BEE FPGA the Cellular Automata based RNG was found to be highly efficient, requiring only 3% of the available RAM, 1% of LUTs (look up tables) and 3% of the available Slices.

<b>Logic Utilization</b>	
Number of Slice Flip Flops:	472 out of 38,400 1%
Number of 4 input LUTs:	570 out of 38,400 1%
<b>Logic Distribution</b>	
Number of occupied Slices:	687 out of 19,200 3%
Total Number 4 input LUTs:	639 out of 38,400 1%
Number of Block RAMs:	1 out of 160 1%
Number of GCLKs:	2 out of 4 50%
Total equivalent gate count for design:	29,915
Peak Memory Usage:	117 MB

Table 1. Area utilization of Cellular Automata based RNG on BEE FPGA.

Fig. 8 compares the spectral characteristic of the two RNGs. As it can be observed in the figure, there are some frequency components in the spectrum, but the RNG based on the cellular automata has less peaks, and the height of the peaks is smaller, thus showing another advantage over software based implementation of RNG.

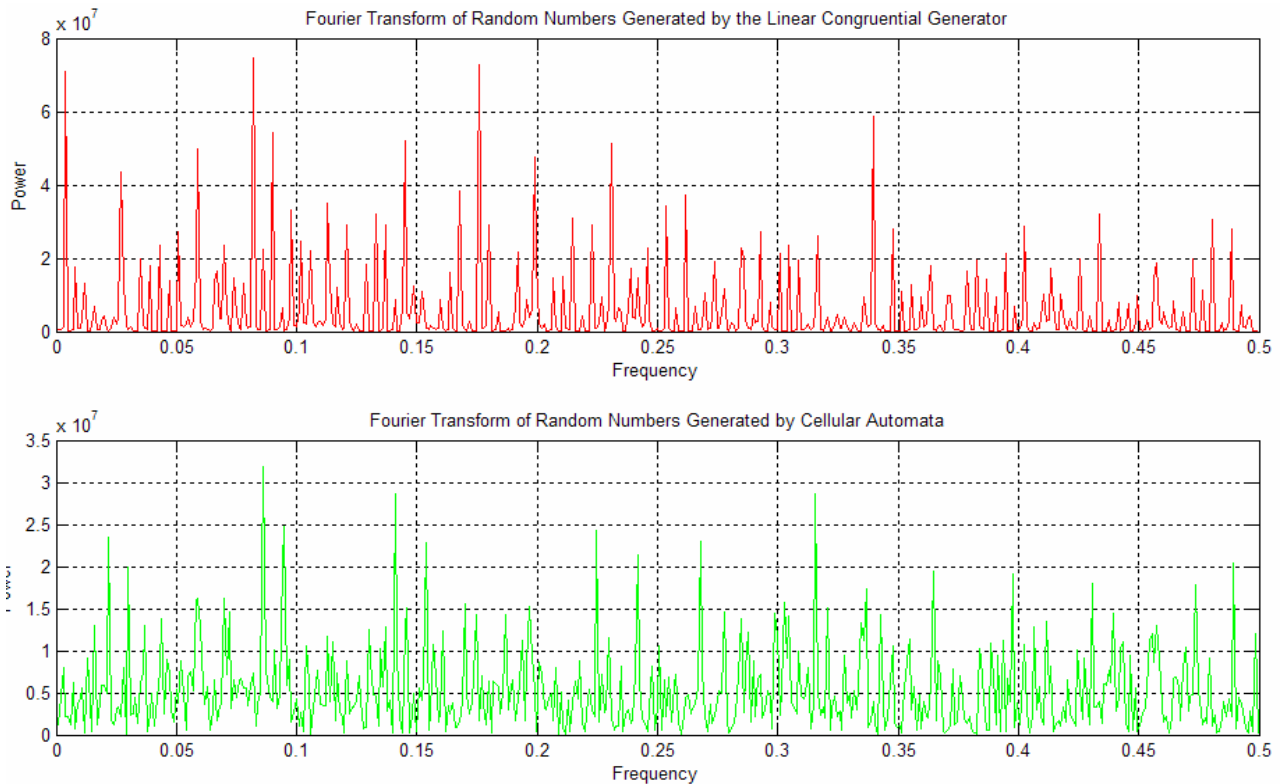


Fig. 8. Graph showing the distribution of 8-bit random numbers generated by the Linear Congruential Generator (top graph) and by Cellular Automata (lower graph) in the frequency domain. The y-axis represents the power within the signal. The wider the spread in the power distribution in frequency the domain, the less uniformly distributed the random sequence.

## 6.0 Future Considerations

RAM depth would increase the “randomness” and periodicity of the RNG implemented in this paper. This increase should be roughly linear such that a 1x64 RAM should have a period of about  $2^{56}$ . This register in the RAM of the RNG cell should be initialized with a best 32 bit numbers that ensure the highest quality random numbers as output, thus all  $2^{32}$  possible initializing numbers have to be tested to see which numbers produce random sequences of the greatest length.

**Acknowledgements** – This project would not have been possible without the help of many people. First I would like to thank my graduate mentor and faculty advisor, Danijela Čabrić and Prof. Bob Brodersen, who saw to it that my eight weeks in this program was filled with technical excitement and growth. Special thanks go to the administrative staff of the SUPERB program, especially Erika Tate, Marie Mayne and Sheila Humphreys, for seeing to it that all transactions went smoothly. Special thanks go to the technical and administrative staff at the BWRC, especially Kevin Zimmerman and Tom Boot, who saw to it that I was always comfortable. Lastly and most importantly, this program and many more like this would not occur without funding thus the author wishes to thank the NSF and UCB for all their monetary.

## **References**

- [1] I. Vattulainen, K. Kankaala, J. Saarinen, and T. Ala-Nissila, A Comparative study of pseudorandom number generators, *Computer Phys. Comm.* 86 (1995) 209-226
- [2] S. Wolfram, Random Sequence Generation by Cellular Automata, *Advances in Applied Mathematics*, 7 (June 1986) 123-169
- [3] Wikipedia, Pseudorandom Number Generators, [http://wikipedia.com/wiki/Pseudorandom\\_number\\_generator](http://wikipedia.com/wiki/Pseudorandom_number_generator) (2003)
- [4] J. Ackermann, U. Tangen, B. Bodekker, J. Breyer, E. Stoll, J.S. McCaskill, Parallel random number generator for inexpensive configurable hardware cells, *Computer Phys. Comm* 140(2001) 293-302