Available online at www.sciencedirect.com
ScienceDirect
Procedia Technology 24 (2016) 1155 – 1162
Procedia Technology

International Conference on Emerging Trends in Engineering, Science and Technology (ICETEST 2015)

FPGA Implementation of High Quality Random Number Generator using LUT based Shift Registers

Remya Justin[a*], Binu K Mathew[b], Susan Abe[c]

[a, b, c] *Dept. of Electronics and Communication, SAINTGITS College of Engineering, Kottayam, India, 686532*

Abstract

Random numbers are required for wide range of applications such as in encryption of data, testing and Monte-Carlo simulations. So, hardware implementation of random number generator is inevitable. FPGA Optimized RNGs are more efficient in terms of resource than software based RNGs. The LUT-SR RNG, a type of FPGA RNG in which LUTs are configured into shift registers with varying length. The existing work provides a midpoint between LUT-OPT RNG and LUT-FIFO RNG. In the enhancement work, we proposed modified LUT-SR RNG which provides more randomness, quality and minimum resource utilization than the existing LUT-SR generator. Inorder to improve the randomness quadratic residue method is employed. Linear Congruential Generator (LCG) algorithm, one of the oldest and well known algorithm is also used in modified LUT-SR RNG to enhance the performance. Here design was made by VHDL programming language by using Xilinx software.

Peer-review under responsibility of the organizing committee of ICETEST – 2015

Keywords: **Field Programmable Gate Array (FPGA); Look Up Table (LUT); Random Number Generator (RNG)**

1. Introduction

Random number generators play a vital role in the field of cryptography, Monte- Carlo calculations [1], testing, banking etc. In order to function properly, these applications require many parallel streams of high quality, large period, uncorrelated uniform random number generators as well as large processing power. FPGA optimized RNGs are resource efficient than other software RNGs because they utilizes the advantages of bit wise operation and parallelism of FPGA. Generally, cost of uniform random number generation is lower in FPGA using Look Up Tables (LUT) [1] or First In First Out (FIFO) [3] queues. The various needs of the application determine the use of

* Corresponding author. Tel.: +91-9400831566;

E-mail address: remyatess@gmail.com

Peer-review under responsibility of the organizing committee of ICETEST – 2015
doi:10.1016/j.protcy.2016.05.069

the above mentioned generators. FPGA optimized RNGs are not widely popular because the construction of generator with given parameters is time consuming [10]. Faced with this problem, designer under time constraint choose less efficient generators like Linear Feedback shift Register (LFSR) and Tausworthe Generators.

FPGA optimized generators provides an easier and simple method to instantiate a RNG that meets the unique needs of the engineer. LUT - SR RNG is a family of generators which uses LUT as shift registers to achieve high quality and long period with minimum resource utilization [3]. The main aim of this project is to construct a highly efficient random number generator which overcomes the demerits of existing systems. In this work, modified LUT-SR RNG is proposed with minimum resource utilization, high quality and reduced delay than the existing LUT-SR RNG. In the modified LUT-SR RNG, the architecture is enhanced using Linear Congruential Generator (LCG) which is the best known pseudo random number generator algorithm. The principle of quadratic residue is also applied in the modified system to enhance the randomness and quality.

2. Previous Works

2.1. Binary Linear RNGs

Bit-level generators are of much more interest for FPGAs and they use binary linear recurrences, in which implementation of multiplication and addition of bits is done using bitwise – and ($\otimes$) and exclusive –or ($\oplus$) [2]. A linear generator contains an n bit state and produces r bit outputs

$$x_{i+1} = Ax_i \quad (1)$$

$$y_{i+1} = Bx_{i+1} \quad (2)$$

where $x_{i=} (x_{i,1},....., x_{i,n})^T$ represents the generator with n-bit states, $y_{i=} (s_{i,1},....., s_{i,r})^T$ is the r -bit output of the generator, A represents a n × n binary transition matrix, and B represents a r × n binary output matrix [2]. The sequence gets repeated due to the finite period. The RNG designers always aim to achieve a maximum period of 2^n-1. A period of $2^n$is not achievable because it is unable to select a matrix 'A' with $x_0 = 0$ maps to any state other than $x_1 = 0$. This implies that there are two random number series in a maximum period generator: a sequence contains only zero of length 1 and the main sequence that go through every possible non-zero n bit pattern before repeating [3]. The characteristic polynomial P (z) of the matrix 'A' (transition matrix) must be primitive is the essential criteria for a generator to own maximum period.

2.2. LUT-Optimized (LUT-OPT) RNGs

In this family of generators, each row and column of matrix 'A' contains t-1or t 1s, where t is the no. of taps. Each row maps to t or t-1 input XOR gate [1]. Each new state bit can be generated using one LUT and r-bit generator can be generated using r- fully utilized LUT-FFs [3].A general structure of the 4- bit LUT-OPT generator is illustrated in Fig.1.

Merits of LUT-OPT RNG

1) Efficient resource Utilization: Every bit need same LUT and FF, as a result resource utilization increases linearly. This RNG requires r LUT-FFs for generating r bits per cycle.

2) Performance: These generators are extremely fast since the critical path is a single LUT-delay.

However, these generators have some demerits also.

1) Complexity: It is difficult to construct a unique matrix because for every (r, t) pair requires a specific matrix of connections, without using specialized software. It is impossible to encode these matrices, if these are constructed randomly.

2) Quality: The random bits are generated as the linear combination of random bits generated in the previous cycle. The new bits will be obtained simply by the XOR-ing of input bits from the preceding cycle. Thereby leads to this lag-1 linear dependence.

3) Period: To achieve a period of 2^n - 1, it is essential to take r = n, even if lower than n bits are required per cycle. The minimum period for LUT-OPT RNG is 2^{64}– 1. But much larger periods are preferred.

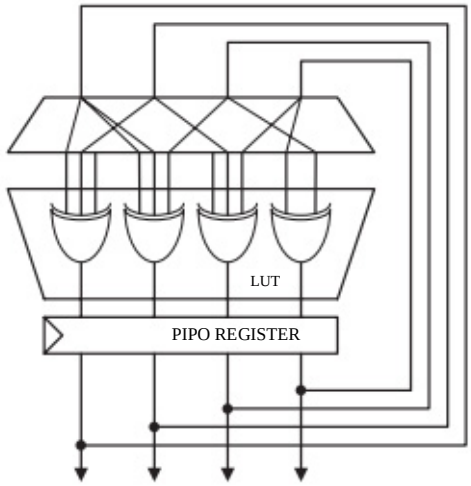

Fig.1. 4-bit LUT-OPT RNG

4) Seeding: If the initialization is different, then each instances of same algorithm gives different random sequences. For a randomly selected matrix A, the only way is to load the generator in parallel. The serial loading is not much practical.

2.3. LUT-First In First Out (LUT-FIFO) RNGs

One solution for avoiding the problems faced by LUT-OPT RNG is to provide LUT-FIFO generators [2]. Here, these generator includes an extra depth- k, width-w first-in-first-out (FIFO), giving an overall period of 2^n- 1, where n = r + w k [2], shown in Fig.2. LUT-FIFO generators are generators with longer period than LUT-OPT RNG such as 2^{11213}– 1 and 2^{19937}– 1.

But, these RNGs also have some demerits:

1) FIFO in the LUT-FIFO RNG is implemented using a block of RAM. RAM is an expensive resource which is preferably used else-where in a design. So, it is not much resource efficient as LUT-OPT RNG.

2) The choice of r varied only in multiples of depth k. The reduction in flexibility is due to the word-wise granularity of block-RAM-based FIFOs.

The quality and period related disadvantages of LUT-OPT RNG are eliminated by the LUT-FIFO generators. The LUT-FIFO generator provides longer period than LUT-OPT generator. But, it faces the problem of increase in complexity. The efficient initialization of this generator is slightly worse as LUT-OPT generator. Even if the generator is quite expensive due to the usage of block of RAM, the presence of FIFO provides the advantage of longer period to this generator. LUT-FIFO generators are the fastest and efficient generators with high quality and long period.

2.4. LUT-Shift Register (LUT-SR) RNGs

LUT-based shift registers are the cheapest generator when comparing with other FPGA based generators. The state bits equal to n = r (1 +k) can be obtained using r shift registers. one for every output state. The complexity and seeding related issues can be eliminated using LUT-SR registers. LUT-SR RNG [4] provides much higher period than LUT-OPT RNG at the expense of one extra LUT-FF per bit. It doesn't need the block RAM also. Every shift register [6] is allotted with some unique length $k_i \leq k$, lowering the state size to $n = \sum_{i=1}^{r} (1 + k_i)$ .By randomly assigning each shift register with $k_i = k$ or $k_i = k - 1$, a much better period $r*k < n < r(1 + k)$ is obtained. The better solution is to randomly select $1 < k_i \leq k$, such that $\exists i, j : i \neq j \wedge \gcd(k_i + 1, k_j + 1) = 1$. This provides better random mixing of bits within the state [5].

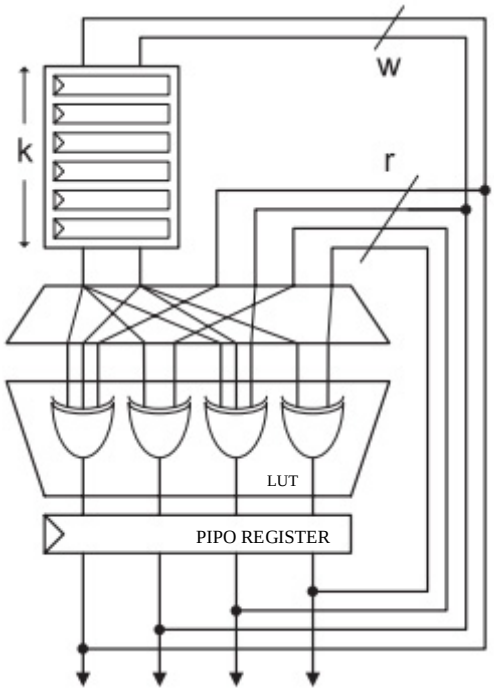

Fig.2. 4-bit LUT-FIFO RNG

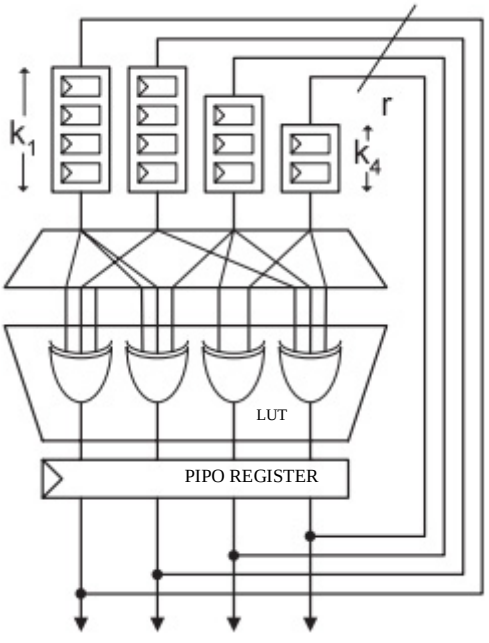

Fig.3. 4-bit LUT-SR RNG

Algorithm for LUT-SR RNGs

The algorithm needs a 5-tuple input (n, r, t, k and s).

where, ‘n’ denotes Number of bits in the generator.

'r' denotes Number of random output bits generated per cycle.
't' denotes XOR gate input count.
'k' denotes Maximum shift register length.
's' denotes Free parameter used to select a specific generator.

The algorithmic steps are as follows:

1) Create Initial Seed Cycle: A cycle of length r is generated from the 'r' numbers of X-OR logic gates at the output stage of the generator. Here, there are 'r' numbers of FIFOs with no length.
2) FIFO Extension: Random selection of a FIFO and incrementing its length by 1. The cycle is randomly increased, until required cycle length is achieved.
3) Add Loading Connections: The predetermined cycle is given to "t's", that specifies the matrix A. The cycle explains the whole FIFO connections, and also specifies the initial input to each of the 'r' numbers of XOR gates.
4) Add XOR Connections: Initial input for each of the XOR gates in the generator is given by the cycle. The additional t – 1 random inputs for the XOR gates are given over t − 1 rounds. Each round is obtained from the first in first out outputs and some bits are assigned with the same FIFO bit in more than one round.
5) Output Permutation: The final output permutation is used to avoid the dependency between the adjacent bits.

3. Proposed System Description

The LUT-FIFO RNG provides high quality random number sequence, but uses the expensive resource that is, the RAM block. Even if the existing LUT-SR RNG generates random sequence with reduced resource utilization than LUT-FIFO RNG, it won't give random numbers with high quality as LUT-FIFO RNG provides. The modified LUT-SR RNG, the proposed system aims to provide high quality, minimum resource utilized RNG than the existing LUT-SR RNG. The modified LUT-SR RNG aims to generator a random number sequence with high quality as LUT-FIFO RNG and minimum resource utilization as LUT-OPT RNG. The Fig 4 shows the block diagram of the modified LUT-SR RNG system.

In the modified LUT-SR RNG, the first in first out shift register (FIFO SR), quadratic residue block, xor connection block and parallel in parallel out shift register (PIPO SR) are the blocks. They together generate the random number sequence. Each individual block has its own unique functionality.

In the existing LUT-SR RNG, simple permutation is provided in order to improve the randomness. The simple dependency between adjacent bits is masked up using the permutation block. This method only avoids overlapping of nearby bits. This did not much increase the randomness of the sequence generated. In existing method of permutation, the first and last bits are interchanged and remaining bits positions are remains same. In the modified LUT-SR generator architecture, the randomness is enhanced by replacing the permutation block with 'quadratic residue' block, a modular arithmetic. This also improves the unpredictability in a better way than the simple permutation method.

When '*p*' is a prime number, 'x^2 mod p' has some interesting properties. Numbers produced this way are called quadratic residues. The quadratic residue of '*x*' is unique as long as 2x<p.

In the proposed system, the oldest and well known linear congruential generator (LCG) algorithm is used. This LCG algorithm is used in order to enhance the overall performance and to minimize the resource utilization. LCG algorithm uses discontinuous piecewise linear equation for yielding sequence of pseudo-randomized numbers. The theory behind them is easy to understand, and they are easily implemented and fast. The method of this random number generation by linear congruential method, works by computing each successive random number from the previous.

Starting with a seed, Xo, the linear congruential method uses the following formula:

$$X_{i+1} = (A*X_i + C) \bmod M \quad (3)$$

where,

X is the sequence of pseudo random bits and
M, 0<M — the "modulus"
A, 0<A<M — the "multiplier"
C, 0<C<M — the "increment"

X_o, $0<X_o<m$ — the "seed" or "start value" are constant integer values which are used to specify the generator.

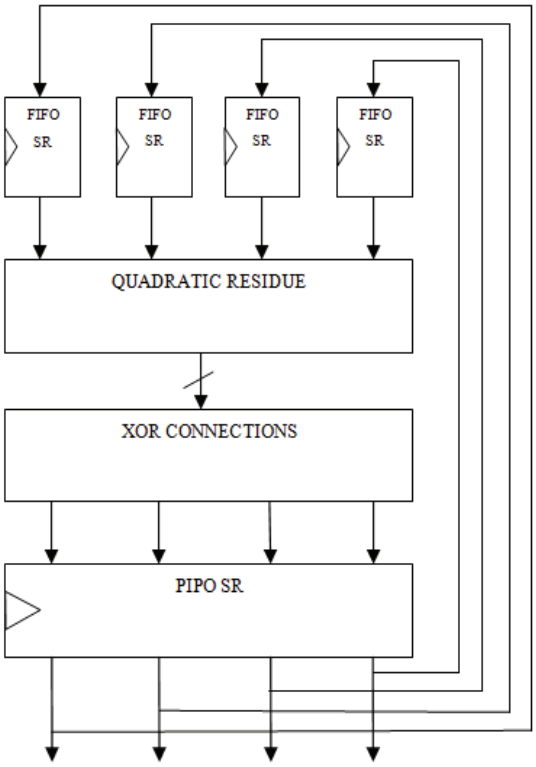

Fig.4.Block diagram of modified LUT-SR RNG

The algorithmic flow of modified LUT-SR RNG is shown in Fig. 5

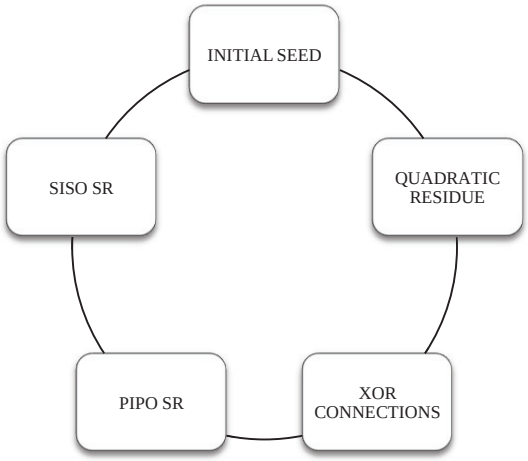

Fig.5. Algorithmic flow

4. Results

In the proposed system, the initial seed for the 16-bit modified LUT-SR RNG block is given through Parallel in Parallel Out Shift Register. An n-bit shift register shifts the input data by one position for every clock pulse. The resulting sequence in each PIPO SR is fed back to the SISO SR. Quadratic residue method is employed in order to enhance the randomness and permuted outputs are given to the X-OR gated logic. Then the output of the X-OR gates are passed to the PIPO SRs. In PIPO SRs, the X-OR gate results are shifted. Thus random number generation is completed. The simulations are performed in Model Sim 6.4a and synthesized using Xilinx Plan Ahead Virtex5 kit verified on the Spartan3E kit. The VHDL is used to write the program. The results design summary is obtained from Xilinx 14.5i.

The initial seed is given as input through PIPO SR and fed back to SISO SR. The quadratic residue block will randomize the sequence and given to XOR gates. For a 16-bit generator, the no. of XOR gates is 16. The resulting outputs generate the random number cycle. The cycle is fed into the SISO SR [FIFO] of varying lengths (length=k). The length should not exceed r. As each bit crosses the flip-flop, it will be set to zero. Thus random number generation takes place. The count of all zero state is avoided because all zero state leads to idle condition. The initial seed is the factor which determines the period after which the whole generated sequence goes on recurring and the permutation done through quadratic residue method.

Table 1. Overall Comparison using Spartan3E Family

Overall comparison(16-bit generator)	n	Resources used			Delay (ns)
		RAM	LUT	FF	
LRSR	16	0	1	16	2.258
CA	16	0	16	16	4.182
LUT-OPT	16	0	16	16	3.211
LUT-FIFO	336	1	47	52	4.040
LUT-SR	256	0	23	32	3.590
Modified LUT-SR	256	0	33	16	3.500

The overall comparison of the 16-bit LFSR, CA, LUT-OPT RNG, LUT-FIFO RNG, LUT-SR RNG and modified LUT-SR RNG are indicated in the table I. The number of states, n is an essential factor which determines the randomness of the generator. The number states bits are highest for LUT-FIFO RNG when comparing with all other generators. The LFSR have the minimum resource utilization and lowest delay in sequence generation. All 16-bits in a cycle can be used by both CA and FPGA Optimised RNGs but only 1 bit per state is usable for LFSR. Since period of generated sequence of CA depends on seeding, its maximum period is unpredictable.

The LUT-SR RNG and modified LUT-SR RNG have same number of state bits which less than LUT-FIFO RNG but much greater than LUT-OPT generator. The LUT-FIFO RNG provides a generator with high quality and high resource utilization. It uses expensive RAM block. The LUT-OPT RNG provides a generator with low quality and low resource utilization. The LUT-SR generator is at midpoint between LUT-OPT RNG and LUT-FIFO RNG. The modified LUT-SR RNG provided a generator with lesser resource utilization than the existing LUT-SR RNG without sacrificing the qualities of LUT-SR generator. There is also a decrease in the delay of LUT-SR generators when compared with LUT-FIFO RNGs.

Table 2. Quality Vs Resource utilization

16-bit Generator	n	r	*w*P(*z*)/*n*	LUT	FF
LUT-FIFO	336	16	.50	47	52
LUT-SR	256	16	.45	23	32
Modified LUT-SR	256	16	.48	33	16

The quality and resource utilization of 16-bit LUT-FIFO RNG, LUT-SR RNG and modified LUT-SR RNG are analyzed in the table II. The resource utilization is highest for LUT-FIFO RNG with longest period of generation. *w*P(*z*)/*n* shows the 1 s to 0 s ratio in the characteristic polynomial equation. Even if all the other parameters are equal, large n vale indicates good quality, while *w*P(*z*)/*n* must be nearly equal to 0.5. But, it is difficult to interpret on an absolute scale because all parameters are relative in nature. The highest quality is provided by the LUT-FIFO RNG. It has a *w*P(*z*)/*n* ratio of .5, which indicates the goodness of characteristic polynomial and the superior quality. The existing LUT-SR generator provides *w*P(*z*)/*n* ratio of .45. The modified LUT-SR RNG provides a*w*P(*z*)/*n* ratio .48. From this it is clear that the quadratic residue method improves the quality of modified LUT-SR RNG. This is done using the TestU01 [12], a software package used for the testing of random number generators and numbers

5. Conclusion

Generally, FPGA optimized generators generates high quality, long period random sequences. LUT-FIFO generator provides long period but it uses a block of RAM. LUT-OPT RNG needs only minimum resources when compared with LUT-FIFO generator. LUT is configured as Shift register in modified LUT-SR RNG. The modified LUT-SR RNG provides a good balance between quality and performance. The degree of randomness is increased in the modified LUT-SR RNG with minimum resource utilization.

References

[1] D. B. Thomas and W. Luk, "High quality uniform random number generation using LUT optimized state-transition matrices," *J. VLSI Signal Process*, vol. 47, no. 1, pp. 77–92, 2007.
[2] D. B. Thomas and W. Luk, "FPGA-optimized high-quality uniform random number generators," *in Proc. Field Program. Logic Appl. Int. Conf.*, pp. 235–244, 2008.
[3] D. B. Thomas and W. Luk, "FPGA-optimized high-quality uniform random number generators using luts and shift registers," *in Proc. Field Program. Logic Appl. Int. Conf.*, pp. 77-82, 2010.
[4] D. B. Thomas and W. Luk, "The LUT-SR Family of Uniform Random Number Generators for FPGA Architectures", *IEEE transactions on very large scale integration (vlsi) systems*, Vol. 21, No. 4, April 2013.
[5] K.H. Tsoi, K. H. Leung and P.H.W. Leong, "Compact FPGA-based True and Pseudo Random Number Generators", *IEEE Int. Symposium on Field-Programmable Custom Computing Machine*, 2003.
[6] P. Alfke," Efficient Shift Registers, LFSR Counters, and Long Pseudo-random Sequence Generators", *Technical Report, Xilinx, Inc.*, 1996
[7] P. H. Bardell, W. H. McAnney and J. Savir, "Build-in Test for VLSI: Pseudo-random Techniques", John Wiely and sons, 1987.
[8] Pong P Chu and Robert E Jones, "Design Techniques of FPGA based Random Number Generator", Available: www.citeseer.ist.psu.edu.1999
[9] C. Subbaramireddy, B. Srinath, "Uniform Random Number Generators using LUT-SR for Novel FPGA Architectures", *in Proc. Digital Signal and Image Processing (DSIP)*, 05th May-2013, Bhubaneswar.
[10] D. S. Monisha, R. Shantha Selva Kumari, "Implementation of RNG in FPGA using Efficient Resource Utilization", in Proc. *IJRTE*, Volume-2, Issue-2, May 2013.
[11] P. L'Ecuyer, "Tables of maximally equi-distributed combined LFSR generators", in Proc. *Math.Comput.*, vol.68, no. 225, pp. 261–269, 1999
[12] P. L'Ecuyer and R. Simard, TestU01 Random Number TestSuite[Online](2007).Available:http://www.iro.umontreal.ca/~imardr/indexe.html