Research Journal of Information Technology

ISSN 1815-7432

Academic Journals Inc.

www.academicjournals.com

OPEN ACCESS Research Journal of Information Technology

ISSN 1815-7432
DOI: 10.3923/rjit.2017.32.37

Research Article
Efficient Implementation of Pseudo Random Numbers

[1]Edla Kumari, [1]Bharath Kompelli, [1]Reshma Kalicheti, [1]Naga Saride, [1]Khaled Elleithy and [2]Laiali Almazaydeh

[1]Department of Computer Science and Engineering, University of Bridgeport, Bridgeport, CT, USA
[2]Faculty of Information Technology, Al-Hussein Bin Talal University, Ma'an, Jordan

Abstract

Background: Pseudo random number generation is an algorithm for generating a stream of numbers as having the appearance of randomness. Random numbers are essential for many applications, including simulations, cryptography and random sampling. In this study, a model of Linear Feedback Shift Register is implemented in Verilog language using Xilinx software. The simulation results demonstrate that it is possible to generate a perfect random sequence. **Materials and Methods:** Practically, Verilog language is used in order to implement the LFSR and generate a random sequence. Verilog has a random number generator within it but it is permitted to only test benches. In Verilog, some modules will be written such as for flip-flops and multiplexer. In this study, a module naming LFSR is created using different parameters including clock, reset, load, input/seed, etc. **Results:** Simulation results show the outputs of the LFSR in the software. Xilinx ISE design suite system for the simulation of the Verilog code of the LFSR is used. The sequences are generated from the operations performed by Mux and flip-flop and the feedback too. **Conclusion:** The LFSR model is implemented in Verilog language using Xilinx software considering suitable time factor, inputs and clock signals. A perfect random sequences and synthesis of implemented model are generated.

Key words: Random numbers, pseudo random numbers, entropy, diffusion, bit manipulation

Received: August 26, 2016 Accepted: November 11, 2016 Published: December 15, 2016

Citation: Edla Kumari, Bharath Kompelli, Reshma Kalicheti, Naga Saride, Khaled Elleithy and Laiali Almazaydeh, 2017. Efficient implementation of pseudo random numbers. Res. J. Inform. Technol., 9: 32-37.

Corresponding Author: Laiali Almazaydeh, Faculty of Information Technology, Al-Hussein Bin Talal University, Ma'an, Jordan

Competing Interest: The authors have declared that no competing interest exists.

Data Availability: All relevant data are within the paper and its supporting information files.

INTRODUCTION

Pseudo Random Number Generation (PRNG) is one of the crucial problems of computer science. Its history from computer infancy onward is documented, for instance, in Donald Knuth's classic book[1]. The PRNG is an algorithm for generating a stream of numbers that approximate randomness. Many applications rely on random numbers, including statistical modeling, electronic games design for behavior of a computer-controlled character and cryptography for keys of data encryption[2]. However, true random values have two drawbacks. The entropy available is limited and many applied problems would be awkward if limited by the accessibility of physical entropy.

A PRNG is needed in order to have a major benefit in numerous applications. In a parallel program, every thread must have its own pseudo random sequence and each of this sequence must be statistically independent of all the others. If new threads are created dynamically, new PRNGs must be seeded with the generating string's PRNG as the main wellspring of entropy. Moreover, PRNGs are efficient, they can generate many numbers in a short span of time and deterministic, meaning that a given sequence of numbers can be emulated at a later stage if the beginning stage in the sequence is known. The PRNGs are normally likewise occasional, which means that the sequence will eventually repeat itself. While periodicity is barely ever an attractive trademark, modern PRNGs have a period that is so long that it can be overlooked for most viable purposes. These qualities make PRNGs suitable for applications where sequence of numbers is required and where it is helpful that the same sequence can be replayed easily. Popular examples of such applications are modeling and simulation applications. The PRNGs are most certainly not suitable for applications where it is important that the numbers are truly unusual, for example, data encryption and gambling.

A PRNG is an algorithm that produces a series of numbers whose properties inexact the properties of groupings of random numbers. A PRNG is also called as Deterministic Random Bit Generator (DRBG). But the output series is not truly random, as it is totally determined by a nearly small set of values, called the PRNG's seed (that include random values). While the series that are closer to truly random can be generated using hardware random number generators, PRNG's are essential in form for their speed in number generation[3,4].

A PRNG combines bitwise logical operation and bitwise simulation in order to generate the sequence of random numbers. It starts with a randomly selected input key which are known as seeds which consists of any combination of letters (lower or upper) and numbers (0-9). Normally the seeds can be exchanged between the sender and the receiver publicly but it would be better if it would be done secretly. For generation of 64 bits key, the generator needs 8 characters likewise for 128 bits key the generator needs 16 characters. These characters will be converted to ASCII codes and the operations are performed for generation of sequence of random numbers. Some sequences are also used for the generation of the next sequence. Mixing of bitwise operation and simulation serves to avoid purely algebraic and bit oriented attacks. This generator is used in many applications as it generates good number of sequences[5-8].

This study demonstrated the generation model while considering many factors which should affect the quality of generated pseudo random numbers including long period, uncorrelated sequences, uniformity and efficiency.

MATERIALS AND METHODS

Related works and schemas

Linear Feedback Shift Register (LFSR) in encryption: The use of LFSR in cryptography has been increasing extensively. Many algorithms used nowadays rely on LFSR generator. For example, the cryptographic algorithms in the GSM mobile-phone system use the concept of LFSRs. An LFSR comprises of registers which contain sequence of bits and a feedback function. The main operation which is performed in the LFSR is exclusive-OR on certain bits in the register. Performing XOR operation on the bits in generating the random sequences. This list of bits may be called as "Tap sequence". The generated sequence is also used for the generation of the next sequence where feedback function helps us in doing that. While generating the sequence the registers will be loaded with non-zero content and then the implementation computes the XOR operation and generates the sequences. At present, LFSRs are well suited for the hardware implementations[9].

LFSR as a cipher device: Differentiating the data to its ASCII value, each character at a time, using a 2^8×8 priority encoder with 1 byte per character, the 8-bit word is stored in an 8-bit right shift register M which has a parallel input. Then a shift control input is brought in with the clock having an octal word-time signal so that number pulse is same as the number of bits in the shift register. When the input bits are moved towards right, a 0-bit enters from the left most register so that by the time of 8th clock pulse finishes the contents of 8-bit register is reset back to 0. The output mode is in serial out mode.

Feedback function: The importance of the feedback function in the LFSR is crucial as it helps in generating the random sequences[10]. The operations performed here are very simple and it generates random sequences by performing few operations on the 8-bits in the shift register and 3-bit register (serially in and serially out). After providing the first clock pulse, the extreme right bit of the register will undergo some transformations caused by LFSR set-up. Let us consider a 3-bit register initially set to 000 from left to right be named as A, B and C. Consider both registers are working under the same clock pulse or timing sequence. Now the initial output of C is XORed with the first bit of the 8-bit shift register. Here the feedback used supplies input to A which is f = ((AB XNOR C) XOR A).(ABC)'. These operations are performed for 8 clock pulses and the random sequences are generated.

Solution: The LFSRs are very simple to construct and are very useful in variety of applications but they are overlooked by many designers. The LSFR is a simple shift register with a general feedback. Let us consider an example which gives us clear understanding how the problems are solved. In following circuit, which shown in Fig. 1, the bits tapped are bit 0 and 2 where these share a common clock input. Here input is generated by XOR the tap bits where remaining bits are standard shift register. The sequences generated are based on the feedback. Consider two 3-bit XOR based LFSRs with different tap selections.

Both start with initial value but their sequences rapidly diverge as clock pulses due to different taps. In some rare cases LFSR end up cycling round a loop generating a limited number of values. Here the binary field with n bits can assume 2^n unique values but whereas maximal length LFSR with n-register bits will generate through (2^n-1) values. This is due the fact that the LFSR having XOR feedback will not sequence through the value where all bits are 0 while XNOR equivalents will not generate sequence through the value where all bits are 1. This is the way how the random sequences are generated. Figure 2 shows the table of the sequences generated for the above example[11].

Mathematical model: Based on the exclusive-OR circuits and shifts of the registers, the LFSR's are classified. The shifts are normally based on two sides from left to right and right to left. Similarly the basis of exclusive-OR operation is classified, the

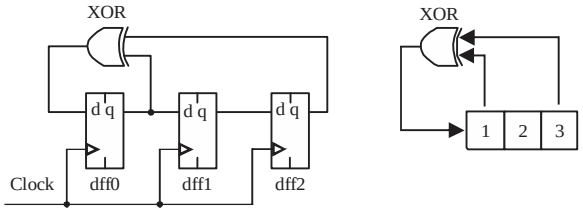

Fig. 1(a-b): LFSR (a) Circuit and (b) Symbol

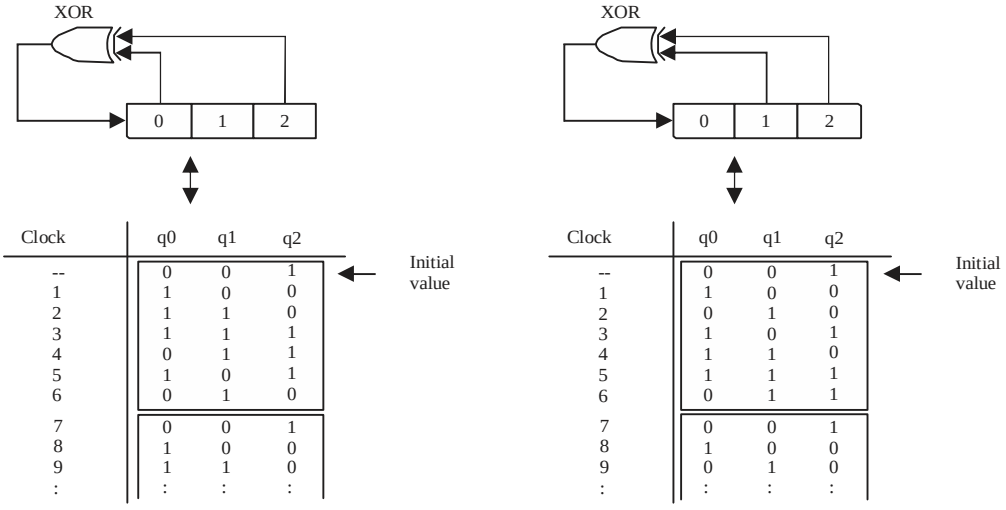

Fig. 2: Feedback from 3-bit shift register

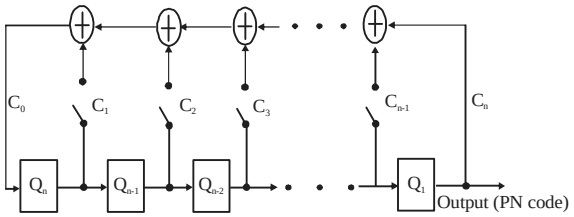

Fig. 3: An n-bit LFSR

$$\begin{bmatrix} Q_1(t+1) \\ Q_{n-1}(t+1) \\ Q_{n-2}(t+1) \\ \vdots \\ Q_2(t+1) \\ Q_1(t+1) \end{bmatrix} = \begin{bmatrix} C_n & C_{n-1} & \cdots & C_2 & C_1 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} Q_n(t) \\ Q_{n-1}(t) \\ Q_{n-2}(t) \\ \vdots \\ Q_2(t) \\ Q_1(t) \end{bmatrix} \quad (1)$$

Fig. 4: A feedback taps

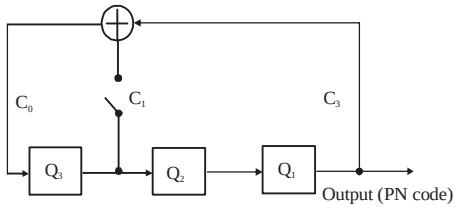

Fig. 5: A 3-bit LFSR

LFSR's are classified into external exclusive-OR (EEOR) and internal exclusive-OR (IEOR). Here the mathematical model that is based on external exclusive-OR circuits and shift register shifts the bits from right to left is considered. Figure 3 shows an n-bit LFSR.

In the above considered model, the feedback taps vector $[C_0,\cdots, C_n]$ and these are linked with flip-flop's outputs $Q_n, Q_{n-1},\cdots, Q_1$ respectively. The state space model of the above structure can be represented by the following Eq. 1, as shown in Fig. 4.

In the equation of Fig. 4, LFSR feedback stages are numbered from $C_0,\cdots, C_n$ and those are proceeded in the way of shifting, i.e., from left to right. Whereas the present state is represented by Q(t) and the next state by Q(t+1) and the relation between them was represented above[5]. This LFSR is governed by the following equation [Q(t)], [A][Q(t)], [A]2[Q(t)], [A]3[Q(t)] so on for any non-zero loadings. Considering any small integer 'p' which is the period of matrix, then $[A]^p[Q(t)] = [Q(t)]$ for any non-zero initial vector [Q (0)]. Based on the property of periodicity of LFSR, $[Q(t)] = [Q(t+p)] = [A]^p [Q(t)]$. For demonstrating this, a 3-bit LFSR is considered as an example, as shown in Fig. 5.

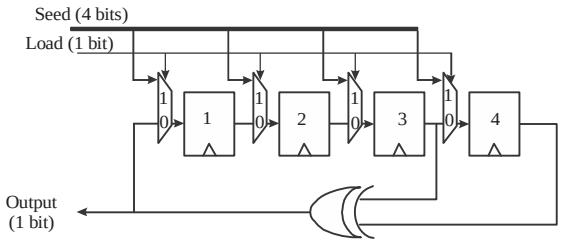

Fig. 6: Circuit diagram for LFSR generator module

Table 1: Operation of LFSR

States			
Q_3	Q_2	Q_1	Comments
1	1	1	Initial
0	1	1	
1	0	1	
0	1	0	
0	0	1	
1	0	0	
1	1	0	
1	1	1	Repeats after a period p = 7

Here, only three feedbacks and neglected one back and the states of LFSR are shown in the Table 1 where generation of sequences will be repeating after the period of seven.

RESULTS

Verilog language has been considered to implement the LFSR and generate a random sequence. Verilog has a random number generator that is limited only to test benches. In Verilog some different modules are written such as for flip- flops and multiplexer. We need to declare some values for setup time, hold time, delays, etc. In this implementation, a module LFSR is created using of different parameters including clock, reset, load, input/seed, etc.

Figure 6, represents the operations performed in the process of generation of random sequences. The seed, which is known as input, is considered with only 4 bits and the load is given to the multiplexers. The output bit comes from the mux and that bit is given as feedback in the process of generation of next sequence.

Figure 7a and b, show the outputs of the LFSR. Xilinx ISE design suite system for the simulation of the Verilog code for the LFSR is used. Figure 7a is the output window which is showing the inputs and outputs of the LFSR. The sequences are generated from the operations performed by Mux and flip-flop and the feedback too. Figure 7b shows the synthesis of the program. As the time factor is very important, for every 50 nsec the state of the clock in our module is reversed.

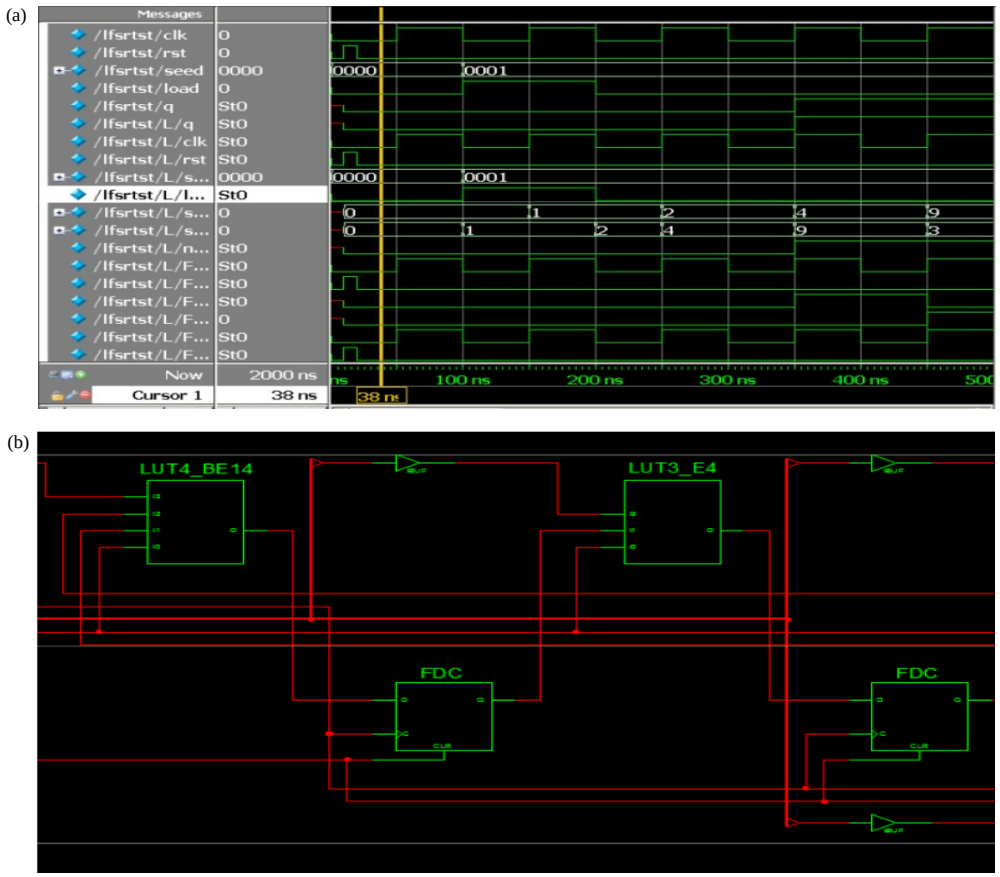

Fig. 7(a-b): (a) Simulation results output and (b) Synthesis output

DISCUSSION

Many factors affect the quality of the generated pseudo random numbers. Although LFSR are very simple to construct and are very useful in variety of applications but it was overlooked by many designers. Therefore, this study demonstrated LFSR are very simple to construct and are very useful in variety of applications. However, other study of Hazwani *et al.*[12] proposed PRNG circuit consisting of 24-bit LFSR which has much longer period, in addition, the study by Bonde and Kale[13] required multiple bits, where LFSR will be extended by utilizing extra time and extra circuitry.

CONCLUSION

The LFSR model is implemented in Verilog language using Xilinx software with consideration of suitable time factor, inputs and clock signals. A perfect random sequences and synthesis of the implemented model are generated.

SIGNIFICANT STATEMENT

Although generation of perfect pseudo number sequences is of utmost importance in many applications, hardware designers have overlooked its importance and consequently its design. This study demonstrated an efficient LFSR implementation to generate a perfect random sequence. The experimental results show that the LFSR model implementation considering many factors which should affect the quality of the generated pseudo random numbers.

REFERENCES

1. Knuth, D.E., 1997. The Art of Computer Programming, Volume 1: Fundamental Algorithms. 3rd Edn., Addison-Wesley, Massachusetts, ISBN-13: 978-0201896831, Pages: 672.
2. Hastad, J., R. Impagliazzo, L.A. Levin and M. Luby, 1999. A pseudorandom generator from any one-way function. SIAM J. Comput., 28: 1364-1396.

3. Stipcevic, M. and C.K. Koc, 2014. True Random Number Generators. In: Open Problems in Mathematics and Computational Science, Koc, C.K. (Ed.)., Springer International Publishing, Switzerland, ISBN: 978-3-319-10682-3, pp: 275-315.
4. Fischer, M.J., 2006. Pseudorandom sequence generation. Yale University. http://zoo.cs.yale.edu/classes/cs467/2006f/course/handouts/ho15.pdf
5. Maaita, A.A. and H.A.A. Al Sewadi, 2015. Deterministic random number generator algorithm for cryptosystem keys. Int. J. Comput. Electr. Automation Control Inform. Eng., 9: 972-977.
6. Mishra, M. and V.H. Mankar, 2015. Text encryption algorithms based on pseudo random number generator. Int. J. Comput. Applic., 111: 1-6.
7. Bagdasar, O.D. and M. Chen, 2014. A horadam-based pseudo-random number generator. Proceedings of the UKSim-AMSS 16th International Conference on Computer Modelling and Simulation, March 26-28, 2014, Cambridge, pp: 227-231.
8. Anikin, I.V. and K. Alnajjar, 2016. Pseudo-random number generator based on fuzzy logic. Proceedings of the International Siberian Conference on Control and Communications (SIBCON), May 12-14, 2016, Moscow, pp: 1-4.
9. Jamil, T. and A. Ahmad, 2002. An investigation into the application of linear feedback shift registers for steganography. Proceedings of the IEEE SoutheastCon, 2002. April 5-7, 2002, Columbia, SC, USA., pp: 239-244.
10. Golomb, S.W., 1982. Shift Register Sequences. Aegean Park Press, Leguna Hills, USA., Pages: 324.
11. Maxfield, C., 2008. Bebop to the Boolean Boogie: An Unconventional Guide to Electronics. 3rd Edn., Newnes, UK., ISBN-13: 978-1856175074, Pages: 568.
12. Hazwani, S., S. Khan, M.U. Siddiqi, K.A.S. Al-Khateeb, M.H. Habaebi and Z. Shahid, 2014. Randomness analysis of pseudo random noise generator using 24-bits LFSR. Proceedings of the 5th International Conference on Intelligent Systems, Modelling and Simulation, January 27-29, 2014, Langkawi, Malaysia, pp: 772-774.
13. Bonde, V.V. and A.D. Kale, 2015. A review on implementation of random number generation based on FPGA. Int. J. Sci. Res., 4: 2749-2753.