International Journal of Electronics Communication and Computer Engineering
Volume 6, Issue 6, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

Optimization of FPGA Architecture for Uniform Random Number Generator using LUT-SR Family

Ms. Rita Rawate
M.Tech (Student), PCE, Nagpur, (M.S)
Email: rita.rawate@gmail.com

Prof. M. V. Vyawahare
Assistant Professor, PCE, Nagpur, (M.S)
Email: mvvyawahare@yahoo.co.in

***Abstract* – Field-Programmable Gate Arrays (FPGAs) are widely used to implement logic without going through an expensive fabrication process. Field-programmable gate array optimized random number generators (RNGs) are more resource-efficient than software-optimized RNGs because they can take advantage of bitwise operations and FPGA-specific features. The software community has developed a number of high-quality, long period Random Number Generators (RNGs), some of which have been adapted for use in FPGAs. However, these generators were designed to meet the needs of word-level instruction processors, and so are less efficient when mapped to the bit-level operations available in FPGAs. This paper describes a type of FPGA RNG called a LUT-SR RNG, which takes advantage of bitwise XOR operations and the ability to turn lookup tables (LUTs) into shift registers of varying lengths. This provides a good resource–quality balance compared to previous FPGA-optimized generators. This paper deals with optimization of FPGA and simulations is done in VHDL.**

***Keywords* – FPGA (Field Programming Gate Array), LUT (Look UP TABLE), LUT-SR (Look Up Table Shift Register),Uniform Random Number Generator (RNG).**

INTRODUCTION

MONTE CARLO applications are ideally suited to field- programmable gate arrays (FPGAs) because of the highly parallel nature of the applications, and because it is possible to take advantage of hardware features to create very efficient random number generators (RNGs). 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 [1], or first-in-first- out (FIFO) queues [2]. 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, and for the FPGA architecture of the target 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.

Random numbers have applications in many areas: simulation, game-playing, cryptography, statistical sampling, and evaluation of multiple integrals, particle transport calculations, and computations in statistical physics, to name a few[1,6]. Since each application involves slightly different criteria for judging the “worthiness” of the random numbers generated, a variety of generators have been developed, each with its own set of advantages and disadvantages. Many applications are reliant on random numbers, such as financial calculations, simulated equipment test beds, and simulation of communications channels. Such applications require large amounts of processing power, while providing many opportunities to exploit fine-grain and coarse-grain parallelism, and so are often ideally suited to implementation in FPGAs [5, 7]. In order to function correctly, these applications require many parallel streams of high quality, large period, uncorrelated uniform random number generators. These are most commonly used as input to transformation functions which will provide the non-uniform distributions, and typically require many uniform input bits for each nonuniform output sample [1,2]. This paper explains a family of generators which makes it easier to use FPGA-optimized generators by given a simple method instantiate an RNG. This helps to achieve the specific needs of their application. Specifically, it shows how to create a family of generators called LUT-SR RNGs, which use LUTs as shift registers to achieve high quality and long periods, while requiring very few resources.

This paper is structured as follows. Section II presents general idea of field-programmable gate array. Section III introduces uniform random number generator. Section IV explains lut-opt RNGs .Section V gives idea about lut-FIFO RNGs. Section VI introduces LUT-SR RNGs. Finally Section VII deal with device utilization summery. section VIII gives idea about comparision of generators by resourse usage and simulation results and synthesis report are given in section IX and X. Section XI concludes the paper.

II. FIELD PROGRAMMING GATE ARRAY (FPGA)

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing—hence "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC).

FPGAs can be used to implement any logical function that an ASIC can perform. The ability to update the functionality after shipping, partial re-configuration of the portion of the design and the low non-recurring engineering costs relative to an ASIC design, offer advantages for many applications[1,6]. FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "connected together"—somewhat like a one-chip programmable breadboard. Logic blocks can be

Copyright © 2015 IJECCE, All right reserved

configured to perform complex combinational functions, or merely simple logic like AND and NAND[8]. Figure 1 show The most common FPGA architecture which consists of an array of logic blocks (called Configurable Logic Block, CLB, or Logic Array Block, LAB, depending on vendor), I/O pads, and routing channels. Generally, all the routing channels have the same width (number of wires). Multiple I/O pads may fit into the height of one row or the width of one column in the array.

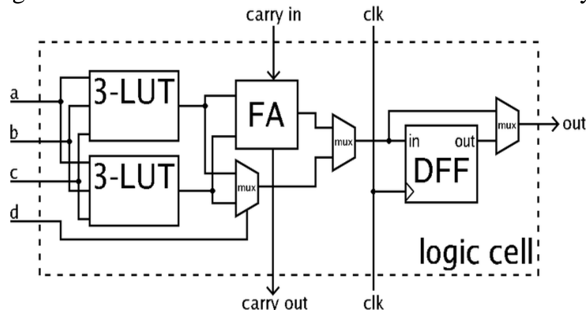


Fig.1. FPGA Architecture.

In general, a logic block (CLB or LAB) consists of a few logical cells. A typical cell consists of a 4-input Lookup table (LUT), a Full adder (FA) and a D-type flip-flop, as shown. The LUT are in this figure split into two 3-input LUTs. In normal mode those are combined into a 4-input LUT through the left mux. In arithmetic mode, their outputs are fed to the FA. The selection of mode is programmed into the middle mux. The output can be either synchronous or asynchronous, depending on the programming of the mux to the right.

III. Uniform Random Number Generator (RNG)

Random values play a crucial role in several areas of science. In dependency on field of application the requirements for parameters of random sequence and generator of sequence itself may vary. Focusing on the sequence origin 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-codable‖ patterns. Methods for generating random results have existed since ancient times, including dice, coin flipping, the shuffling of playing cards, the use of yarrow stalks and many other techniques.

The many applications of randomness have led to many different methods for generating random data. These methods may vary as to how unpredictable or random they are, and how quickly they can generate random numbers..

IV. LUT-Optimized (LUT-OPT) RNGs

A simple example of a maximum period LUT-OPT generator with $r = 6$ and $t = 3$ is given by

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

LUT-OPT generators have two key advantages.

1) Resource efficiency: Each additional bit requires one additional LUT and FF, so resource usage scales linearly, and generating r bits per cycle requires r LUT-FFs.

2) Performance: The critical path in terms of logic is a single LUT delay, so the generators are extremely fast, so usually the clock net is the limiting factor, with routing delay and congestion only becoming a factor for large n.

Some disadvantages of LUT-OPT generators are following:

1) Complexity: Each (r, t) combination requires a unique matrix of connections, which must be found using specialized software. If these matrices are randomly constructed (as in previous work), then it is difficult to compactly encode these matrices, so it is difficult for FPGA engineers to make use of the RNGs.

2) Quality: The random bits are formed as a linear combination of random bits produced in the previous cycle— when $t = 3$, some of the new bits will be a simple two-input XOR of bits from the previous cycle. The input of this lag-1 linear dependence is minimal in modern FPGAs where $t \geq 5$, and also diminishes quickly as r is increased, but remains a source of concern.

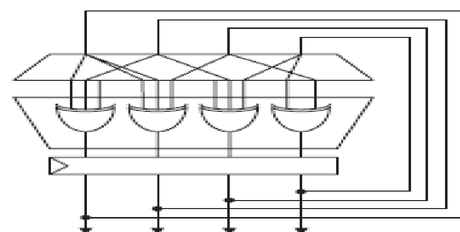


Fig.2. LUT-OPT RNG

3) Period: In order to achieve a period of $2^{n} - 1$, it is necessary to choose $r = n$, even if far fewer than n bits are needed per cycle. An absolute minimum safe period for a hardware generator is $2^{64} - 1$, but it is preferable to have much larger periods of $2^{1000} - 1$ or mor.

4) Seeding: It is necessary to initialize RNGs with a chosen state at run time, so that different hardware instances of the same RNG algorithm will generate different random streams. In a LUT-optimized generator, it is possible to implement serial loading of state using one LUT input per RNG bit to select between RNG and load mode, but in practice, for a randomly chosen matrix A, only parallel loading is possible.

V. LUT-FIFO RNGs

One way of removing the quality and period problems is provided by LUT-FIFO generators [2]. These augment

Copyright © 2015 IJECCE, All right reserved

International Journal of Electronics Communication and Computer Engineering
Volume 6, Issue 6, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

the r bits of state held in FFs with an additional depth-k width-w first-in-first-out (FIFO), for a total period of $2^n - 1$, where $n = r + wk$, shown in Fig.3. LUT-FIFO generators can provide long periods such as $2^{11213} - 1$ and 2^{19937}.

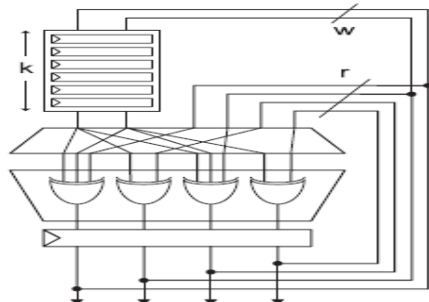


Fig.3. LUT-FIFO RNG

Some disadvantages are following:

1) For reasonable efficiency, the FIFO needs to be implemented using a block RAM, a relatively expensive resource which one would usually prefer to use else-where in a design.

2) The wordwise granularity of block-RAM-based FIFOs reduces the flexibility in the choice of r, as it can only be varied in multiples of k.

These are mild disadvantages when compared to the quality and period problems of LUT-optimized generators that have been eliminated, but LUT-FIFO generators also make the problems of complexity and efficient initialization slightly worse. If extremely high quality and period are needed, then LUT-FIFO generators present the fastest and most efficient solution, but few applications actually require such high levels of quality, particularly given the need for expensive block- RAM resources.

VI. LUT-SR

LUT-SR generator sits between the LUT-optimized and LUT-FIFO generators. It fixes all problems related to complexity and serial seeding found with both generators, and provides much higher periods than LUT-OPT generators for a cost of one extra LUT-FF per bit, while eliminating the block- RAM resource needed for an LUT-FIFO RNG[7]. LUTs can be configured in a number of different ways, such as basic ROMs, RAMs, and shift registers. Configuring LUTs as shift registers provides an attractive means of adding more storage bits to a binary linear generator.

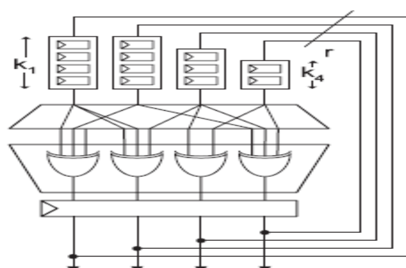


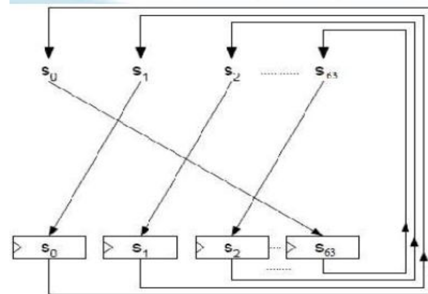
Fig.4. LUT-SR RNG

The algorithm will take input of 5-tuple as follows:

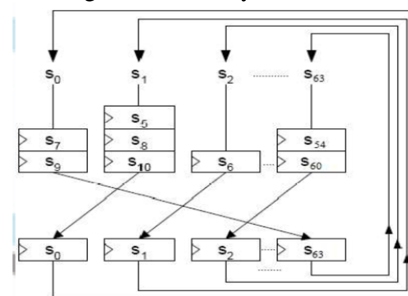
1) Number of state bits in the RNG.
2) Number of random output bits generated per cycle.
3) XOR gate input count[1,6].
4) Maximum shift register length.
5) Free parameter used to select a specific generator.

The RNG will be generated in few stages as follows:

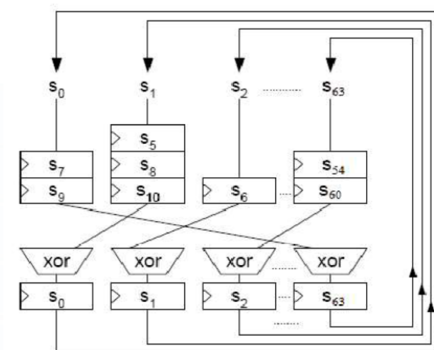
1) Initial Seed Cycle: A cycle will be created through the XOR gates at the output of the RNG.



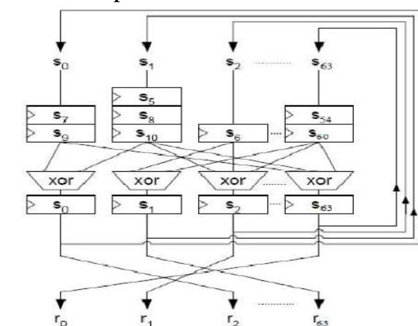
2) Cycle Extension: The cycle will be extended randomly while maintaining the known Cycle[4]



3) Input implication: The cycle will describe the input to each of the XOR gates.



4) Output implication: Each cycle will be constructed by alteration of the outputs.



Copyright © 2015 IJECCE, All right reserved

VII. Device Utilization Summery

The device utilization summary results for 8-bit, RNG shows the number of (resources) flip-flops and LUTs utilized

Table I: Comparison Of Generators By Resource Usage

RNG Type	Bits(8)	No. of slices	No.of flip-flop	No.of 4 input LUTs	frequency
LUT-OPT RNG	8 bit	9	16	8	471
LUT-FIFO RNG	8 bit	13	16	18	471
LUT-SR RNG	8 bit	19	22	22	494

VIII. Comparison of Generators by Resource Usage

The device utilization summary results for 8-bit, RNG shows the number of (resources) flip-flops and LUTs utilized. The device utilization summary table is displayed by Xilinx Design Suite soon after the RTL implementation is completed.

Table II: Device utilization summary of LUT-OPT RNG

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	9	1920	0%
Number of Slice Flip Flops	16	3840	0%
Number of 4 input LUTs	8	3840	0%
Number of bonded IOBs	19	141	13%
Number of GCLKs	1	8	12%

Table III: Device utilization summary of LUT-FIFO RNG

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	13	1920	0%
Number of Slice Flip Flops	16	3840	0%
Number of 4 input LUTs	18	3840	0%
Number of bonded IOBs	19	141	13%
Number of GCLKs	1	8	12%

Table IV: Device utilization summary of LUT-SR RNG

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	19	1920	0%
Number of Slice Flip Flops	22	3840	0%
Number of 4 input LUTs	22	3840	0%
Number of bonded IOBs	19	141	13%
Number of GCLKs	1	8	12%

IX. Simulation Result

The proposed method is simulated in VHDL .Figure 1 shows simulation result of LUT-OPT RNG. Figure 2 shows simulation result of LUT-FIFO RNG. and Figure 3 shows the simulation result of LUT-SR RNG respectively.

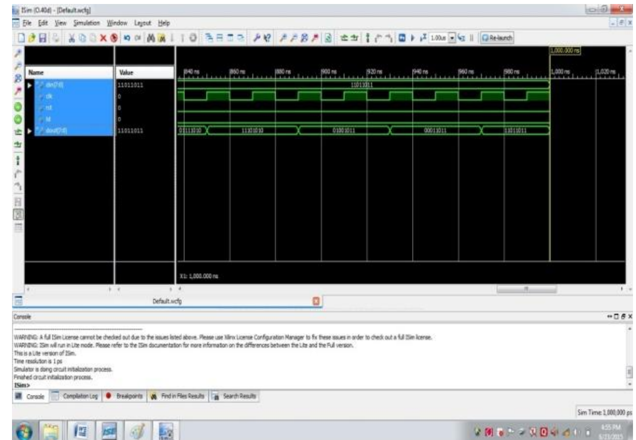


Fig.1. Simulation result of 8 bit LUT-OPT RNG

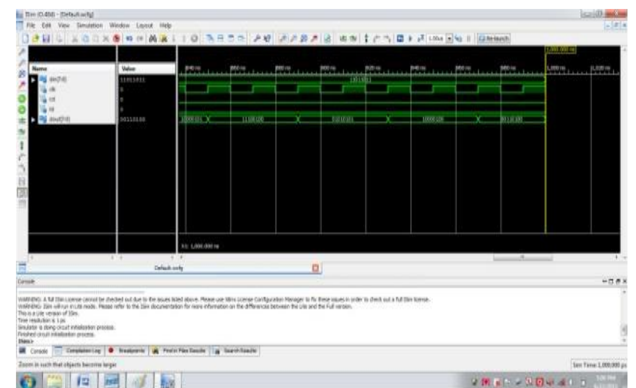


Fig 2.simulation result of LUT-FIFO RNG

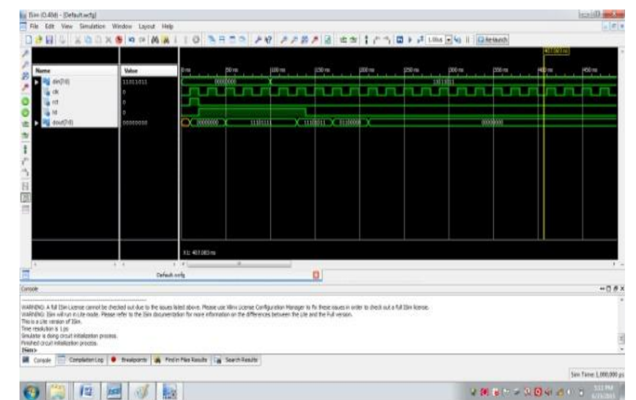


Fig. 3 . Simulation result of 8 bit LUT-SRRNG

X. Synthesis Results

A. LUT-OPT RNG

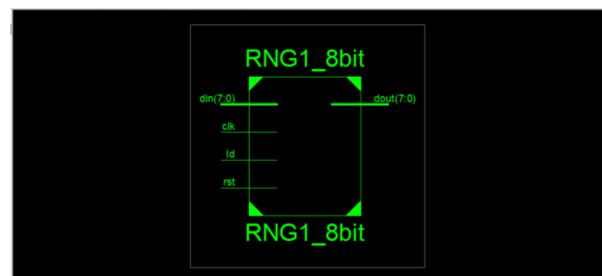


Fig.1. 8- Bit LUT-OPT RNG Block

Copyright © 2015 IJECCE, All right reserved

International Journal of Electronics Communication and Computer Engineering
Volume 6, Issue 6, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

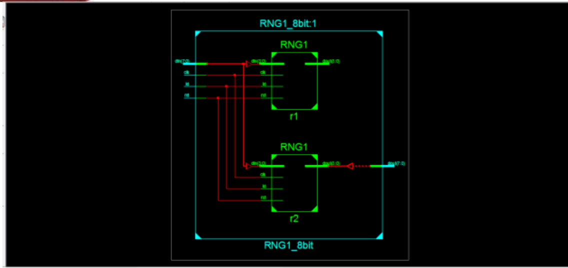


Fig.2. RTL Schematic Of 8- Bit LUT-OPT RNG

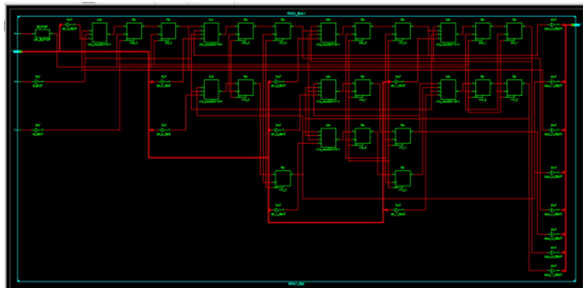


Fig.3. Technology Schematic Of 8 Bit LUT-OPT RNG

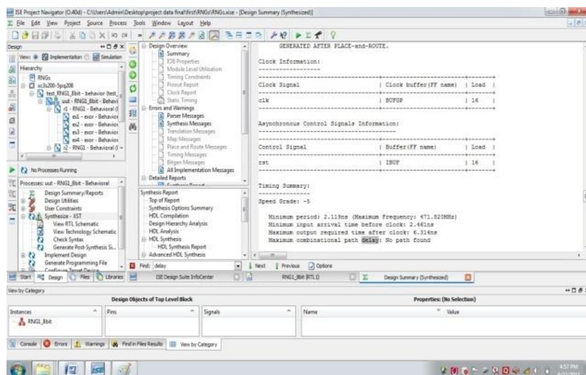


Fig.4. Delay of 8 bit LUT-OPT RNG

B. LUT-FIFO RNG

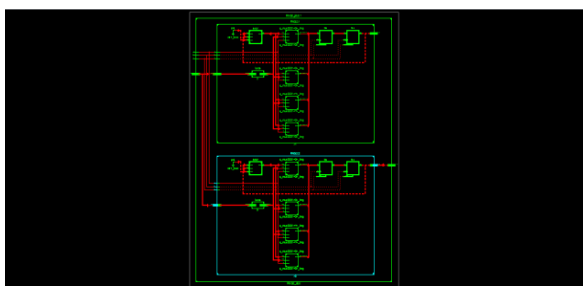


Fig.5. Technology Schematic Of 8 Bit LUT-FIFO RNG

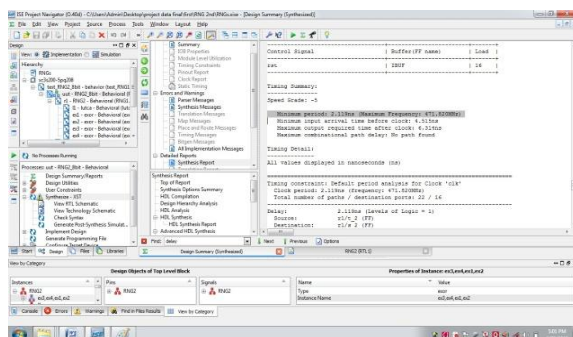


Fig. 6. Delay of 8- bit LUT-FIFO RNG

C. LUT-SR RNG

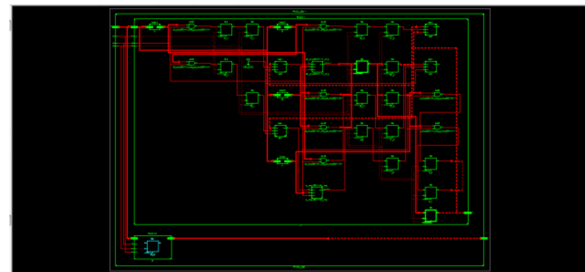


Fig.7. Technology schematic of 8 bit LUT-SR RNG

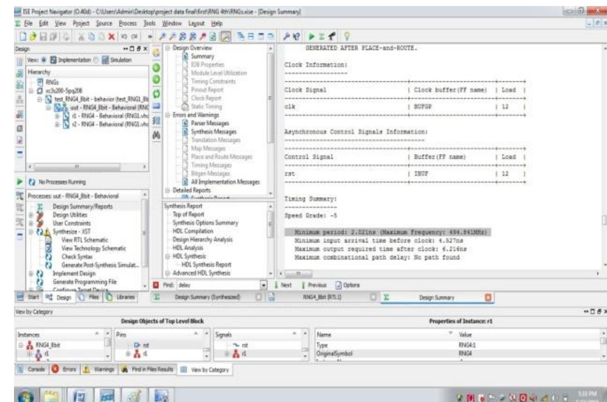


Fig.8. Delay of 8 bit LUT-SR RNG

XI. Conclusion

A family of FPGA-optimized uniform random number generator, called a LUT-SR RNG. LUT-SR RNGs takes 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 random number generators is that they can be reconstructed using a simple algorithm ,new RNGs without needing to find generator instances themselves.

This paper uses a hardware description language called VHDL to design LUT-OPT RNG, LUT-FIFO RNG and LUT –SR RNG. In this dissertation, strategies & implementation of different RNGs is described. The LUT-OPT RNG, LUT-FIFO RNG, LUT -SR RNGs coded in VHDL and VHDL code executing on the Xilinx ISE 13.1i VHDL tools.

References

[1] D. B. Thomas and W. Luk, "High quality uniform random number generation using LUT optimized state-transition matrices," in *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.

Copyright © 2015 IJECCE, All right reserved

[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: A 128-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 recurrences modulo 2," *ACM Trans. Math. Software*, vol. 32, no. 1, pp. 1–16, 2006.

[11] V. Shoup. (1997, Jan. 15). *NTL: A Library for Doing Number Theory* [Online]. Available: http://www.shoup.net/ntl/

[12] M. Albrecht and G. Bard. (2010). *The M4RI Library - Version 20100817*[Online]. Available: http://m4ri.sagemath.org

[13] S. Duplichan. (2003). *PPSearch: A Primitive Polynomial Search Program* [Online]. Available: http://users2.ev1.net/∼sduplichan/primitivepolynomials/

[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'S PROFILE

Rita S. Rawate

She born in Bhandara, (M.S) on May 7th 1986.
She completed her B.E (ECE). From Manoharhai Patel College Of Engineering & Technology, Gondia, (M.S)
She is pursuing her M.Tech in VLSI from Priyadarshini College of Engineering and Technology, Nagpur, Maharashtra, INDIA.

Copyright © 2015 IJECCE, All right reserved