I J C T A, 9(18) 2016, pp. 9011-9017

Analysis of Pattern Generation and Randomness for LFSRs

Atul K. Srivastava*

ABSTRACT

The main goal of genetic programming is to get best pattern in terms of randomness. There are various methods for random pattern generation. The patterns generated in this paper are pseudo-random in nature. Different types of linear feedback shift register (LFSR) are used to generate pseudo-random patterns and test their randomness by various testing algorithms. Random patterns are compared according to their p-value which is achieved from different testing algorithms. Hence, the goal is to get the best pattern in terms of randomness. The objective of this paper is to explore statistical tests for detection of randomness in a true random number generator that may be used in designing genetic processor.

Keywords: Genetic programming, Random number generator, Pattern generation, Randomness, LFSR.

I. INTRODUCTION

The objective of VLSI systems is constrained by different parameters. Some applications are real time and need the hardware as less as possible on a chip. There are numerous applications in VLSI that are optimized using different methods. One such method, popular nowadays is genetic programming (GP). The genetic programming consists of various modules. The most important module and initial step is generation of initial population or solution space. Hardware produces randomly streams of bits that work as initial population [1]. There are different methods of producing these random streams of bits. All applications of VLSI technology need the implementation of genetic programming modules either in the form of hardware or software. Since hardware is faster than software, therefore those modules that are simpler can be realized on hardware whereas other can be on software, depending on the complexity of applied genetic functions.

Most popular method of hardware realization is LFSR. There are different size and kind of LFSR that produces streams of random numbers. Since the real time applications using genetic programming needs the size of hardware least possible, therefore in this paper, different methods of LFSR have been implemented and detailed analysis on the bit pattern generation using these LFSR has been performed to show which minimum size of LFSR needed to generate bit streams using different test methods. The LFSR of a defined size which passes the random tests may be selected for a particular application using genetic programming. Hence the minimum sized LFSR can be chosen and realized that shall consume minimum area on the chip and henceforth the objective is met.

Many applications that search for an optimal result before implementation require some random values generator. These may be for application implementing hardware synthesis using genetic programming, engineering design. Applications like encryption and code breaking that needs high combination of random values. Robotics, computer gaming, sampling, decision making and aesthetics also requires some form of random number generator for the better results.

* Department of Electronics and Communication Engineering, Jaypee Institute of Information Technology, NOIDA, India, *E-mail: atul.srivastava@jiit.ac.in*

II. RANDOM NUMBER GENERATOR TYPES

Basically there are two kind of method to produce random sequences – true random number generators (TRNGs) and pseudorandom number generators (PRNGs). The major difference between the two types is that TRNGs perform sampling of a source whereas PRNGs use a deterministic algorithm to generate random numbers [2]. A brief overview of TRNGs and PRNGs is given below. In the proposed work pseudorandom numbers are generated. Pseudorandom patterns are not really random in nature but look so. If an individual finds the key and the algorithm, then the person can predict the forthcoming pattern which is not the case for true random numbers.

2.1. True Random Number Generators

It is desired that a random number generator should generate numbers that are random in real sense. The generated random samples are processed through a processor to produce a sequence of random numbers [3]. True RNGs refer to physical RNGs and should not be taken as completely random because they are called true. True random numbers are by definition entirely irregular. The use of a refinement process is usually desirable to overcome any deficiency in the random source that results in the creation of non-random numbers. The process that is taking place naturally may be one kind of random source. This may be any emission from planet [4]. Randomness may be in melting of ice at the polar region. Noise generated in a solid state device

2.2. Pseudorandom Number Generators

Pseudorandom numbers are not rigorously random; they are predictable, i.e., they are figure out using a statistical algorithm. If the algorithm and the starting point are known, then the numbers generated are predictable. A deterministic formula could generate a random sequence seems like a disagreement. The main principle with PRNGs is to obtain sequences that perform as if they are random [5]. The output sequences of many PRNGs are statistically impossible to differentiate from fully random sequences and inconsistent, PRNGs often appear to be more random than random numbers obtained from TRNGs. By their characterization, however, the utmost length of the progressions produced by all these algorithms is restricted. These progressions are reproducible, and thus can be random only in narrow sense [6].

2.3. Comparison of TRNGs and PRNGs

Both true random number producer and pseudorandom number producer have their merits and demerits. Generally the constraint of one type is the merit of the other. Because of this only the advantages and disadvantages of TRNGs are listed in Table-1. The table applies to TRNGs that are deemed to be completely random. This analysis possibly sheds light on the correctness of particular RNGs to particular applications.

Table 1
Comparision of RNGs

Merits	*Demerits*
No periodicities	sluggish and incompetent
No predictability of random numbers based on acquaintance of prior sequences	unwieldy to mount and run
Assurance that there are no reliancepresent	Random number sequences are notreproducible
High level of protection	expensive

III. TYPES OF PRNGS

Pseudorandom number generators (PRNGs) use an algorithm to produce progression of random numbers. Common classes of algorithms are linear congruential generators, lagged Fibonacci generators, linear feedback shift registers and generalized feedback shift registers.

3.1. Linear Congruential Generators

Linear congruential generators (LCGs) represent one of the primitive and finest acknowledged pseudorandom number generator algorithms. The assumption behind them is simple to understand, and they are easily implemented. It is, however, well known that the properties of this class of generator are far from ideal. LCGs are defined by the recurrence relation:

$V_{n+1} = (AV_n + B) \bmod M$, where V is the sequence of random values and A, B and M are generator specific constants.

3.2. Linear Feedback Shift Register Generators

A linear feedback shift register is a shift register whose input is the XOR of some of its outputs. The outputs that influence the input are called taps [4]. A maximal LFSR produces an n sequence, unless it contains all zeros. If the taps are at positions 17 and 15 (as shown in Fig. 1), the polynomial is $A^{17} + A^{15} + 1$. When this polynomial is primitive, then the LFSR is maximal.

LFSR can be implemented in hardware, and this makes it useful in applications that require very fast generation of a pseudorandom sequence.

LFSR has been applied as a pseudorandom number generator for employing in stream ciphers (especially in military cryptography), due to the ease of building from simple electromechanical or electronic circuits, long periods, and very uniformly distributed outputs.

There are two types of LFSR: Fibonacci (Internal type) and Galois (External type)

Fibonacci LFSR- In this type of LFSR, XOR gate is applied internally between two D flip-flops. This type of LFSR works as comparator.

Galois LFSR- In this type of LFSR, XOR gate is applied externally between D flip-flops. This type of LFSR is used for generating pseudo random pattern.

IV. TESTING OF RANDOMNESS

The sequences generated by different generators may be put for testing of their randomness.

4.1. Statistical Testing

Each test tries to detect a different kind of non-randomness. The tests are not totally exclusive - there will generally be some overlapping. If the sequence is deemed to have failed any one of the statistical tests, then the generator may be abandon as being non-random; alternatively and advisably, the generator may be subjected to further testing. Contrary to this, if the sequence passes all of the statistical tests, the generator is concluded to being random from a statistical point of view [8][9]. This conclusion is, of course, not definite, but rather probabilistic.

Although statistical testing, is the most widely used method of testing the randomness of a random number generator [6], there are other ways to detect non-randomness in a sequence also but few points may be taken care before selecting a testing method.

There are numerous possible statistical tests, each assessing the presence or absence of one of the many departing from that, would of a completely random sequence and if detected, would indicate that the sequence is non-random [7]. Because there are so many tests for judging whether a sequence is random or not, no specific finite set of tests is deemed to be complete.

There is no perfect test for randomness rather it depends upon the type of RNG and its application. Selecting a good quality of RNG for all applications may not be trivial.

V. DESCRIPTION OF TESTS

5.1. Frequency (Monobit) Test

The objective of this test is the emergence of high and low values for the entire progression. The reason of the test is to decide whether the quantity of high and lows occurring in a progression are around the same as it is predictable for a truly random progression. The tests evaluate the proximity of the fraction of high to be fifty percent, i.e., the number of high and lows in a progression must be about matching. This is a primary test and failing this increases the possibility of further tests deteriorating.

For a bit string of length n, high and low of the input progression are altered into the sequence of values -1 and +1 and then added collectively to produce $A_1 + A_2 + ... + A_n$, where $A_i = 2å_i$ "1. The test statistics and P-value of the sequence are obtained using equations (1) and (2).

$$S_{obs} = \frac{|Sn|}{\sqrt{n}} \tag{1}$$

$$\text{P-value} = \text{erfc}\left(\frac{Sobs}{\sqrt{2}}\right) \tag{2}$$

Here the "erfc" is the complementary error function. If the calculated P-value is <0.01, then it may be concluded that the progression is non-random. NIST suggests that each progression to be tested should consist of at least 100 numbers of bits. Big positive values of S_n are pinpointing of too many ones, and big negative values of S_n are pinpointing of too many zeros in the bit stream.

5.2. Frequency Test within a Block

The test spotlight on the percentage of ones within M-bit blocks. The intention of this test is to establish whether the occurrence of ones in an M-bit block is in the order of M/2, as would be predictable under the hypothesis of randomness. Except for block size M=1, the test be reduced to the Frequency (Monobit) test. Some more parameters can be derived using equations (3) and (4).

$$\pi_i = \frac{\Sigma_{j=1}^{M} \varepsilon(i-1)M + j}{M}, \quad for\ 1 \le i \le N \tag{3}$$

$$\chi^2(obs) = 4M\ \Sigma_{i=1}^{N} (\pi_i - 1/2)^2 \tag{4}$$

Here M is the size of each block, n is the bit sequence size, and å is the sequence of bits as produced by the RNG or PRNG being tested.

Now compute the P-value = chidist (χ^2 (obs), df), where chidist returns the one-tailed probability of the chi-squared distribution and df is the degrees of freedom.

If the calculated P-value < 0.01, subsequently it is concluded that the progression is non-random. Otherwise, the progression is random. When P-values are small (<0.01) then it shall indicate a large variation from the equal percentage of high and lows in at least one of the blocks. NIST recommends that the size of each progression to be tested should be a minimum of 100 bits (i.e., n ≥ 100) and that the block size M should be chosen such that M ≥ 20, M> 0.1n and N<100. The lower bounds of M =20 has been chosen with N =50 and n=1000. Table-3 shows the proportion of Ones (π_i) for different blocks.

Table 3
Proportion of Ones (π_i) for different blocks

Block No.	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
Proportion of Ones (π_i)	0.4	0.7	0.4	0.3	0.5	0.3	0.4	0.4	0.4	0.4

VI. RUNS TEST

A run is defined as a series of ascending and descending values. The number of ascending or descending, values is known as length of the run. In a random data set, the possibility that the (n+1)th value is bigger or lesser than the nth value follows a binomial distribution. This distribution is used when the events are mutually exclusive. This distribution forms the starting point of the runs test. In the input sequence, the pre-test proportion π of ones, the pre-test Frequency test, the test statistic, and P-value can be obtained by using equations (5), (6) and (7) respectively.

$$\pi = \frac{\Sigma_j \varepsilon j}{n} \tag{5}$$

$$V_n(\text{obs}) = \Sigma_{k=1}^{n-1} r(k) + 1 \tag{6}$$

Here r(k) = 0 if $\varepsilon_k = \varepsilon_{k+1}$, and r(k) = 1 otherwise.

$$\text{P-value} = \text{erfc}\left(\frac{Vn(obs) - 2n\pi(1-\pi)}{2\sqrt{2n}\,\pi(1-\pi)}\right) \tag{7}$$

If the computed P-value <0.01, then it is concluded that the sequence is non-random. Otherwise, the sequence is random.

A huge cost for V_n (obs) indicates an oscillation in the string which is too fast; a small value would indicate that the alternation is too sluggish. A fast alternation occurs when there are a many changes. A sequence with a slow oscillation has fewer runs that would be expected in a random sequence. NIST recommends that each run to be tested must consist of at least 100 bits (i.e., n ≥ 100).

VII. EXPERIMENT AND ANALYSIS

LFSR of degree 8 is implemented using the primitive polynomial $x^8 + x^6 + x^5 + x + 1$ and generated bit patterns are subjected to three different tests for randomness: a) Frequency Monobit, b) Block test and c) Run test. The P-value of LFSR is calculated for different sizes of bit patterns generated. Fig. 4-6 show the variations of P-value with frequency monobit test, frequency test within a block and runs test respectively.

On the similar lines, the bit patterns are generated with LFSR of degree 9 and degree 10 using the primitive polynomial defined by $x^9 + x^4 + 1$ and $x^{10} + x^3 + 1$ respectively. The output random bit patterns generated from both the LFSRs are subjected to test for randomness by varying the bit pattern size and P-value is plotted. Fig. 7-9 are related with LFSR of degree 9 and Fig. 10-12 are related with LFSR of degree 10.

The output is summarized in table No 4. for LFSR 8. The columns are for different bit size and rows of different test of randomness performed on them. A tick mark (ü) denotes the suitability of a particular size of LFSR generating that particular pattern of bit size whereas a cross mark (û) denotes unsuitability and failure to that particular test of specific bit pattern length.

Table 4.1
Analysis of testing for LFSR 8 for randomness (✓) and non-randomness (✖)

Bit Pattern Size / *Tests*	*20*	*40*	*60*	*80*	*100*
Frequency Monobit	✓	✖	✖	✖	✖
Block	✓	✖	✖	✖	✖
Runs	✓	✓	✓	✓	✓

The Table 4.2 and Table 4.3 summarize the output obtained for LFSR 9 and LFSR 10 respectively for the same test and bit pattern of different lengths.

Table 4.2
Analysis of testing for LFSR 9 for randomness (✓) and non randomness (✖)

Bit Pattern SizeTests	*20*	*40*	*60*	*80*	*100*
Frequency Monobit	✓	✓	✓	✓	✓
Block	✓	✓	✓	✓	✓
Runs	✖	✓	✓	✓	✓

Table 4.3
Analysis of testing for LFSR 10 for randomness (✓) and non randomness (✖)

Bit Pattern Size Tests	*20*	*40*	*60*	*80*	*100*
Frequency Monobit	✓	✓	✓	✓	✓
Block	✓	✓	✓	✓	✓
Runs	✓	✓	✓	✓	✓

VIII. RESULT AND DISCUSSION

There are three different most popular LFSR that have been design and the bit patterns of different lengths generated are tested using three tests for randomness. The criterion for testing randomness is the P- value, has been calculated for all the three LFSRs of size 8, 9, 10 bits respectively. The respective plots between P-value and bit pattern size length has been drawn in fig (a-i) that for same pattern different tests are giving different P-values. According to decision rule, if the P-value is greater or equal to 0.01 then the sequence is considered as random. Since the P-values are different for each pattern and each test so one can assume a sequence is random when it passes all the three tests.

The LFSR 8 passes randomness testing for frequency monobit test when the bit pattern size is 20, block test for all pattern sizes and runs test when the bit pattern size is 20. LFSR 9 passes randomness testing for frequency monobit test and block test for all bit pattern sizes, and runs test when the bit pattern size is 40, 60, 80 and 100. Whereas LFSR 10 passes randomness testing for the entire three tests, frequency monobit test, block and runs test for all bit pattern size 20, 40, 60, 80, and 100. Since the sequence generated by LFSR 10 passes all of the statistical tests, the generator is concluded to being random from a statistical point of view. Therefore the minimum size LFSR would be best out of four different kinds i.e., twenty bit.

IX. CONCLUSION

In this paper the experiment for randomness is performed, because in many electronics circuits design there is requirement of random bit patterns. These random bit patterns are generator by many ways. In this paper the analysis is performed on different LFSR that are also used for the generation of random number especially using hardware implementation. As the circuits' needs to be fabricated on a chip of minimum size and many embedded systems are area constrained and also to achieve proper randomness in the bit patterns generated by LFSR along with the constrained defined, one can decide the minimum size LFSR needed for its application that also passes the test of randomness. The experiment performed in this paper are on software further research is to implement theses using any HDL , generate the bit pattern and test these generated bit patterns using same tests.

REFERENCES

[1] P. R. Fernando, "Genetic Algorithm Based Design and Optimization of VLSI ASICs and Reconfigurable Hardware," Graduate Thesis, University of South Florida, 2009.

[2] E. J. Dudewicz, T. G. Ralley, "The Handbook of Random Number Generation and Testing with Testrand Computer Code," American Sciences Press, 1981.

[3] S. K. Park, and K. W. Miller, "Random Number Generators: Good Ones Are Hard To Find," Communications of the ACM, vol. 31, no. 10, pp. 1192-1201, 1988.

[4] http://www.fourmilab.ch/hotbits/

[5] M. Matsumoto, and T. Nishimura, "Dynamic Creation of Pseudorandom Number Generators," Proc. of the Third International Conference on Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing, pp. 56-69, 1998.

[6] I. Vattulainen, K. Kankaala, J. Saarinen, and T. Ala-Nissila, "A Comparative Study of Some Pseudorandom Number Generators," Computer Physics Communications, Elsevier, vol. 86, no. 3, pp. 209-226, 1995.

[7] P. Schaumont, "A Random Number Generator in Verilog: A Design Lecture," ECE 4514 Digital Design II, 2008.

[8] P. L'Ecuyer, Peter Hellekalek, "Random Number Generators: Selection Criteria and Testing," Lecture Notes in Statistics, vol. 138, pp. 223-265, Springer-Verlag New York Inc., 1998.

[9] A. Rukhin, J. Soto, J. Nechvatal et. al., "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," NIST (National Institute of Standards and Technology) Special Publication 800-22, 2001.