Implementation of RNG in FPGA using Efficient Resource Utilization

D. S. Monisha, R. Shantha Selva Kumari

Abstract- Computers' required random numbers initially, for simulations and numerical computations like Monte Carlo calculations. Random number generators offer an important contribution to many communication systems for security. They are critical components in computational science. However the tradeoff between quality and computational performance is an issue for many numerical simulations. FPGA optimized RNGs are efficient in terms of resources than other types of software-based RNGs which means that they can take advantage of bitwise operations and FPGA based specific features. One of the types of FPGA based RNG called a LUT-SR RNG is illustrated using an algorithm. Shift registers are used to improve mixing rate between numbers. Results will be misleading when correlations exist between the random numbers and hence permutations are used. The LUTs are configured into shift registers. The algorithm is simplified based on the architecture such that it ensures longer periods. A generator with a period of $2^r - 1$ can be implemented and provides r random output bits. This provides a good quality balance compared to previous generators. The critical path between all registers is a single LUT. The program is run in ModelSim 6.4a and implementation is done using Xilinx PlanAhead Virtex5 kit.

Index Terms–random number generator (RNG), field programmable gate arrays (FPGA), SIMD, Look up table, Shift Register(LUT SR).

I. INTRODUCTION

Most of the RNG mathematicians analyze on the basis of software terms, which can be proved from their designed algorithms. They assume that the execution in the existing environment is word-based and sequential. FPGAs are based on bit-by-bit operations and are rigidly parallel. Therefore mapping the algorithms from a software based environment to hardware based environment leads to inefficient conditions.The inefficiency might result in several problems including untrackable rate of the generator, lower clock rates, partial usage of resources, lesser usage of RAMS, nonuniform word lengths.
Considering the untrackable rate of the generator many of the best RNGs use the rejection method, which do not form a new number for every execution of the main RNG iteration. This creates a problem during simulation of hardware, which shows that it deals on some cycles where the RNG postpones the required downstream processing components and needs extra resources, and may also lead to failure of scheduling schemes as mentioned by Thomas[1,2].

Revised Manuscript Received on 30 May 2013.
* Correspondence Author
Ms.D.S.Monisha, M.E.(Communication Systems), Mepco Schlenk Engineering College, Sivakasi, India.
Dr.R.Shantha Selva Kumari, ECE Dept,Mepco Schlenk Engineering College, Sivakasi, India.

II. RELATED WORKS

Mersenne Twister (MT) is a widely-used fast pseudorandom number generator (PRNG), designed by Matsumoto [8].More CPU time is required for initialization than for generation in MT and hence, next to Mersenne Twisters, WELL generators were introduced by Panneton [9]. CPUs for personal computers later, acquired new features of SIMD operations (i.e., 128- bit operations) and multi-stage pipelines.128-bit based PRNG was proposed which was named as SIMD-oriented Fast Mersenne Twister (SFMT), which is analogous to MT using SIMD operations proposed by Saito[7]. Tsoi[10] mentioned that if the function call is avoided, WELL may be slower than MT for some CPUs.
The SFMT pseudorandom number generator is a very fast generator with satisfactorily high-dimensional equidistribution property. Then Random number generators based on linear recurrences modulo 2 were introduced. Linear Feedback Shift Register random number generators, also called Tausworthe generators, which work on linear recurrences modulo 2. Trinomial-based generators have important statistical defects, but combining them can yield generators that are relatively fast and robust. Such combinations have been proposed and analyzed by Matsumoto and Wang [11, 12].The generators given in are for 32-bit computers. Nowadays 64-bit computers are becoming increasingly common and so it is important to have good generators designed to fully use the 64-bit words given by P.L'Ecuyer [6].
The huge-period generators proposed thereafter were not quite optimal. New generators with better equidistribution and bit-mixing properties were required. The state of these generators evolves in a more chaotic way than for the Mersenne twister. The reduction of the impact of persistent dependencies among successive output values can be observed in certain parts of the period of the Mersenne twister which was given by Saito [7]. A generator with a period of $2^k - 1$ can be implemented using k flip-flops and k LUTs, and provides k random output bits each cycle.
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 combined Tausworthe generators [3] or parallel linear feedback shift registers (LFSRs).

III. SYSTEM DESCRIPTION

A) The LUT-SR RNG

The LUT-SR generators provide a middle performance between the LUT-Opt [1] and LUT-FIFO generator[1], by using cheap bit-wise shift-registers to provide long periods and good quality without requiring expensive resources. The number of bits generated per cycle is chosen generally to meet the needs of the application.

B) Algorithm

LUT-SR generator family uses a short and precise algorithm for expanding the full RNG structure. The algorithm [1] uses r, t and k with period $2^r - 1$ where r is the number of random output bits generated per cycle, t is the XOR gate input count, k is the maximum shift register length. The parameters (r, t, k) describe the properties of the generator in terms of application requirements and architectural restrictions. The algorithmic steps are as follows,

- ***Initial loading***

Initially the loading step is done by giving a seed. For r-bit generator the seed size is r .As soon as the seed is given the bits are permuted. Any seed other than "all-zero state" can be given. The seed is also known as initial seed. All zero state condition cancels random number generation and makes the generator idle[1].

- ***Permutation***

The simple dependency between adjoining bits is masked up using a final output permutation. The model is shown in Fig.1.

- ***Loading XOR connections***

The permuted outputs are given as inputs to XOR gates. The number of inputs should not exceed r. The number of rounds should be t or t-1[1], where t is the number of XOR gates given. Each permuted output bit is used at most t times. Some bits will be assigned the same FIFO bit in multiple rounds. The XOR- ed outputs are given to the PIPO SR and fed back to the FIFO extensions[1].

- ***PIPO SR***

Universal shift register performs shifting operation in addition to the parallel-in-parallel-out function. At a time multiple input processing happens in Parallel-in-parallel-out-shift register. The purpose of the parallel-in parallel-out shift register is to take in parallel data, shifts it, then output the data[1].

- ***FIFO Extension***

1-bit shift registers are used. Bitwise shift registers improve the rate of mixing [1]. For 8-bit RNG the length of FIFO SR should not exceed k, where k=8.The length of the shift register is given by the number of flip-flops. The outputs from PIPO SR are fed back to FIFO or SISO SR. 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[1].

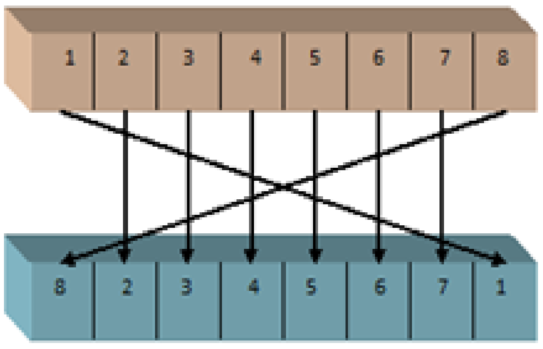


Fig.1. Permutation

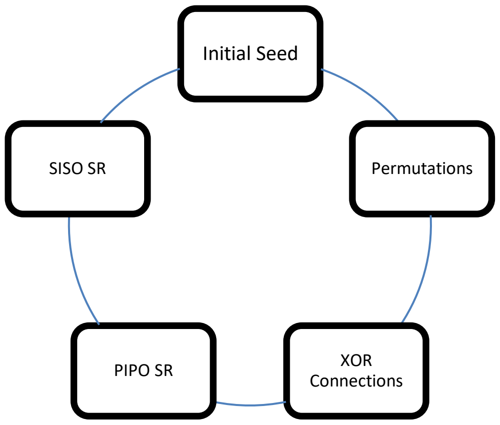


Fig. 2.Algorithmic flow

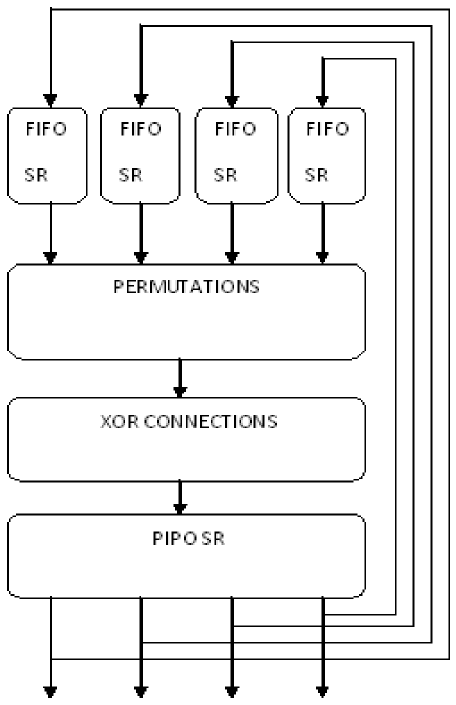


Fig. 3. Architecture

IV. RESULTS AND DISCUSSIONS

A) ModelSim Results

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 FIFO SR. Permutation of the resulting outputs is given to the XOR gates. The output of the XOR gates are then given to the PIPO SRs, where the XOR gate outputs are shifted 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 Verilog. The results that are obtained from the tools and the design summary obtained from Xilinx 8.1i are shown below.

The initial seed is given as input. The seed is permuted. The results for 8-bit RNG are discussed below. 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. Thus the obtained XOR gate output bits are fed in a parallel basis into the PIPO SR. 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 2^r -1.If the number of bits is 16, then the period is 2^{16}-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 2^{32}-1, 2^{64}-1, 2^{128}-1, 2^{512}-1.

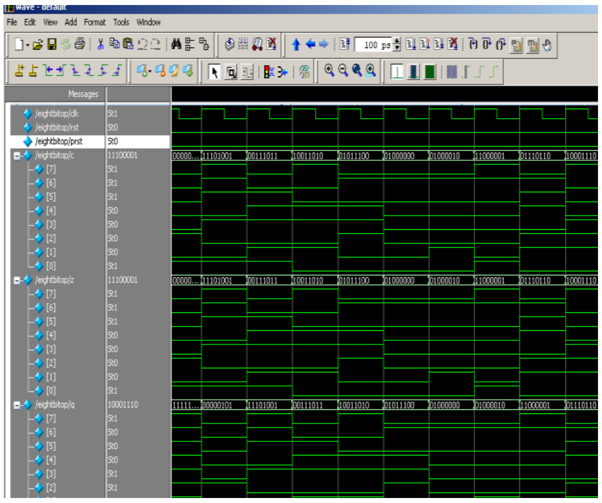


Fig.4. Simulation results for 8-bit RNG

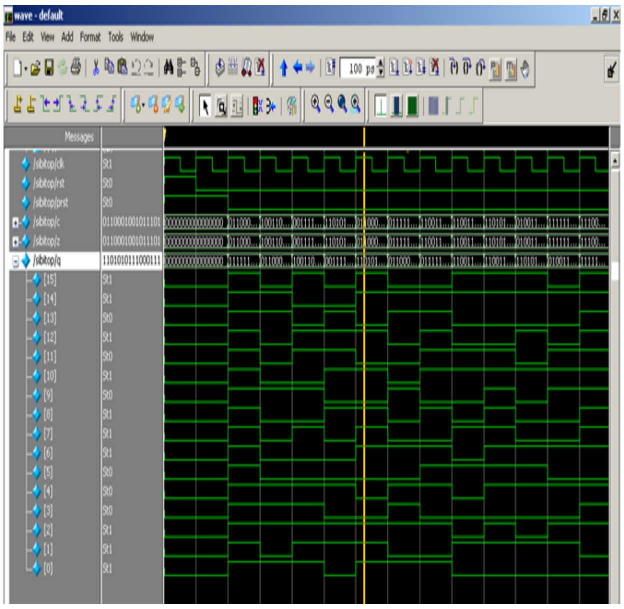


Fig. 5.Simulation results for 16-bit RNG

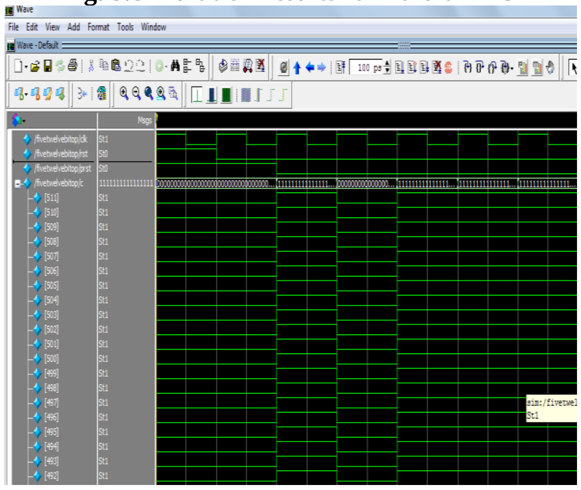


Fig. 6. Simulation results for 32-bit RNG

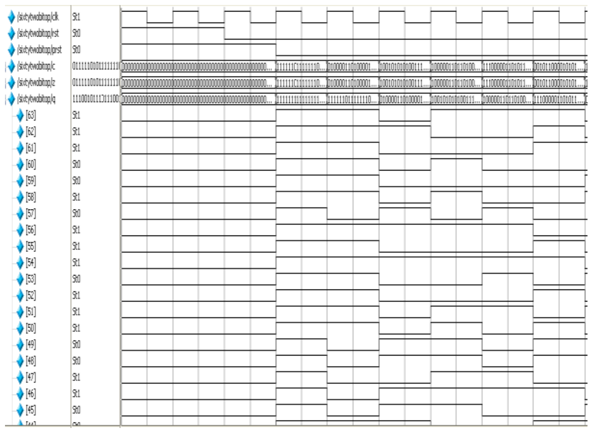


Fig.7. Simulation results for 64-bit RNG

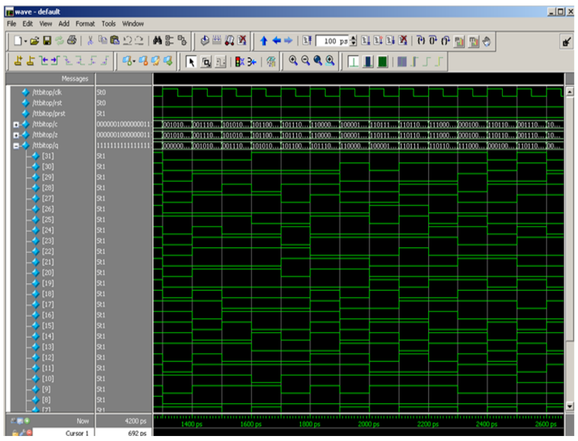


Fig. 8.Simulation results for 128-bit RNG

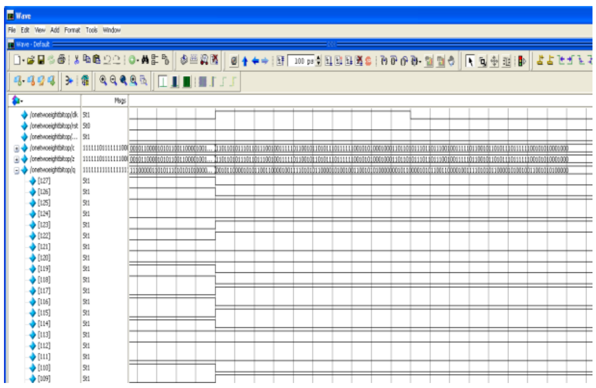


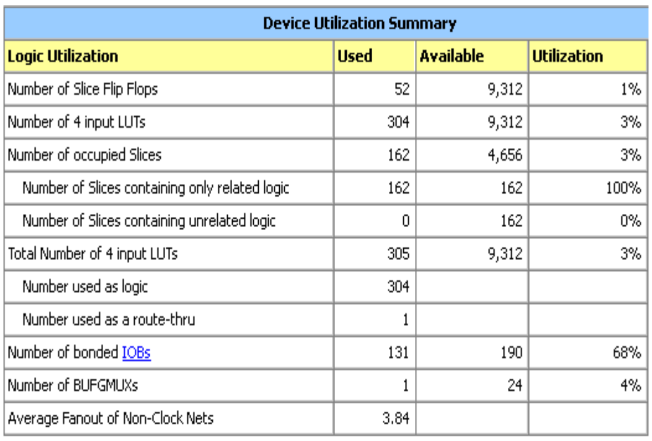
Fig. 9. Simulation results for 512- bit RNG

B) Device Utilization Summary

The device utilization summary results for 8-bit, 16-bit, 32-bit, 64-bit, 128-bit, 512-bit RNG shows the number of (resources) flip-flops and LUTs utilized.

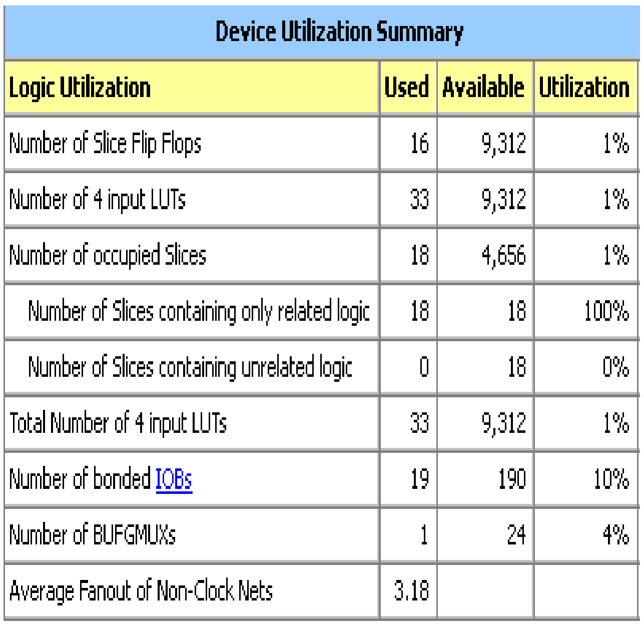
Number of errors: 0

Number of warnings: 0



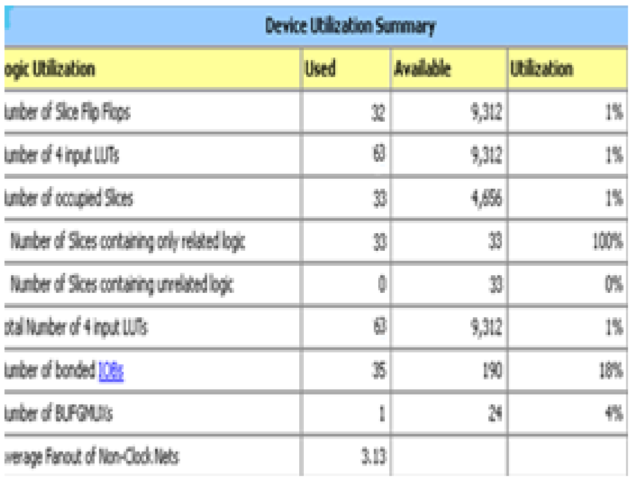
Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	52	9,312	1%
Number of 4 input LUTs	304	9,312	3%
Number of occupied Slices	162	4,656	3%
Number of Slices containing only related logic	162	162	100%
Number of Slices containing unrelated logic	0	162	0%
Total Number of 4 input LUTs	305	9,312	3%
Number used as logic	304		
Number used as a route-thru	1		
Number of bonded IOBs	131	190	68%
Number of BUFGMUXs	1	24	4%
Average Fanout of Non-Clock Nets	3.84		

Fig.10. Device utilization summary for 8-bit RNG



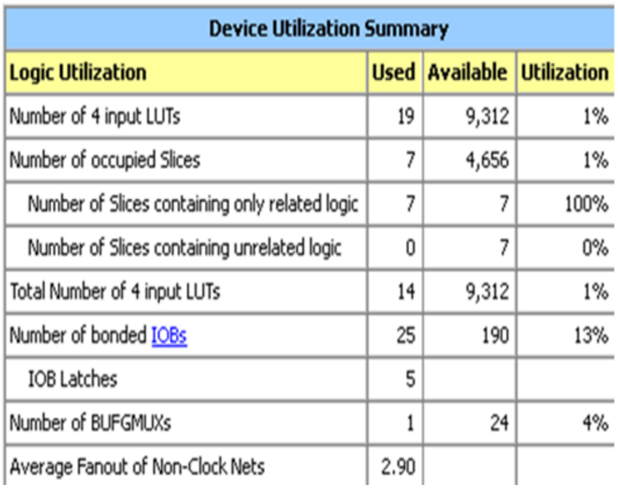
Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	16	9,312	1%
Number of 4 input LUTs	33	9,312	1%
Number of occupied Slices	18	4,656	1%
Number of Slices containing only related logic	18	18	100%
Number of Slices containing unrelated logic	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		

Fig. 11. Device utilization Summary for 16-bit RNG



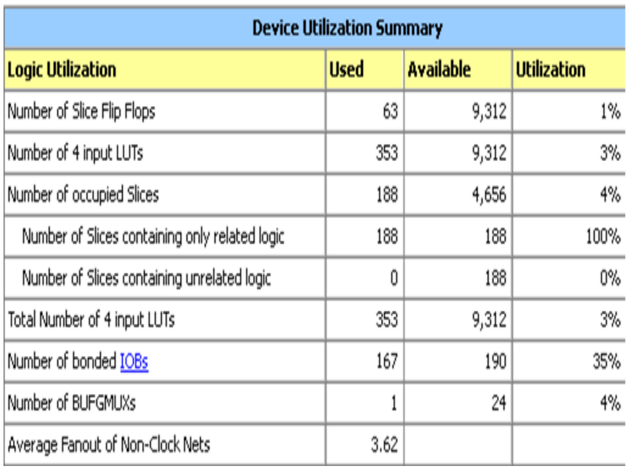
Device Utilization Summary			
ogic Utilization	**Used**	**Available**	**Utilization**
umber of Slice Flip Flops	32	9,312	1%
umber of 4 input LUTs	63	9,312	1%
umber of occupied Slices	33	4,656	1%
Number of Slices containing only related logic	33	33	100%
Number of Slices containing unrelated logic	0	33	0%
otal Number of 4 input LUTs	63	9,312	1%
umber of bonded IOBs	35	190	18%
umber of BUFGMUXs	1	24	4%
verage Fanout of Non-Clock Nets	3.13		

Fig.12. Device utilization summary for 32-bit RNG



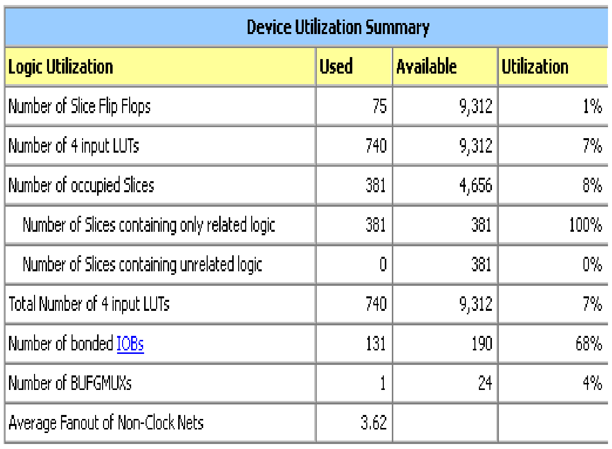
Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of 4 input LUTs	19	9,312	1%
Number of occupied Slices	7	4,656	1%
Number of Slices containing only related logic	7	7	100%
Number of Slices containing unrelated logic	0	7	0%
Total Number of 4 input LUTs	14	9,312	1%
Number of bonded IOBs	25	190	13%
IOB Latches	5		
Number of BUFGMUXs	1	24	4%
Average Fanout of Non-Clock Nets	2.90		

Fig.13. Device utilization summary for 64-bit RNG



Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	63	9,312	1%
Number of 4 input LUTs	353	9,312	3%
Number of occupied Slices	188	4,656	4%
Number of Slices containing only related logic	188	188	100%
Number of Slices containing unrelated logic	0	188	0%
Total Number of 4 input LUTs	353	9,312	3%
Number of bonded IOBs	167	190	35%
Number of BUFGMUXs	1	24	4%
Average Fanout of Non-Clock Nets	3.62		

Fig. 14.Device utilization summary for 128-bit RNG



Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	75	9,312	1%
Number of 4 input LUTs	740	9,312	7%
Number of occupied Slices	381	4,656	8%
Number of Slices containing only related logic	381	381	100%
Number of Slices containing unrelated logic	0	381	0%
Total Number of 4 input LUTs	740	9,312	7%
Number of bonded IOBs	131	190	68%
Number of BUFGMUXs	1	24	4%
Average Fanout of Non-Clock Nets	3.62		

Fig.15.Device utilization summary for 512-bit RNG

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. For 16-bit, 32-bit, 64-bit, 128-bit, 512-bit RNGs the number of flip-flops utilized is 16,32,52,63 and 75 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.

C) RTL Schematic

Register-Transfer-Level abstraction is used in verilog and VHDL languages for the formation of high level representation of the circuit and it clearly depicts the amount of LUTs used.The technology schematic depicts the exact number of LUTs and FFs considered.

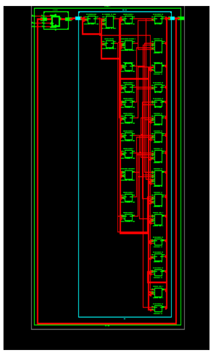


Fig.16. Technology Schematic- 8-bit RNG

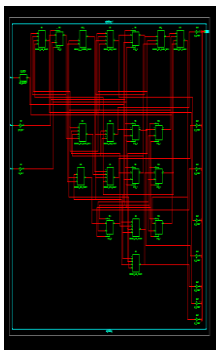


Fig.17.Xor connections- Technology schematic

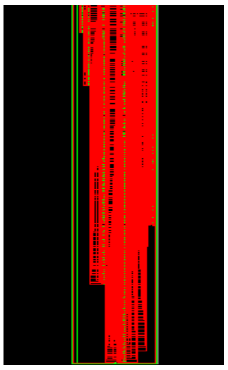


Fig.18.RTL schematic of 64-bit RNG

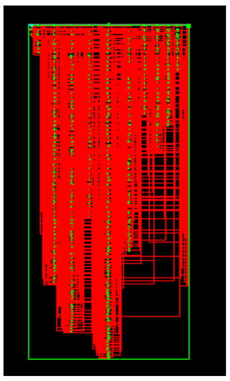


Fig.19.RTL Schematic of 128-bit RNG

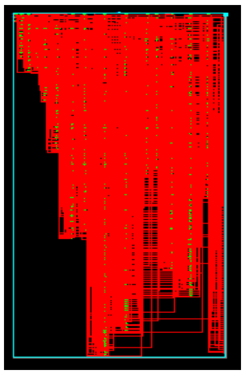


Fig.20. RTL Schematic of 512-bit RNG

D) Performance Comparison

TABLE I .COMPARISON BETWEEN EXISTING AND PROPOSED METHODOLOGY

Number of bits	Existing Methodology[1]		Proposed Methodology	
	LUT	FF	LUT	FF
8	16	16	19	8
16	32	32	33	16
32	64	64	63	32
64	128	128	204	52
128	512	512	353	63
512	1024	1024	740	75

TABLE II.OVERALL COMPARISON (32-BIT)

TABLE II.OVERALL COMPARISON (32-BIT) Generator	Resources used	
	LUT	FF
Taus-114[6]	87	208
TT 800[14]	162	162
MT-19937[15]	278	-
WELL 19937[16]	633	537
LFSR 160[4]	448	384
LUT-OPT[4]	32	32
LUT-SR[1]	64	64
LUT-SR[proposed]	63	32

V.CONCLUSION

Thus 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. 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 and 1024 for 512-bit RNGs.

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. 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 2^r-1.

REFERENCES

1. D. B. Thomas and W. Luk, "The LUT-SR Family of Uniform Random Number Generatorsfor FPGA Architectures," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, March 2012.
2. D. B. Thomas and W. Luk, "FPGA-optimized uniform random number generators using lut and shift registers," in Proc. Int. Conf. Field Program. Logic Appl., 2010, pp. 77–82.
3. D. B. Thomas and W. Luk, "FPGA- optimized high - quality uniform random number generators," in Proc. Field Program. Logic Appl. Int.Conf., 2008, pp. 235-244.
4. 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.
5. 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.
6. P. L'Ecuyer, "Tables of maximally equidistributed combined LFSR generators," Math.Comput., vol. 68, no. 225, pp. 261– 269, 1999.
7. M. Saito and M. Matsumoto, "SIMD-oriented fast mersenne twister: A 128-bit Pseudo random number generator," in Monte-Carlo and Quasi- Monte Carlo Methods. NewYork: Springer-Verlag, 2006, pp. 607–622.
8. 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.
9. 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.
10. K. H. Tsoi, K. H. Leung, and P. H. W. Leong, "Compact FPGA-based True Random Number Generators", in IEEE Symposium on FPGAs for Custom Computing Machines, IEEE Computer Society, Washington, DC, 2003,p. 51.
11. M. Matsumoto and Y. Kurita, "Twisted GFSR generators II," ACM Trans. Modeling Comput. Simulat., vol. 4, no. 3, pp. 254–266, 1994.
12. D. Wang and A. Compagner, On the use of reducible polynomials as random number generators, Mathematics of Computation 60 (1993), 363{374}. MR 93e:65012
13. P. L'Ecuyer and R. Simard, "TestU01 Random Number Test Suite", (2007).
14. 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.
15. 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.
16. 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

D. S. Monisha has completed B.E(ECE) and is pursuing M.E (Communication Systems) in Mepco Schlenk Engineering College, Sivakasi, TN. She has published paper in 1 IC.

Dr. R. Shantha Selva Kumari is working as Professor in Mepco Schlenk Engineering College, TN. So far she has published 15 IJ papers, 1 NJ paper, 38 IC papers and 37 NC papers.