International Journal of Scientific Engineering and Research (IJSER)
www.ijser.in
ISSN (Online): 2347-3878
Volume 2 Issue 3, March 2014

Linear Congruential Generator for LUT-SR Architecture

Mary Evanchalin .S[1], Arulmozhi .P[2]

PG Scholar, Department of ECE, Nandha Engineering College, Erode-52, Tamil Nadu, India

Assistant Professor, Department of ECE, Nandha Engineering College, Erode-52, Tamil Nadu, India

Abstract: ***A random number generator (RNG) is a device designed to generate a sequence of numbers or symbols that don't have any pattern. Hardware-based systems for random number generation are widely used, but often fall short of this goal, albeit they may meet some of the statistical tests for randomness for ensuring that they do not have any "de-cod able" patterns. In the existing work, they proposed LUTs as shift registers to achieve high quality and long periods, while requiring very few resources. In defining the LUT-SR generators, the provision of a serial load chain is explicitly taken into account, by embedding a chosen cycle into the matrix A from the start. Specifically, we embed a very simple cycle of the form i ← (i + 1)mod r through the XOR bits. In the enhancement work, we proposed enhanced LUT – SR architecture with a Linear Congruential Generator (LCG) represents one of the oldest and best known pseudorandom number generator algorithms. The theory behind them is easy to understand, and they are easily implemented and fast. Experimental result shows performance level of our proposed architecture. In this, we implement our architecture in VLSI platform. Here our design was made by VHDL programming language by using Xilinx software.***

Keywords: Linear Congruential Generator, field-programmable gate array (FPGA), uniform random number generator (RNG).

1. Introduction

In the past, the major concerns of the VLSI designer were area, performance, cost and reliability; power consideration was mostly of only secondary importance. In recent years, however, this has begun to change and, increasingly, power is being given comparable weight to area and speed considerations. Several factors have contributed to this trend. Perhaps the primary driving factor has been the remarkable success and growth of the class of personal computing devices (portable desktops, audio- and video-based multimedia products) and wireless communications systems (personal digital assistants and personal communicators) which demand high-speed computation and complex functionality with low power consumption.

The low power interests are driven both by evolutionary and revolutionary trends. It is estimated that in the next five years about 50% of the electronic market will be in the portable system, while keeping the heat generation under the control to avoid forced cooling. The low power requirement thus calls for global solution. The solution can be classified into two faces, supply and demand. On the supply side we need better denser and smarter batteries efficient power conversion and regulation, improved heat dissipation, distribution and cooling technique etc., on the other hand we try to reduce the demand for low power by better processes and device technique efficient computation structures design technique etc.

Field programmable gate arrays are used in Monte Carlo application because of their highly parallel nature of the application, and because it is possible to take advantage of hardware features to create very efficient random generators (RNGs). In particular, uniform random bits are extremely cheap to generate in an FPGA, as large numbers of bits can be generated per cycle at high clock rates using lookup tables, or first-in-first-out (FIFO) queues. In addition, these generators can be customized to meet the exact requirements of the application, both in terms of the number of bits required per cycle, for the FPGA architecture of the target on platform. Despite these advantages, FPGA-optimized generators are not widely used in practice, as the process of constructing a generator for a given parameterization is time consuming, in terms of both developer man hours and CPU time. While it is possible to construct all possible generators ahead of time, the resulting set of cores would require many megabytes and be difficult to integrate into existing tools and design flows. Faced with these unpalatable choices, engineers under time constraints understandably choose less efficient methods, such as understandably choose less efficient methods, such as combined Taus-worthe generator or parallel linear feedback shift registers (LFSRs).

However, while the FIFO in a LUT-FIFO RNG is usually an expensive block RAM, LUT-based shift registers are very cheap, as cheap as the LUTs used to build the XOR gates. So it now becomes economical to use the shift registers, one per output bit.

2. LUT-SR RNG

2.1 Our Contributions

In recent years some researchers have developed FPGA Optimized random number generator which is one of the family of uniform random number generator with a matrix A where each row and each column contains t - 1or tones. In hardware terms this means that each row maps to a t-1or t input XOR gate, and so can be implemented in a single t-input LUT. Thus if the current vector state is held in a register, the new vector state can be calculated in a single LUT, and an r-bit generator can be implemented in r fully utilized LUT-FF's. FPGA contains different types of storage

elements, such as block RAM's, distributed RAM's, and shift-registers. All of these can be configured to act as fixed-length FIFO's, which delay data for some fixed number of cycles. To extend the period of a generator, it need to add state, and as well as the flip flops in logic elements. The uniform random numbers are designed by using the LUT-SR RNGs (Lookup table-Shift Register Random number generator).In hardware, there are two different methodologies such as interleaved parallelization (IP), chunked parallelization (CP)for parallelizing long period RNG's. The proposed LUT-SR random number generator provides a middle ground between the LUT-Opt generator and LUT-FIFO generator, by using cheap bit-wise shift-registers to provide long period and duty cycle without requiring expensive resources.

2.2 LUT-SR RNG

The architecture of the proposed LUT-SR random number generator is shown in Figure 3.1. The main goal of the proposed method is to achieve the maximum period of P=22048 – 1(i.e., P=2n – 1).It can be defined as how much amount of time the sequence is repeated itself is called the period. The repeatability is one of the major requirements of the random number generator. Although this may seem to contradict randomness it is a virtue that, using the same initial conditions, the sequence exactly repeats itself. The lookup table is a memory with a one bit output that essentially implements a truth table where each input Combination generates a certain logic output. The input combination is referred to as an address. The HDL synthesizer implements an AND gate by programming the stored elements in a LUT.

Consider the row of the generator is 64 bit and the depth of the FIFO queue is 31bit, the number of stages in shift register increases to n=2048. This provides a potential period of P=22048 – 1 for a cost of 1 LUT, and 16 flip flops as compared to a previous methods such as LUT-OPT random number generator and LUT-FIFO random number generator. The four stages of the developed uniform random number generator are given as follows.

Figure 3.1: Stages of Random Number Generator

2.3 Create initial seed cycle

A cycle of length *r* is created through the *r* XOR gates at the output of the RNG. FPGA optimized pseudo uniform random number generator with a large period and with the ability to generate large quantities of uniform random numbers from a single seed. At this stage there are no FIFO bits, or equivalently there are r FIFOs of length is zero is shown in Figure 3.2.

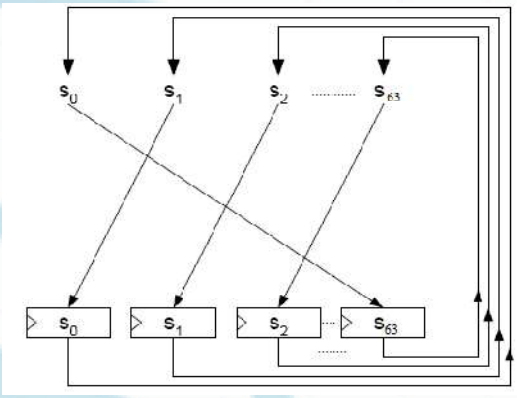

Figure 3.2: Create seed cycle

2.4 FIFO Extension

The cycle is randomly extended until a total cycle length of *n* is reached, by randomly selecting a FIFO and increasing its length by 1, while maintaining the known cycle is shown in Figure 3.3. AFIFO is a sequential data buffer that is very easy to use. Very small FIFOs can be implemented with flip-flops or register arrays, sometimes even with shift registers.

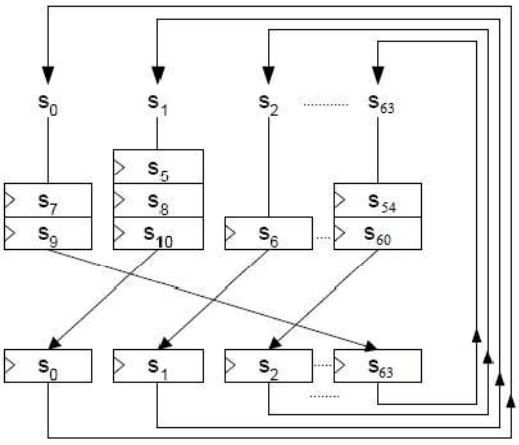

Figure 3.3: Randomly extended FIFO

2.5: Add XOR Connections

The cycle provides one input for each of the XOR gates, so now the additional t – 1random inputs are added over t−1rounds. Each round is constructed from a permutation of the FIFO outputs, which ensures that at the end each FIFO output is used at most t times is shown in Figure 3.4.

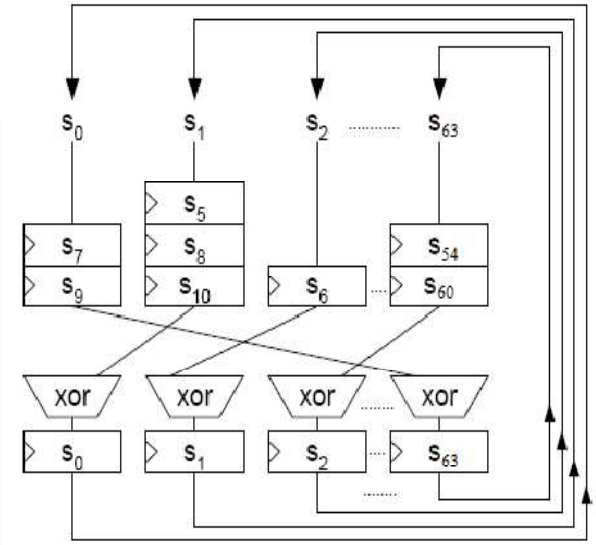

Figure.3.4: Add XOR Connections

A FIFO is a sequential data buffer that is very easy to use. Very small FIFOs can be implemented with flip-flops or register arrays, sometimes even with shift registers.

2.6 Output permutation

The simple dependency between adjacent bits is masked using a final output permutation is shown in Figure 3.5. Each permuted output bit is used at most times. Some bits will be assigned the same FIFO bit in multiple rounds. The XOR outputs are given to the PIPO SR and fed back to the FIFO extensions

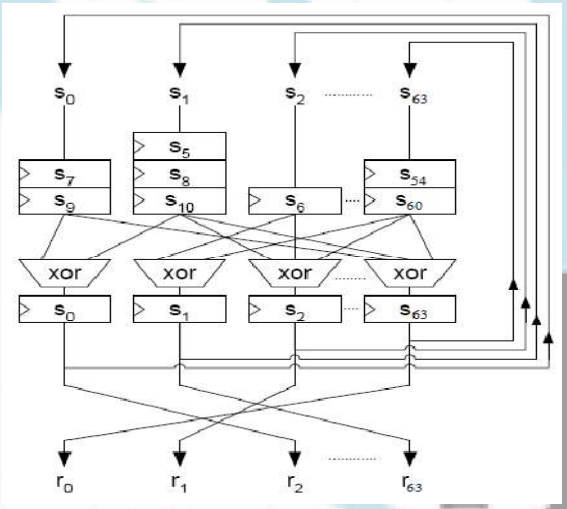

Figure 3.5: Output Permutation

3. Proposed Work

In proposed work, enhanced LUT – SR architecture with a Linear Congruential Generator (LCG) represents one of the oldest and best known pseudorandom number generator algorithms. The theory behind them is easy to understand, and they are easily implemented and fast. Experimental result shows performance level of our proposed architecture. 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:

Xi+1 = (A*Xi + C) mod M
m,0<m — the "modulus“
a, 0<a<m — the "multiplier"
c,0<c<m — the "increment“
Xo,0<Xo<m — the "seed" or "start value"

4. Results and Discussion

The initial seed for an 8-bit RNG is given or triggered 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 is fed back to the SISO SR or FIFOSR. Permutation of the resulting outputs is given to the XOR gates, where the XOR gate outputs are shifted and thus random number generation takes place successfully.

The results for 8-bit RNG are discussed in figure. The same scheme is carried out for 64 bit RNG. The permuted bits output is given to the XOR gates. For 8-bit RNG the number of XOR gates is 8(t=8). The concept of permutation is used up for improving randomness among bits and thus employing unpredictability. The first and last bits are interchanged. The same concept of permutation is used for different bit RNGs. The permuted outputs are fed into the XOR gates and for remaining inputs to XOR gates round basis is used. 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 resulting random numbers are generated such that their period is 2r - 1.If the number of bits is 16, then the period is 216-1. 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. So, the period for 32, 64, 128 and 512 bit RNGs are 232-1, 264-1, 2128-1, 2512-1. Register-Transfer-Level abstraction is used in VHDL languages for the formation of high level representation of the circuit and it clearly depicts the amount of LUTs used.

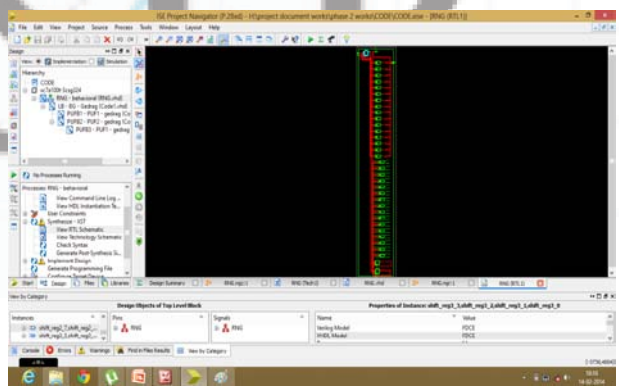

Figure 4: RTL schematic view

The technology schematic depicts the exact number of LUTs and FFs considered in fig

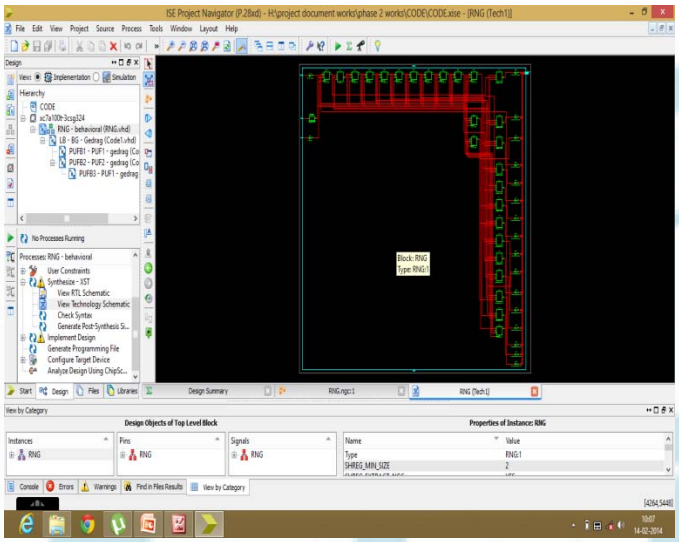

Figure 5: Technology schematic

As each bit crosses the flip-flop, it will be set to zero. The count of all zero state is reduced since the all zero state leads to idle condition.

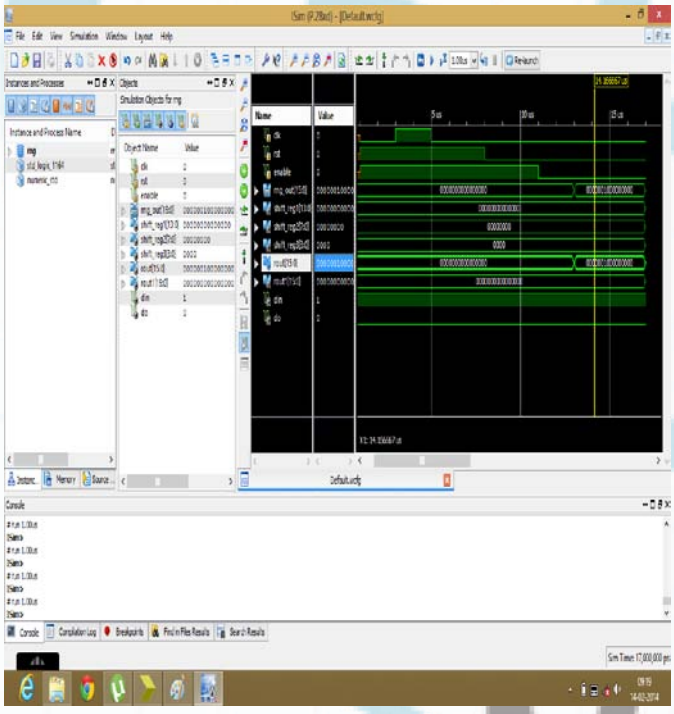

Figure 6: Output waveform

4.1 Device Utilization Summary

The device utilization summary results for 16-bit, RNG shows the number of (resources) flip-flops and LUTs utilized.

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	16	9,312	1%
Number of 4 input LUTs	33	9,312	1%
Number of occupied Slices	18	4,656	1%
Number of Slices containing only related logic	18	18	100%
Number of Slices containing unrelated logic	0	18	0%
Total Number of 4 input LUTs	33	9,312	1%
Number of bonded IOBs	19	190	10%
Number of BUFGMUXs	1	24	4%
Average Fanout of Non-Clock Nets	3.18		

Figure 7: Device utilization summary for 16-bitRNG

The device utilization summary table is displayed by Xilinx Design Suite soon after the RTL implementation is completed. The number of flip-flops utilized for 8-bit RNGs are 8 in number. The resource usage has considerably been reduced compared to the existing methodology. The number of LUTs has also been reduced in the work based on the proposed architecture.

4.2 Performance Comparison

The RNGs have the ability to configure LUTs as independent shift registers and require less amount of logic. The key point of the LUT-SR generators over previous FPGA optimized uniform RNGs is that they can be reconstructed using a simple algorithm. The number of LUT and FF utilized for 512-bit RNGs are 740 and 75 compared to the existing methodology's resource usage of 1024 and1024 for 512-bit RNGs as shown in Table.

4.3 Comparison between existing and proposed methodology

Existing Methodology		*Proposed Methodology*	
LUT	FF	LUT	FF
16	16	19	8
32	32	33	16

4.4 Overall Comparison Chart

The system of resource efficient RNG is described using the algorithm. Thus by the use of the simplified algorithm random numbers are generated successfully with the period of 2r -1. The overall comparison of all random number generators, in case of Lookup table and Flip flop is shown in Figure.

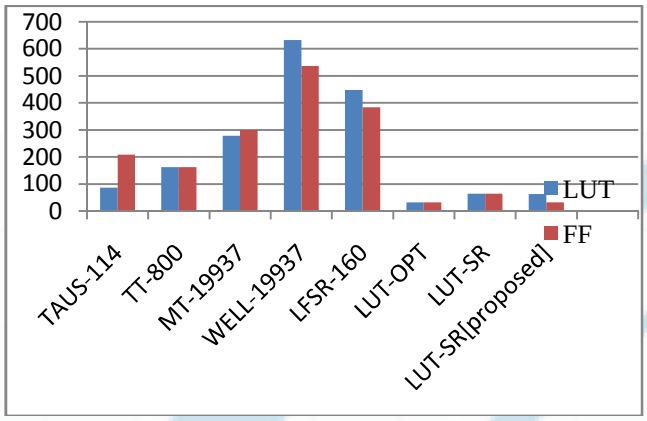

Figure 8: Overall Comparison Chart

The overall resource usage increases linearly as the number of bits increases. Compared to the existing RNGs the resource usage has reduced thus leading to improvement in resource efficiency.

5. Conclusion and Future Work

5.1 Conclusion

In this existing system, a family of FPGA optimized uniform RNGs, called LUT-SR RNGs is generated using a simplified algorithm. The RNGs have the ability to configure LUTs as independent shift registers and require less amount of logic. The key point of the LUT-SR generators over previous FPGA optimized uniform RNGs is that they can be reconstructed using a simple algorithm. This paper presented a family of FPGA-optimized uniform RNGs, called LUT-SR RNGs. These RNGs take advantage of the ability to configure LUTs as independent shift registers, allowing high-quality long-period generators to be implemented using only a small amount of logic. In addition, the period and quality scale with the number of output bits, unlike generators adapted from software. A key advantage of the LUT-SR generators over previous FPGA-optimized uniform RNGs is that they can be reconstructed using a simple algorithm, which is contained in this paper. In concert with the tables of maximum period generators, this allows FPGA engineers to use the new RNGs without needing to find generator instances themselves.

5.2 Future Enhancement

A generator called Linear Congruential Generator (LCG) is introduced in LUT-SR random number generator, which represents one of the oldest and best known pseudorandom number generator algorithms. The theory behind them is easy to understand, and they are easily implemented and fast. Experimental result shows performance level of our proposed architecture.

Reference

[1] D. B. Thomas and W. Luk, "High quality uniform random number generation using LUT optimised state-transition matrices," *J. VLSI Signal Process.*, vol. 47, no. 1, pp. 77–92, 2007.
[2] D. B. Thomas and W. Luk, "FPGA-optimised high-quality uniform random number generators," in *Proc. Field Program. Logic Appl. Int.Conf.*, 2008, pp. 235–244.
[3] P. L'Ecuyer, "Tables of maximally equidistributed combined LFSR generators," *Math. Comput.*, vol. 68, no. 225, pp. 261–269, 1999.
[4] D. B. Thomas and W. Luk, "FPGA-optimised uniform random number generators using luts and shift registers," in *Proc. Int. Conf. Field Program. Logic Appl.*, 2010, pp. 77–82.
[5] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator,"*ACM Trans. Modeling Comput. Simulat.*, vol. 8, no. 1, pp. 3–30,Jan. 1998.
[6] M. Saito and M. Matsumoto, "SIMD-oriented fast mersenne twister: A128-bit pseudorandom number generator," in *Monte-Carlo and Quasi-Monte Carlo Methods*. New York: Springer-Verlag, 2006, pp. 607–622.
[7] F. Panneton, P. L'Ecuyer, and M. Matsumoto, "Improved long-period generators based on linear recurrences modulo 2," *ACM Trans. Math. Software*, vol. 32, no. 1, pp. 1–16, 2006.
[8] M. Matsumoto and Y. Kurita, "Twisted GFSR generators II," *ACM Trans. Modeling Comput. Simulat.*, vol. 4, no. 3, pp. 254–266, 1994.
[9] P. L'Ecuyer and R.Simard. (2007). *TestU01 Random Number Test Suite* [Online]. Available: http://www.iro.umontreal.ca/~imardr/indexe.html.
[10] F. Panneton, P. L'Ecuyer, and M. Matsumoto, "Improved long-period generators based on linear
[11] recurrences modulo 2," *ACM Trans. Math.*
[12] *Software*, vol. 32, no. 1, pp. 1–16, 2006.
[13] V. Shoup. (1997, Jan. 15). *NTL: A Library for Doing Number Theory* [Online]. Available: http://www.shoup.net/ntl/
[14] M. Albrecht and G. Bard. (2010). *The M4RI Library - Version 20100817* [Online]. Available: http://m4ri.sagemath.org
[15] S. Duplichan. (2003). *PPSearch: A Primitive Polynomial Search Program* [Online]. Available: http://users2.ev1.net/~sduplichan/ primitivepolynomials/
[16] V. Sriram and D. Kearney, "A high throughput area time efficient pseudo uniform random number generator based on the TT800 algorithm," in *Proc. Int. Conf. Field Program. Logic Appl.*, 2007, pp. 529–532.
[17] S. Konuma and S. Ichikawa, "Design and evaluation of hardware pseudorandom number generator mt19937," *IEICE Trans. Inf. Syst.*, vol. 88, no. 12, pp. 2876–2879, 2005.
[18] Y. Li, P. C. J. Jiang, and M. Zhang, "Software/hardware framework for generating parallel long-period random

numbers using the well method," in *Proc. Int. Conf. Field Program. Logic Appl.*, Sep. 2011, pp. 110–115.

Author Profile

S. Mary Evanchalin has completed her B.E (ECE). She is pursuing M.E (VLSI Design) in Nandha Engineering College, Erode, Tamil Nadu, India