International Journal of Software Engineering and Its Applications
Vol.7, No.5 (2013), pp.233-242
http://dx.doi.org/10.14257/ijseia.2013.7.5.21

Segmented Leap-Ahead LFSR Architecture for Uniform Random Number Generator

Je-Hoon Lee[1] and Seong Kun Kim[2]

[1]*Div. of Electronics, Information and Communication, Samcheock Campus, Kangwon National University*
[2]School of General Studies, *Samcheock Campus, Kangwon National University*
[1]*jehoon.lee@kangwon.ac.kr*, [2]*kimseong@kangwon.ac.kr*

Abstract

Random numbers are widely used in various applications. In the majority of cases, a pseudo-random number generator is used since true random number generators are slow and they are barely suitable for the hardware implementation. In this paper, we present new architecture of URNG (uniform random number generator) employing Leap-Ahead LFSR architecture for hardware implementation. In particular, the proposed URNG consists of two more segmented Leap-Ahead LFSRs to overcome the drawback of the conventional URNG employing Leap-Ahead architecture, that is, the sharp decrease of a maximum period of the generated random numbers. Thus, the proposed URNG with segmented LFSR architecture can generate multiple bits random number in a cycle without the frequent diminishing of maximum period of the generated random numbers. We prove the efficiency of the proposed segmented LFSR-architecture through the mathematical analysis. The simulation results show that the proposed URNG employing segmented Leap-Ahead LFSR architecture can be increased 2.5 times of the maximum period of generated random numbers compared to the URNG using the conventional Leap-Ahead architecture.

Keywords: *Uniform random number generator, Leap-Ahead, LFSR, FPGA*

1. Introduction

In general, random numbers are used in a wide variety of applications, in particular, cryptography. The many applications of randomness have led to the development of several different methods for generating random data. There are two different methods used to generate random numbers. One measures some physical phenomenon that is expected to be random and then compensates for possible biases in the measurement process. The other uses computational algorithms that can produce long sequences of apparently random numbers, which are in fact completely determined by a shorter initial value, known as a seed value or key. The latter case is often called pseudo-random number generator against true random number obtained from the former case. Pseudo-random numbers are very useful in developing cryptography so long as the seed keep secret. Sender and receiver can generate the same set of random numbers automatically to use as the secret key, seed. A pseudo-random number generator is the algorithm that can automatically produce the long runs of numbers with good random properties but eventually the sequence repeats.

The cryptography mechanisms can be categorized into symmetric key cryptography, asymmetric key cryptography and crypto hash function [1]. In particular, a symmetric key cryptography uses same secret key, that is, private key, to encrypt and decrypt the data, whereas an asymmetric key cryptography uses both public key and private one [2]. Both of

ISSN: 1738-9984 IJSEIA

them have a crucial problem in the key distribution and management. If the secret key, seed leak out, the original message shall be resolved by the cracker regardless of the cryptography mechanisms employed. Thus, the secret key must be long enough so that an attacker cannot try all possible combinations. In addition, the lack of randomness in the sequences of random numbers obtained from the RNG (random number generator) shall be disastrous. Thus, the random numbers generated by the key should contain sufficient entropy to prevent the cryptanalytic breaks. .

Most pseudo-random number generating algorithms involve complex mathematical operations and they are not suitable to be implemented in hardware owing to its complexity. There are less complex methods that can be implemented in hardware. In particular, A LFSR based RNG is one of the most popular algorithm since it could be easily described with HDL language as compact hardware [3-7]. The most common way to implement a random number generator in hardware is a LFSR based RNG that can generate single random bit per cycle. The ever growing complexities in the various applications require the RNG generating multiple random numbers per cycle. A multiple LFSR based RNG architecture is introduced to implement a URNG can generate multiple random bits per cycle. This architecture requires n different LFSRs to output n-bit random numbers per cycle. Even though this architecture can generate multiple random bits per cycle, it suffers from the hardware complexity. Gu and Zhang presented a Leap-ahead architecture has presented to solve this problem [6]. They introduced the architectures of RNG employing Leap-Ahead LFSRS of both Galois type and Fibonacci one. This RNG can generate an m-bits random number per cycle even though it requires only one LFSR. In spite of the advantages in hardware complexity, this architecture has significant drawback that is the decrease in the maximum period of generated random numbers. In this architecture, the maximum period of the generated random numbers can be kept as 2^n-1, when 2^n-1 and m could not be divided by each other. Otherwise, the maximum period of random numbers shall be smaller than 2^n-1. Here, n is the number of LFSR stages and m is the number of the output bits of the RNG.

This paper presents new architecture for Leap-Ahead LFSR architecture by employing the segmentation technique. The proposed architecture can solve the problem of the conventional Leap-Ahead architecture without significant area overhead. Nevertheless, it can generate the multi-bits random number per cycle. We prove the efficiency of the proposed segmentation technique using the precise mathematical analysis. We also demonstrate the proposed segmented Leap-Ahead LFSR architecture on Xilinx Vertex V FPGA so as to present the comparison results with other reported ones. The reset of this paper is organized as follows. Section 2 introduces the fundamentals of Leap-Ahead architecture. Section 3 presents the proposed segmented Leap-Ahead LFSR architecture. Section 4 describes the simulation results and we conclude the paper in Section 5.

2. Leap-Ahead LFSR Architecture

This section introduces the mathematical and practical characteristics of the LFSR based RNG, then we describe existing techniques for building multi-bits RNG including the conventional Leap-Ahead LFSR architecture.

In general, there are two different types of LFSRs. The one is Galois type and the other is Fibonacci one. Both of them generate single random number per cycle. The generated random numbers are considered as pseudo-random sequences since the sequence repeats itself after a certain number of cycles. It is known as the period of the random number generated.

The conventional Galois type LFSR is illustrated in Figure 1. An LFSR is a shift register whose input bit is a linear function of its previous state. It consists of n D-F/Fs and x_i is the

output of *i-th* D-F/F. It also contains *n-1* taps from C_1 to C_{n-1} as shown in Figure 1. This figure shows that *i-th* bit can be generated by performing *m* previous values with XOR operations. The output of last D-F/F shall be the generated random number and it only generate single bit at a cycle. In addition, the achievable maximum period becomes 2^m-1.

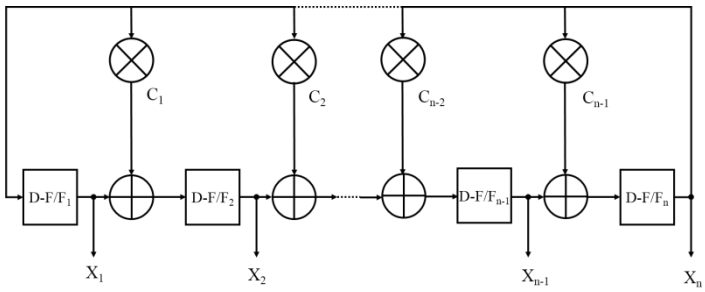

Figure 1. The Architecture of a Conventional Galois LFSR Architecture

Recently, most of applications need to use a multiple random bit at a cycle. Since the conventional LFSR based RNG is not appropriate for the multi-bits, a multi-LFSR architecture was introduced to solve this problem. As multi-LFSR architecture is employed, a set of multiple seeds is computed at the same time. The outputs for all LFSRs are to construct the multi-bits random number. This means 32 different LFSRs are required to implement the RNG having 32-bit output. Even though this architecture can provide efficient statistical properties to satisfy the security randomness for most recent complex applications, it suffers from the hardware complexity due to the LFSR duplication.

To solve the hardware complexity in multi-LFSR architecture, X. Gu and M. Zhang presented uniform RNG using the Leap-Ahead LFSR architecture [6]. Leap-Ahead LFSR can be established as follow. First, the relationship of the random bits generated within two consecutive cycle can be presented as the following recurrence equation as shown in Eq. (1).

$$X(t+1) = AX(t) \quad (1)$$

$X(t)$ is the output of all D-F/Fs in LFSR at current time and $X(t+1)$ represents the output of all D-F/Fs at the next clock cycle. A is the transform matrix. If we use m output bits from X_{n-m-1} to X_n as the generated random numbers, it can be unacceptable due to the very close correlation between the generated random numbers. The Leap-Ahead LFSR architecture exploits the transform matrix so as to avoid the close correlation between the consecutive random numbers generated. The *m*-cycle-late output can be obtained from performing the recurrence equation *m* times repeatedly. The results can be expressed as Eq. (2).

$$X(t+m) = AX(t+m-1) = A\{AX(t+m-2) = \ldots = A^mX(t) \quad (2)$$

Then, the outputs constitute a new sequence of the generated random numbers. In this case, the close correlation between the neighboring random numbers shall be decreased. If *m* operations can be performed in a cycle, Eq. (2) can be represented as Eq. (3).

$$X'(t+1) = A^mX'(t) \quad (3)$$

, where A^m is the new transform matrix to acquire the *m*-cycle-late outputs of all D-F/Fs in Figure 1 in a cycle. In addition, the output from all D-F/Fs is m-bit from X_1 to X_n, thereby it can generate the multiple bits random numbers at a cycle. These two facts are the main advantages of Leap-Ahead LFSR architecture compared to the other LFSR based RNG. However, the Leap-Ahead LFSR architecture has significant drawback associated with the

maximum period of the generated random numbers. The achievable maximum period of the conventional LFSR based RNG becomes 2^m-1 when m is the number of output of RNG. This result can be obtained when the designers carefully choose the taps of the original LFSR. However, the maximum period of the random number generated in the conventional Leap-Ahead LFSR based RNG is obtained from Eq. (4).

$$T = [2^n - 1, m] / m \quad (4)$$

,where $[2^n-1, m]$ represents the LCM (lease common multiple) of 2^n-1 and m. The maximum period, T becomes 2^n-1 when 2^n-1 is not divided into m. In the other cases, the maximum period, T becomes smaller. It becomes a great challenge to make sure that the maximum period of the random numbers generated should be closed to the ideal maximum period of 2^n-1 regardless of whether 2^n-1 is not divided into m or not. This paper presents new Leap-Ahead architecture employing segmented LFSR to enhance the maximum period when 2^n-1 is divided into m. The more detailed description of the proposed segmented Leap-Ahead architecture is introduced in the next section.

3. The Proposed Segmented Leap-Ahead LFSR Architecture

The conventional Leap-Ahead LFSR architecture is enough lightweight due to the simplicity of LFSRs and their low hardware complexity, while still providing efficient statistical property. On the other hand, it suffers from the significant decrease in the maximum period of the random numbers generated. In general, it is important that we make a balance between the maximum period and the close correlation level of the random numbers. Thus, the unintended discrete drop of the maximum period of random numbers generated is occurred when 2^n-1 is not divided into m as described in the previous section. This paper presents new segmentation technique for Leap-Ahead LFSR architecture. In this section, we describe the mathematical analysis and the deduced architecture in detail.

The conventional Leap-ahead architecture having n registers outputs m-th result as the random numbers. It can generate the multi-bits random number per cycle. 4-bit LFSR architecture as shown in Figure 1, for example, generates a sequence of 4-bits random numbers. As shown in Figure 2(a), the generated sequence composes of 15 elements. The sequence of random numbers generated is neither infinite nor truly random. Nevertheless, it is good enough to satisfy almost any statistical test of randomness. However, it suffers from the significant drawback, that is, the very close correlation between the neighboring random numbers. A Leap-Ahead architecture is a good alternative that can overcome this problem since it can output m-cycle-late output as a random number in a cycle. If we set m to 3, the elements of the generated random number sequence become {0000, 0111, 0010, 1101, 0111, 1011, 0110, 0100, 1001, 1110, 0101, 1100, 1000, 0001, 1111, 1010}. The length of the sequence of the random numbers is 15. Such a sequence is said to be pseudo-random since the generated random number will be cyclic.

The conventional Leap-Ahead architecture employing 4-bit LFSR can generate m-clock-late random number per a cycle set as shown in Figure 2. The sequence of random numbers generated is changed by m has a range from 2 to 14. For example, when m is 2, the sequence of random numbers generated is {1011, 0110, 1000, 0010, 1001, 1111, 0111, 0101}. The maximum period of the generated random numbers is 8 and the sequence becomes cyclic with period 8. In a similar way, the maximum period of the generated random numbers are 16 and 4 when m is 3 and 4, respectively. These results are different with the results obtained from Eq. (4) that presented by X. Gu [6]. The period of the random number generated by the

corresponding Leap-Ahead LFSR having following relationship with *n* and *m* as shown in Eq. (5).

$$T = [2^n, m] / m \quad (5)$$

Consequently, the maximum period of generated random numbers are different according to *m*. The period of the random number generated, T is maximum value when 2^n is not divided into *m*. Table 1 shows the maximum period of the generated random numbers associated with *n* and *m*. If the number of finite states is 16, the maximum period is occurred when *m* is 3, 5, 7, 9, 11, 13, and 15. This result suggests that the best choice for the modulus *m* is a relatively prime with 2^n. Thus, we could choose the *m* and *n* of the Leap-Ahead LFSR to make sure that the one-bit stream generated has a maximum period of 2^n.

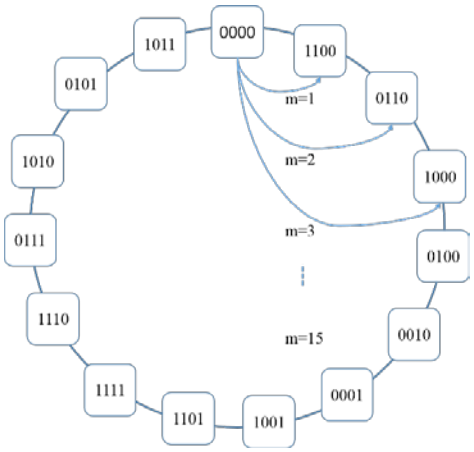

Figure 2. Basic Sequence Set of Galois

Table 1. Maximum Period of Random Number Generated According to *n* and *m*

m \ n	2	3	4	m \ n	2	3	4
2	2	4	8	9	-	-	16
3	4	8	16	10	-	-	8
4	-	2	4	11	-	-	16
5	-	8	16	12	-	-	4
6	-	4	8	13	-	-	16
7	-	8	16	14	-	-	8
8	-	-	2	15	-	-	16

We divide this architecture into two more segments that have maximum period in this paper in order to increase the maximum period of the random number generated in Leap-Ahead architecture. Obviously, the maximum period of generated random numbers for the proposed segmented Leap-Ahead architecture should be shortening than 2^n but the period is significantly longer than the period for the conventional LFSR based URNG. In this section, we describe the proposed segmented Leap-Ahead LFSR in detail.

First, we use Galois type Leap-Ahead LFSR architecture to improve the state utilization ratio. Thus, we also generate transform matrix using the same way of the previously presented by X. Gu [X.Gu]. Thus, the proposed Leap-Ahead LFSR can generate m-cycle-late random number in a cycle using $X(t+1)=A^mX(t)$ as shown in Eq. 3. The maximum period of the

generated random numbers is 2^n as shown in Figure 2. However, the proposed Leap-Ahead architecture employs the segmentation technique for LFSR architecture as shown in Figure 3. The proposed architecture divides entire LFSR as two more segments having same architecture of Leap-Ahead LFSR architecture. The number of stages in each segment is less than half of total stages in Leap-Ahead architecture.

There are two differences in terms of the control mechanism compared to the conventional Leap-Ahead LFSR architecture. First, the proposed architecture employs two more clock signals to drive the corresponding segmented LFSRs. In particular, the upper clock will be divided as the number of all possible states in the lower segmented LFSR. For example, the upper clock should be 1MHz when the lower segmented LFSR uses 16MHz and it can output 16 distinct random numbers. Second, it employs one more multiplexers to select the destination of the feedback output from the end of lower segmented LFSR. Control signal, *sel* chooses the output to each segmented Leap-Ahead LFSR. If the number of random numbers generated, 2^n is not divided by m, the number of outputs, the proposed LFSR is not divided into two parts. Thus, the output of Segment#2 is inputted into Segment#1 and the output of Segment#1 is propagated as input of Segemnt#2. The period of generated random numbers for this case is equal to that of the conventional Leaf-Ahead LFSR. However, the proposed LFSR is divided into two segments when the number of random numbers generated, 2^n is divided by m. Then, the output of Segment#2 is fed into the Segment#2, not Segment#1. In addition, the output of Segment#1 is also inputted into Segment1. Thus, both segments work independently and the period of generated random numbers.

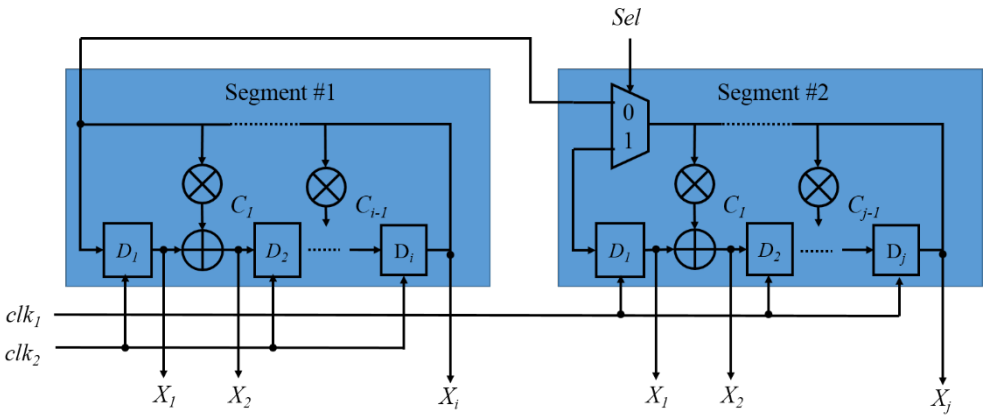

Figure 3. The Proposed Architecture of Segmented Leap-Ahead Architecture

There are four cases are produced depending the number of stages, *n* and the number of outputs, *m* are even or odd when the number of random numbers generated, 2^n is divided by *m* as shown in Figure 4. For example, the segment#1 and segment#2 operate independently when the number of random numbers generated, 2^n is divided by *m*. In case 1, both the number of stages in LFSR *n* and the number of output bits *m* are even numbers. In this case, Segment#1 contains *n/2* stages in LFSR and it outputs *m/2* output bits. In addition, Segment#2 contains *n/2* stages in LFSR and it outputs *m/2* output bits.

Case 1 : n and m are even

$$\Rightarrow \left(\frac{n}{2},\frac{m}{2}\right),\left(\frac{n}{2},\frac{m}{2}\right)$$

Case 2 : n is even, m is odd

$$\Rightarrow \left(\frac{n}{2},\left\lceil\frac{m}{2}+1\right\rceil\right),\left(\frac{n}{2},\left\lceil\frac{m}{2}\right\rceil\right)$$

Case 3 : n is odd, m is even

$$\Rightarrow \left(\left\lceil\frac{n}{2}+1\right\rceil,\frac{m}{2}\right),\left(\left\lceil\frac{n}{2}\right\rceil,\frac{m}{2}\right)$$

Case 4 : n and m are odd

$$\Rightarrow \left(\left\lceil\frac{n}{2}+1\right\rceil,\left\lceil\frac{m}{2}+1\right\rceil,\right),\left(\left\lceil\frac{n}{2}\right\rceil,\left\lceil\frac{m}{2}\right\rceil\right)$$

Consequently, maximum period of random numbers generated in the proposed segmented Leap-Ahead LFSR consists of two segmented LFSRs containing *i* D-F/Fs and *j* D-F/Fs becomes

$$T_{Max} = \frac{[2^{i/2},\frac{i}{2}]}{\frac{i}{2}} \times \frac{[2^{j/2},\frac{j}{2}]}{\frac{j}{2}}$$

The proposed segmented Leap-Ahead LFSR can increase maximum period of random numbers generated even though 2^n is divided by *m*. Furthermore, the proposed architecture does not require the significant area overhead compared to the conventional Leap-Ahead LFSR URNG. Only the circuit for controlling multiplexer and another clock input in Figure 3 is added. Thus, it can enhance the efficiency of the proposed.

4. Simulation Results

In this section, we show the simulation results of the proposed Leap-Ahead LFSR URNG compared to the other counterparts such as multi-LFSRs and Leap-Ahead LFSR URNGs. The proposed URNG is synthesized using VHDL and it is implemented on the Xilinx ISE 10.1, Virtex II PRO with a device XC2VP30. Figure 4 shows the simulation waveform of the design example having 4 stages in LFSR and outputs 4 bits random numbers.

Figure 5 shows the comparison results in respect to maximum period of random numbers generated between the conventional and the proposed Leap-Ahead URNGs. As shown in Figure 5, the maximum period of the proposed architecture is same with that of the conventional architecture when 2^n-*1* and *m* could not divide by each other. However, the maximum period of the random numbers generated is greater when 2^n-*1* and *m* could divide by each other. The difference of maximum period is up to 2.5 times compared to the conventional Leap-Ahead LFSR architecture.

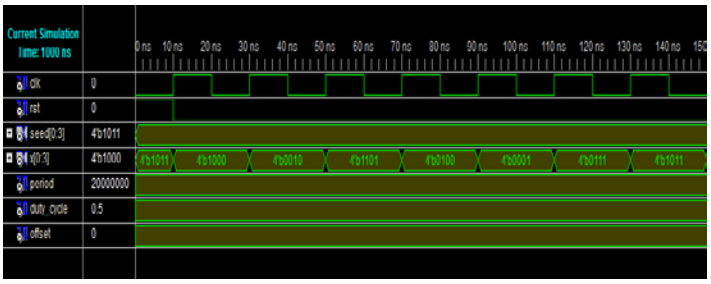

Figure 4. Simulation Results of the Proposed Segmented Leap-Ahead LFSR URNG

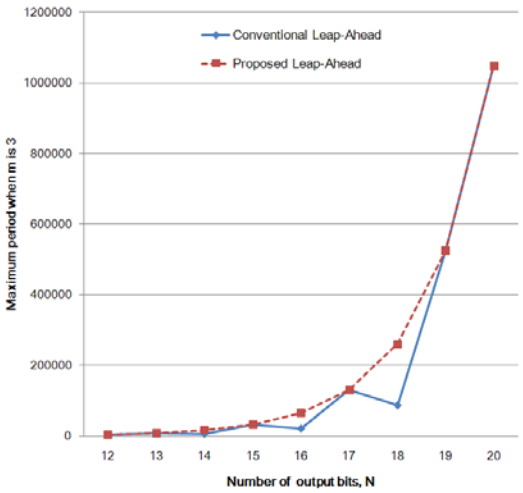

Figure 5. Comparison Results of Maximum Period between the Conventional and the Proposed Leap-Ahead LFS URNGs when the *m* is 3

Table 2 shows the simulation results when we set m to 18. This result shows that the proposed architecture requires 35% area overhead and 21% increase in clock length compared to the conventional Leap-Ahead LFSR. However, the maximum period of the generated random numbers for the conventional Leap-Ahead LFSR and the proposed segmented Leap-Ahead one are 29,127 and 261,121, respectively. Thus, the proposed segmented LFSR architecture is more effective in trade-off between the maximum period of random numbers generated and the hardware complexity.

Table 2. Simulation Results when *m* is 18

	Maximum period	Number of segmentations	H/W complexity (Slices)	Clock freq. (MHz)
Conventional Leap-Ahead LFSR	29,127	1	26	651
Proposed segmented Leap-Ahead LFSR	261,121	2	35	535

Finally, we show the comparison results with respect to the area and throughput between the proposed architecture and its counterparts in Table 3. The area time performance of the proposed segmented Leap-Ahead URNG is 2.36 slices x sec per is

2.36 and it is similar that of the conventional Leap-Ahead architecture. In addition, the throughput is also similar with that of the conventional Leap-Ahead URNG. This result shows that the proposed segmented Leap-Ahead URNG has similar area and performance with the conventional Leap-Ahead architecture even though the maximum period of the generated random number is dramatically increased. Consequently, the proposed architecture is more profitable for the data security system.

Table 3. Area and Throughput Comparison with the Other Counterparts

	Area Time slices x sec per bit $x10^{-9}$ (smaller is better)	Throughput Bits per sec $x10^{9}$ (larger is better)
Conventional Leap-Ahead LFSR	2.18	17.87
Multi-LFSR architecture	19.05	20.63
Proposed segmented Leap-Ahead	2.36	17.56

5. Conclusions

A conventional Galois LFSR random number generator is small and fast to generate random number. However, it generates only one random bit per cycle. As the use of multiple random bits at a time, Multi-LFSRs architecture was presented. However, this Multi-LFSRs architecture requires the number of LFSRs equal to the number of random bits. On the contrary, a conventional Leap-Ahead LFSR is more useful since it occupies small circuit area that is almost 10% of multi-LFSR and it has similar throughput. However, it has significant drawback that is the serious decrease in maximum period of the random numbers generated when 2^n and m could divide by each other. The proposed segmented Leap-Ahead LFSR can preserve the maximum period for all conditions. The simulation results show that the proposed architecture has similar throughput with almost same circuit size. In addition, it can increase the maximum period up to 2.5 times when 2^n and m could divide by each other. Consequently, the proposed architecture is more profitable for the data security system.

References

[1] K. Panda, P. Rajput and B. Shukla, "FPGA implementation of 8, 16 and 32 bit LFSR with maximum length feedback polynomial using VHDL", Proceedings of the 2nd Int'l Conf. on Communication Systems and Network Technologies, Rajot, India, **(2012)** May 11-13.

[2] N. M. Thamrin, G. Witjaksono, A. Nuruddin and M. S. Abdullah, "An enhanced hardware-based hybrid random number generator for cryptosystem", Proceedings of the 1st Int'l Conf. on Information Management and Engineering, Kuala Lumpur, Malaysia, **(2009)** April 3-5.

[3] P. L'Ecuyer, "Random numbers for simulation", Communications of the ACM, vol. 10, no. 33, **(1990)**.

[4] R. S. Katti and S. K. Srinivasan, "Efficient hardware implementation of a new Pseudo-random bit sequence generator", Proceedings of the IEEE International Symposium on Circuits and Systems 2009, Taipei, Taiwan, **(2009)** May 24-27.

[5] M. Goresky and A. M. Klapper, "Fibonacci and Galois representations of feedback-with-carry shift registers", IEEE Trans. on Information Theory, vol. 11, no. 48, **(2002)**.

[6] X. Gu and M. Zhang, "Uniform random number generator using Leap-Ahead LFSR architecture", Proceedings of the Int'l Conf. on Computers and Communication Security 2009, Chicago, US, **(2009)** November 9-13.

[7] J. H. Lee, M. J. Jeon, and S. C. Kim, "Uniform random number generator using Leap-Ahead LFSR architecture", Proceedings of the Int'l Conf. ASEA and DRBC 2012, Jeju, Rep. of Korea, **(2012)** November-December 28-2.

Authors

Je-Hoon Lee received the B.S. and M.S degrees in Computer and Communication Engineering from Chungbuk National University, Cheongju, Korea in 1998 and 2001, respectively. He received Ph.D. in electrical engineering from Chungbuk National University in 2005. From 2005 to 2006, he was a visiting scholar at Univ. of Southern California, USA. Also, from 2007 to 2008, he was a visiting scholar at Murdoch University, Australia. From 2006 to 2009, he was an assistant professor in Chungbuk National University. He is currently an assistant professor in Div. of Electronics, Information and Communications in Kangwon National University, Korea. His research interest is the digital circuit design for high-speed and low-power and the computer architecture.

Seong Kun Kim received the B.S. and M.S degrees in Mathematics in Pusan National University, Pusan, Korea in 1994 and 1996, respectively. He received Ph. D. in Mathematics from Oregon State University, USA in 2002. From 2003 to 2006, he was an instructor at Pusan National University. Also, he was one of BK(Brain Korea) post-doctors in Pusan National University from 2006 to 2009. He is currently an assistant professor in School of General Studies at Kangwon National University, Korea. Most of his current research interests are applications of mathematics and computer science.

