High Speed LUT-SR Family of Random Number Generation

Remya Justin
PG Student
Department of Electronics & Communication Engineering
Saintgits College of Engineering

Binu K Mathew
Associate Professor
Department of Electronics & Communication Engineering
Saintgits College of Engineering

Susan Abe
Head of the Department
Department of Electronics & Communication Engineering
Saintgits College of Engineering

Abstract

FPGA Optimized RNGs are more resource efficient than software based RNGs. One type of FPGA RNG called LUT-SR RNG in which LUTs are configured into shift registers with varying length. LUT-SR generators are used to achieve high quality, long period RNG with minimum resources. Time delay is a factor which determines the bit rate. The aim of this work is to reduce the time delay of the LUT-SR RNG to provide high speed of operation. The paper describes long period LUT-SR RNG with high bit rate by combining identical blocks of short period LUT-SR RNGs. The proposed generator gives same maximum length random sequence with lesser time delay when compared with the existing LUT-SR RNG. This modified generator provides good quality generator than the existing generator.

Keywords: Field Programmable Gate Array (FPGA), Look Up Table (LUT), Random Number Generator (RNG), Shift Register (SR)

I. 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 efficient in terms of resources than software RNGs which means that can take the advantages of bit wise operation and parallelism. In particular, uniform random numbers are cheap to generate in FPGA using Look Up Tables (LUT)[1] or First In First Out (FIFO) queues. These generators can be customized based on the requirements of application. FPGA optimized RNGs are not widely used in practice because the construction of generator with given parameters is time consuming. Faced with this problem, engineers under time constraint choose less efficient methods 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 specific 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. This paper describes how to create LUT-SR RNG with lesser time delay, since high data rate is an essential quality needed for good RNGs. The main contributions of this paper are as follows:

1) A technique for creating r-bit LUT-SR RNG using smaller blocks of s-bit LUT-SR RNGs where r = s*u, u is the number of smaller LUT-SR blocks.
2) Hardware implementation of generators in the Spartan3E architecture.

The overview of random number generators are presented in the first part of Chapter 2, followed by the description of existing FPGA optimised random number generators. Chapter 3 gives the proposed system description. Simulation results are discussed in Chapter 4. Finally, Chapter 5 gives the conclusion.

II. LITERATURE REVIEW

A. Binary Linear RNGs:

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

$$x_{i+1} = Ax_i \quad (2.1)$$
$$y_{i+1} = Bx_{i+1} \quad (2.2)$$

Where $x_{i=}(x_{i,1},\ldots\ldots,x_{i,n})^T$ is the n -bit state of the generator, $y_{i=}(s_{i,1},\ldots\ldots,s_{i,r})^T$ is the r -bit output of the generator, A is an n × n binary transition matrix, and B is an r × n binary output matrix. 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 impossible to choose matrix A such that $x_0 = 0$ maps to any state other than $x_1 = 0$. This implies that there are two sequences 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 necessary and sufficient condition for a generator to have maximum period.

B. LUT – Optimised (LUT-OPT) RNGs:

In this family of generators, each row and column of matrix A contains t-1or t 1s. Each row maps to t or t-1 input XOR gate [1]. Each new state bit can be calculated using single LUT and r-bit generator can be implemented using r- fully utilized LUT-FFs. The structure of a 4- bit LUT-OPT generator is shown in Fig.2.1.

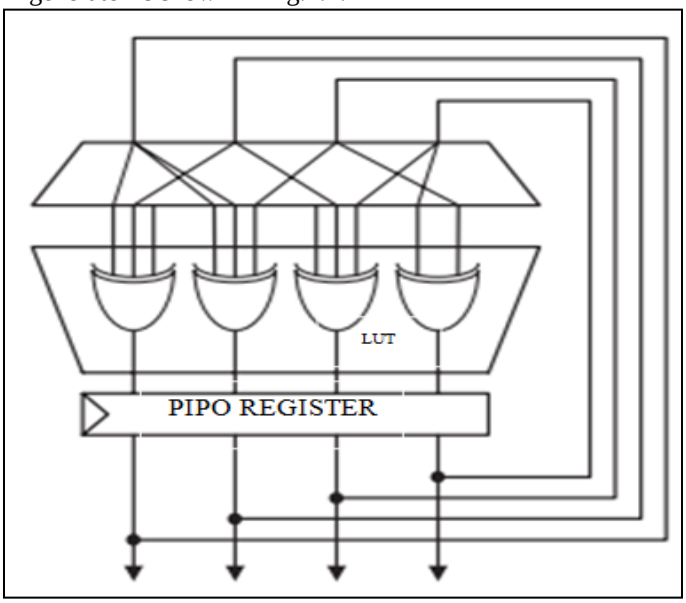

Fig. 2.1: 4-bit LUT-OPT RNG

An example of a maximum period LUT-OPT generator with r = 6 and t = 3 is shown by the recurrence

$$\mathbf{A} = \begin{bmatrix} 0&1&1&0&0&0 \\ 0&1&1&0&0&1 \\ 0&1&0&1&0&0 \\ 1&0&0&0&1&0 \\ 1&0&0&0&0&1 \\ 1&0&0&1&1&0 \end{bmatrix}, \quad \begin{bmatrix} x_{i+1,1} \\ x_{i+1,2} \\ x_{i+1,3} \\ x_{i+1,4} \\ x_{i+1,5} \\ x_{i+1,6} \end{bmatrix} = \begin{bmatrix} x_{i,2} \oplus x_{i,3} \\ x_{i,2} \oplus x_{i,3} \oplus x_{i,6} \\ x_{i,2} \oplus x_{i,4} \\ x_{i,1} \oplus x_{i,5} \\ x_{i,1} \oplus x_{i,6} \\ x_{i,1} \oplus x_{i,4} \oplus x_{i,5} \end{bmatrix}.$$

Merits of LUT-OPT RNG

1) Efficient resource Utilization: Every bit requires one LUT and FF, so resource usage 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, so usually the clock net is the limiting factor, with routing delay and congestion only becoming a factor for large n.

However, these merits are countered by a number of demerits

1) Complexity: It is difficult to construct a unique matrix for each (r, t) combination requires a unique matrix of connections, without using specialized software. It is difficult to encode these matrices, if these are constructed randomly.
2) Quality: The random bits are formed as a linear combination of random bits produced in the previous cycle. The new bits will be obtained simply by the XOR-ing of input bits from the previous cycle. This leads to this lag-1 linear dependence.
3) Period: To achieve a period of 2n - 1, it is necessary to choose r = n, even if far fewer than n bits are needed per cycle. The minimum safe period for a hardware generator is 264− 1. But much larger periods are preferred.

4) Seeding: Different hardware instances of same algorithm provide different random sequences if we initialize with different chosen states. For a randomly selected matrix A, parallel loading is the only possible way. The serial loading is not much practical.

C. LUT-First In First Out (LUT-FIFO) RNGs:

One solution for removing the quality and period problems faced by LUT-OPT RNG is to provide LUT-FIFO generators [2]. Here, these generator includes an additional depth- k width-w first-in-first-out (FIFO), giving a total period of 2^n- 1, where n = r + w k, shown in Fig.2. LUT-FIFO generators can provide long periods 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.

D. LUT- Shift Register (LUT-SR) RNGs:

LUT-based shift registers are very cheap-almost as cheap as the LUTs used to build the XOR gates. So it now becomes economical to use r shift registers, one per output bit, increasing the potential state to n = r (1 +k). It avoids all the problems related to complexity and serial seeding found with LUT-OPT and LUT-FIFO RNGs. LUT-SR RNG [4] provides much higher period than LUT-OPT RNG at the expense of one extra LUT-FF per bit. It eliminates need of the block- RAM needed for LUT-FIFO RNG.

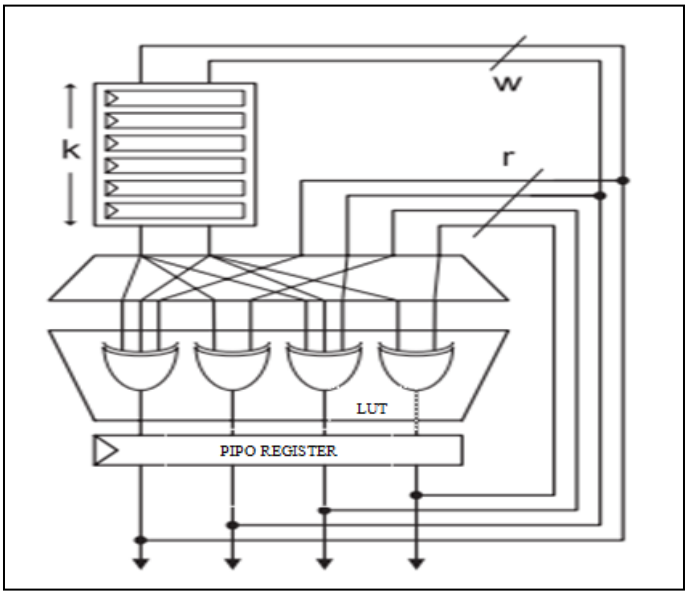

Fig. 2.2: 4-Bit LUT-FIFO RNG

Each of the r shift registers [6] can be assigned some specific length $k_i \leq k$, reducing the state size to $n = \sum_{i=1}^{r} (1 + k_i)$. One solution would be to randomly configure each shift register as $k_i = k$ or $k_i = k - 1$, giving a period $r*k < n < r(1 + k)$. But a much more interesting solution is to randomly choose $1 < k_i \leq k$, subject to the constraint $\exists\, i, j : i \neq j \wedge \gcd(k_i + 1, k_j + 1) = 1$. This allows for much more rapid mixing between bits within the state, while still providing necessary (but not sufficient) conditions for mixing within the state [5].
The algorithmic steps are as follows:

1) Create Initial Seed Cycle: A cycle of length r is created through the 'r' numbers of XOR gates at the output of the RNG. At this stage, there are 'r' numbers of FIFOs of length 0.
2) FIFO Extension: Random selection of a FIFO and increasing its length by 1. The cycle is randomly extended maintaining the known cycle, until a total cycle length is reached.

3) Add Loading Connections: the known cycle is added to the graph "taps," which describes the matrix A. The cycle describes the FIFO connections completely, and also describes the first input to each of the 'r' numbers of XOR gates.
4) Add XOR Connections: One input for each of the XOR gates in the generator is provided by the cycle. The additional t – 1 random inputs for the XOR gates are added over t − 1 rounds. Each round is constructed from the FIFO outputs, which ensures that at the end each FIFO output is used at most t times. Some bits will be assigned the same FIFO bit in multiple rounds, and so will have fewer than t inputs: this is critical to achieve a maximum period generator, and also provides us with an entry point into the cycle for seed loading.
5) Output Permutation: The final output permutation is used to avoid the dependency between the adjacent bits.

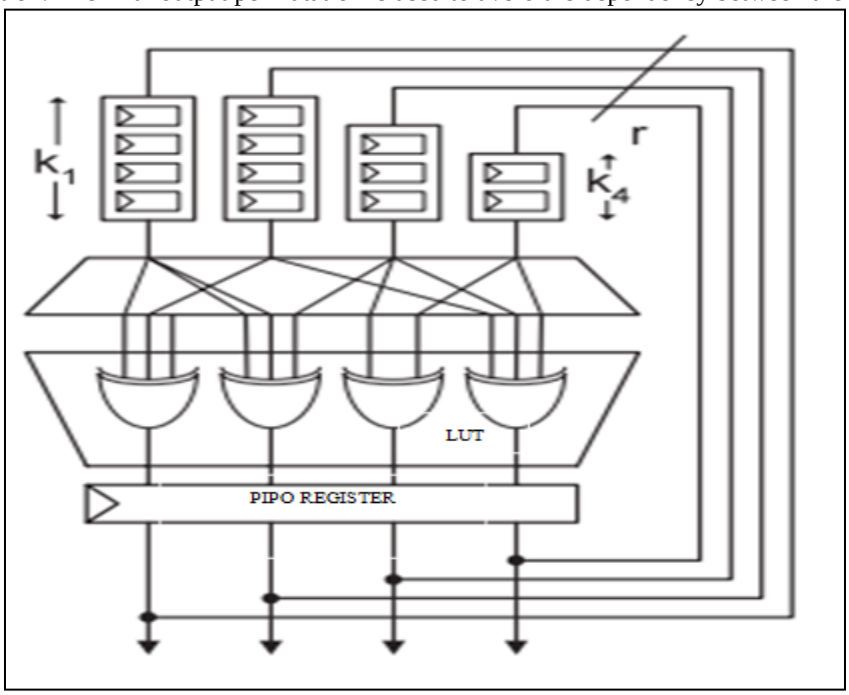

Fig. 2.3: 4-bit LUT-SR RNG

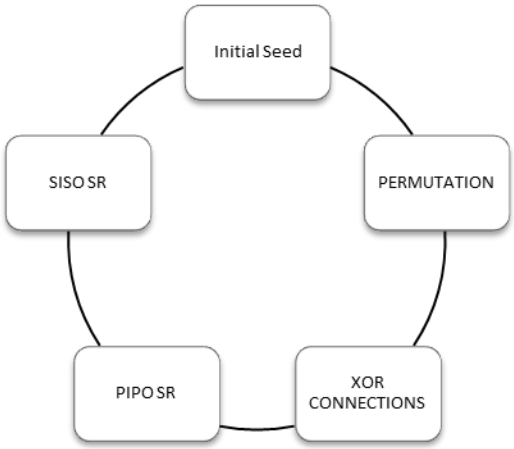

Fig. 2.4: Algorithmic Flow

III. PROPOSED SYSTEM DESCRIPTION

Generally, high speed operation is a desirable quality [7] of any system. Minimum completion time emphasizes the quality of random number generator. The proposed system aims to decrease the completion time of LUT-SR RNG without sacrificing the merits of LUT-SR generators. The modified LUT-SR RNG is much faster than the existing LUT-SR RNG [9].

In the proposed system, r-bit LUT-SR RNG is made by using small blocks or units of s-bit LUT-SR RNGs where r= s*u, u is the no. of small blocks needed. Each block act as the fundamental unit for the creation of proposed LUT-SR RNG. In the proposed system, even if each block having same no. of output bits, the depth of consecutive shift registers are varied. The difference in depth of SR is varied to provide much more randomness. The proposed system is created in such a way as to make the period as same as the existing methodology. The proposed system will provide high speed of operation than existing LUT-SR RNG with a small increase in resource utilization.

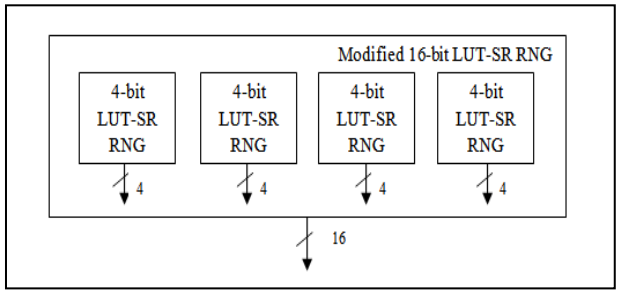

Fig. 3.1 Modified 16-bit Cascaded LUT-SR RNG

The block diagram (Fig.5) shows an example of modified LUT-SR RNG. In the following example, a 16-bit LUT-SR RNG is created using 4 blocks of 4-bit LUT-SR RNG. Each block is constructed using the algorithm describing LUT-SR RNG.
The figure below shows the architecture of basic 4-bit LUT-SR basic block.

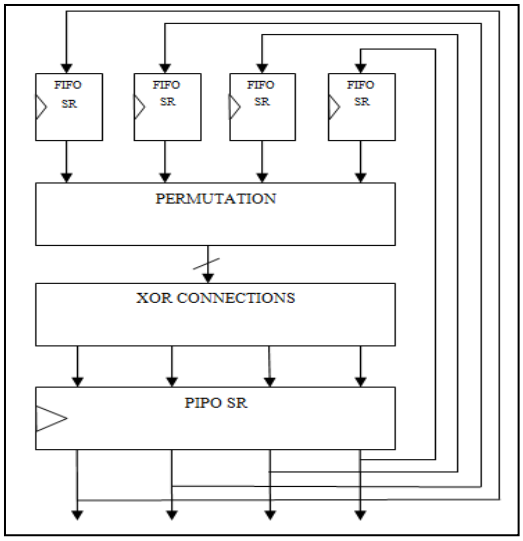

Fig. 3.1: Architecture of 4-bit LUT-SR Block

IV. RESULTS AND DISCUSSION

In the proposed system, the initial seed for each of the 4-bit LUT-SR RNG block is given through PIPO SR. A shift register is an n-bit register that shifts its stored data by one bit position for every clock tick. The resulting sequence in each PIPO SR is fed back to the SISO SR and given to the XOR gates. The output of the XOR gates and then given to the PIPO SRs, where the XOR gate outputs are shifted. The outputs of PIPO SRs are permuted and thus random number generation takes place successfully. The Random Number Generation is performed as per the methodology. The simulations are performed in Model Sim 6.4a which is a tool and synthesized using Xilinx PlanAhead Virtex5 kit verified on the Spartan 3E kit and the programming is written using VHDL. The results that are obtained from the tools and the design summary obtained from Xilinx 14.5i.

The initial seed is given as input. The seed is permuted. The results for 16-bit LUT-SR RNG using 4 blocks of 4-bit LUT-SR RNG are discussed below. The same scheme is carried out for 32-bit LUT-SR RNG using 4 blocks of 8-bit LUT-SR RNG. For a 16-bit generator, the no. of XOR gates is 16. The obtained XOR gate output bits are fed in a parallel basis into the PIPO SR and then permutation is performed. The concept of permutation is used up for improving randomness among bits and thus employing unpredictability. 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 reduced since the all zero state leads to idle condition. The period is the duration after which the entire sequence goes on repeating based on the initial seed and the permutations.

Table - 4.1
. Comparison between existing LUT-SR RNG and proposed Cascaded LUT-SR RNG Based on Delay

No. of bits	*Delay(ns)*	
	Existing LUT-SR RNG	*Proposed LUT-SR RNG*
16	*3.590*	*3.492*
32	*4.134*	*3.988*

Table 4.1 shows the comparison between existing LUT-SR RNG and proposed LUT-SR RNG based on the total delay of operation. Comparison is carried out among 16-bit and 32-bit generators. From the above table it is clearly evident that the delay parameter associated with the cascaded LUT-SR based RNG is less when compared to the existing system. Table I implies that the proposed methodology for the generation of random numbers is faster than the existing methodology.

Table 4.2 provides an overall comparison between 16-bit LUT-OPT RNG, LUT-FIFO RNG, LUT-SR RNG and proposed LUT-SR RNG. All the FPGA-optimized generators provide the best performance in terms of quality versus resources. Amongst the lower period generators, the LUT-OPT generator uses the minimum resources of only single LUT per generated bit, but it is far less efficient than the LUT-SR. The LUT-FIFO generator can provide very long periods to match those of the Mersenne Twister, but requires the use of a block RAM. The LUT-SR and proposed LUT-SR generators provide a useful mid-point between the two, with a good balance between resource utilization and good quality. This table shows that the inherent properties of the LUT-SR generators are also present in the proposed system.

Table - 4.2
Overall Comparison

OVERALL COMPARISON(16-bit Generator)	*n*	*Resources used*		
		RAM	*LUT*	*FF*
LUT-OPT	*16*	*0*	*16*	*16*
LUT-FIFO	*336*	*1*	*47*	*52*
LUT-SR	*256*	*0*	*23*	*32*
Cascaded LUT-SR (Proposed)	*256*	*0*	*32*	*33*

In short, the cascaded LUT-SR based generator can generate random numbers at high speed than the existing one. But, the resource utilization of cascaded design is much greater than the existing LUT-SR generator. If speed is the prime factor of consideration a cascaded LUT-SR based RNG can be used at the expense of excess resources. The quality of the generator is not compromised in this cascaded design.

V. CONCLUSION

This paper presents improved LUT-SR RNGs with implication on FPGA. These RNGs take the advantage of LUTs to configure as independent shift registers, for realizing high-quality long-period generators. The proposed methodology provides a high speed generator than existing LUT-SR generator. The advantages of this LUT-SR generator are the reduction in the complexity than LUT-FIFO RNG, faster speed, huge time period of pattern repetition & no requirement of RAM block. The algorithm used for the construction of LUT-SR RNG is simple and this allows FPGA Engineers to use the new RNGs without needing to find generator instances themselves. The proposed cascaded LUT-SR RNG is a high quality, high speed random number generator at expense of excess of resources.

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] C. Subbaramireddy, B. Srinath, "Uniform Random Number Generators using LUT-SR for Novel FPGA Architectures", International Conference on Digital Signal and Image Processing (DSIP), 05th May-2013, Bhubaneswar.
[9] D. S. Monisha, R. Shantha Selva Kumari, "Implementation of RNG in FPGA using Efficient Resource Utilization", IJRTE, Volume-2, Issue-2, May 2013.
[10] P. L'Ecuyer, "Tables of maximally equi-distributed combined LFSR generators," Math. Comput., vol. 68, no. 225, pp. 261–269, 1999