Efficient Hardware Implementation of Lightweight Pseudorandom Number Generators

Umar Mujahid, Yusra Mahmood, M.Najam-ul-Islam, Atif Raza Jafri
Department of Electrical Engineering
Bahria University
Islamabad, Pakistan
(umar.mujahid@bui.edu.pk)

Abstract— **Radio Frequency Identification (RFID) is rapidly developing technology which has greatly revolutionized our lives around the globe. In RFID scheme, the most leading and foremost issue is of security and confidentiality. An efficient and successful way to provide secure transmission is to implement authentication protocols. Pseudorandom Number Generators (PRNGs) plays an important role for security which not simply introduces randomness but also strengthens the diffusion properties of protocols. Unfortunately, the existence of PRNGs in authentication protocols increases the size of the tag which ultimately boosts up the cost of those tags. In this paper, the efficient hardware implementation of numerous Lightweight PRNGs (Linear Congruential Generator, Linear Feedback Shift Register, AKARI-1, AKARI-2, Gold-code sequence generator) has been performed. Then a novel ultra-lightweight pseudo-random number generator 'Extremely Lightweight PRNG' (EL-PRNG) using Recursive Hash has been proposed and optimally implemented. We have used XILINX Design suite for circuit synthesis and experimental results (for the targeted device virtex-6). Experimental results show that proposed EL-PRNG requires fewer resources than its contending PRNG.**

Index Terms— **Lightweight PRNGs, LCG, LFSR, AKARI-1, AKARI-2, Gold code generator, recursive hash.**

I. INTRODUCTION

RFID is an empowering innovation permitting universal remote tracking and sensing. It consists of three core entities: Transponder (tag), Reader (transceiver) and back-end database. The reader reads the information of the tag through a wireless channel and also acknowledges to the back-end database. Due to the wireless transmission between the tag and reader, the medium or the channel is vulnerable to many security attacks. Thus, the RFID systems face the critical problem of security. Many authentication protocols have been proposed previously in order to resist the security attacks.

Generally tag acknowledge with an invariable value which gives chance to antagonist to track the data. For this purpose, Random Number Generators (RNGs) are needed to introduce the randomness. The RNGs are unpredictable and have no pattern so due to inefficiency of these RNGs they substituted the Pseudorandom Number Generators (PRNGs). This is intended to produce a sequence of numbers which is usually based on a mathematical algorithm. If the algorithm and its initial states are known it can be predictable. After the addition of these PRNGs to the authentication protocols, the output goes beyond the limit of low-cost RFID tags. This is because of the increasing area of the tag which ultimately makes the cost of the tag higher. It is needed to propose and design lightweight PRNGs. In this paper, Linear Congruential Generator (LCG), Linear Feedback Shift Register (LFSR), AKARI-1, AKARI-2 and Gold Code generator have been discussed. Previously many researchers have designed the lightweight PRNGs. In [1], Zulfikar *et.al* described the circuit design of Linear Congruential Generator (LCG) and also its implementation on Field Programmable Gate Array (FPGA). In [2], Honorio Martin *et.al* projected two lightweight PRNGs that are AKARI-1 and AKARI-2. These generators can chase the requirements of low-cost RFID tags. Mohamad Merhi *et.al* analyzed the security of the PRNG which is used in the authentication protocol of the new NXP MIFARE Ultra light C in [3]. In [3], it is explained that any bad properties originate in PRNG would have to compromised the protection of the entire authentication protocol. Honorio Martin *et.al* described the efficient ASIC implementation of two lightweight authentication protocols that are Burmester-Munilla Protocol and Chien-Huang Protocol which covers the standard of EPC-C1G2 RFID tags [4]. They used low-demanding PRNGs (AKARI-1 and AKARI-2) and proposed its implementation architectures. They concluded that the PRNG consumes a significant area of the entire protocol and their proposed protocols are sufficient for low-cost RFID tags.

The rest of the paper is organized in such a way that the section II presents the Lightweight Pseudorandom Number Generators in which their algorithms have been described. The hardware schematics of all the lightweight PRNGs have been shown in section III. The statistical properties of lightweight PRNGs have also been checked through the randomness test suite (NIST, DIEHARD and ENT) and this is discussed in section IV. At the end, the results have been displayed in section V.

II. LIGHTWEIGHT PSEUDORANDOM NUMBER GENERATORS

Radio Frequency Identification (RFID) is widely deployed in billions of units yearly on everything which has made a significant impact. The major hurdle to this technology is security. Lightweight Pseudorandom Number Generators

have been introduced in authentication protocols which deal with the attacks of malicious antagonist. Below is the description of lightweight PRNGs:

1. Linear Congruential Generator (LCG):

The design of LCG was introduced by Lehmer [1]. It is represented by a sequential formula:

$$Y_{n+1} = (aY_n + c) mod m \quad (1)$$

In equation 1, Y_{n+1} is randomly generated number, a is multiplicative constant, c is additive constant and Y_n is initial seed or value and m is modulus. This involves the arithmetic operations like addition, multiplication and modulus. There are types of LCGs on the basis of multiplicative constant "c". If this constant is zero, it is called Multiplicative LCG and if constant is not equal to zero; it is known as Mixed LCG. The very first operation in LCG is multiplication of the initial seed Y_n with the multiplicative constant a. Then the result adds with the additive constant c. After that the answer is compared with modulus m. If the answer is greater than the value of m, it would be the random number and if the answer is less than the value of m, the modulus would be taken and that would be the random number.

2. Linear Feedback Shift Register (LFSR):

LFSR is another technique to generate pseudorandom numbers. LFSR is a shift register whose input bit is a linear function of the previous bit [5]. So the bit-wise exclusive OR operation is used. The output of LFSR depends on the value of input seed. As register having a finite number of states, therefore the sequence of random numbers repeats itself after a cycle. The period of LFSR is $2^n - 1$, where n represents the number of shift register. Figure 1 shows the block diagram of LFSR which is based on number of shift register.

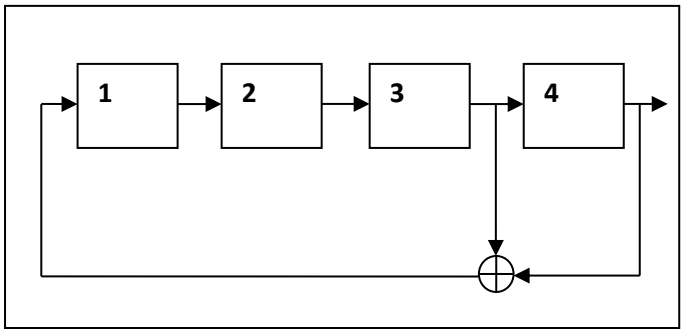

Fig. 1. Block diagram of LFSR

3. AKARI-X:

In [2] and [4], the implementations of AKARI-1 and AKARI-2 have been described. Initially, the concept of T-function was introduced. T-function is a mapping from n-bit words to n- bit words in which, for each $0 \leq k \leq n$, bit k of the output can depend only on bits $0, 1, \ldots k$ of the input. T-functions include bitwise operation like XOR, OR and AND. Unfortunately; these functions are insecure and not reliable. In order to overcome this hurdle non-linear filter functions are used in implementing the AKARI-X to introduce high degree of diffusion. In both designs of AKARI-1 and AKARI-2, the sequence of lower halves of the n output bits shows the final output.

3.1: AKARI-1

In AKARI-1, a single filter function is used which repeats itself about $k = 64$ times. Honorio Martin *et.al* has proposed two architectures for AKARI-1 in [2]. The first architecture (AKARI-1A) tries to reduce the number of clock cycles which is needed to generate an output. In order to achieve the execution of operation in only one cycle, different n bit operation blocks are used. Finite state machine implemented the control of input and output block. The aim of the second architecture (AKARI-1B) is to decrease the overall chip area by using an adder with half number of bits $(n/2)$ additionally the control implemented by the Finite State Machine. This technique requires more clock cycles as the sum takes 3 cycles. By using the adder of half bits, this engages the use of some multiplexers. So it is needed to seek a balance between adder size and number of multiplexers used. Figure 2 shows the Pseudo-code of AKARI-1.

AKARI-1

Initialize *A* and *B* of n-bits

$A = A + ((A * A) \vee 5)$

$B = B + ((B * B) \vee 13)$

$Z = A$

For k from 0 to 63

$Z = (Z \gg 1) + (Z \ll 1) + Z + B$

Output n/2 bits

Lower half of Z

Fig. 2. Pseudo-code of AKARI-1

3.2: AKARI-2

In AKARI-2, two filter functions are combined in order to reduce the number of iterations in the repeating cycle. Each function is iterated about $k = 24$ times. The three

architectures of AKARI-2 have been explored. It is seen that the first two architectures are same as the AKARI-(1A) and AKARI-(1B). The first architecture (AKARI-2A) aims to reduce the number of clock cycles and the second architecture (AKARI-2B) attempts to enhance the occupied area. There is another architecture that is AKARI-2C which is proposed to minimize even more the area with more of the number of clock cycles. In this design, the area of the adder is divided by 4. Below is the Pseudo-code of AKARI-2:

AKARI-2

Initialize *A* and *B* of n-bits

$A = A + ((A * A) \vee 5)$

$B = B + ((B * B) \vee 13)$

$Z = A \hat{} B$

For k from 0 to 24

$Z = (Z \ll 1) + ((Z + (0x56AB0A)) > 1)$

$Y = B \hat{} Z$

For k from 0 to 24

$Y = ((Y \gg 1) + (Y \ll 1) + Y + (0x72A4FB))$

$Z = Z \hat{} Y$

Output n/2 bits

Lower half of z

Fig. 3. Pseudo-code of AKARI-2

4. Gold code generator:

A Gold code is also known as Gold sequence. It is category of binary sequence which is named after Robert Gold. It is an admired and attractive set of sequences that generates a large number of unique codes. A Gold sequence is built by the XOR of two m-sequences with the same clocking.

The required Gold sequences can only be produced by preferred pairs of m-sequence [9]. A set of Gold codes possess of the sequences $\{a, a', a \oplus a', a \oplus Da', a \oplus D^2a', \dots, a \oplus D^{N-1}a'\}$, in which D is the delay element. This D shows a one-bit shift of a' relative to a. The shift register generates the Gold codes with initial state of all- ones vector set in both registers. The resulting sequences are XORed to produce one Gold Sequence. At this time, there is three set sequences. In order to gain the rest of sequences, the second of the first two sequences is shifted by one bit and executed the XOR operation again and this repeats till all the possible shifts occur and produces a new sequence in the set. For a preferred pair of *5*-bit shift registers, a new Gold sequence is produced from *0* to *30.* The period of any code in a Gold set and the m-sequence is $N = 2^n - 1$. There are a total of $N + 2$ codes in any family of Gold Codes. The Gold sequences which are generated by a preferred pair are bounded by $|R| \leq 2^{(n+1)/2} + 1$ for n odd and $|R| \leq 2^{(n+2)/2} + 1$ for n even.

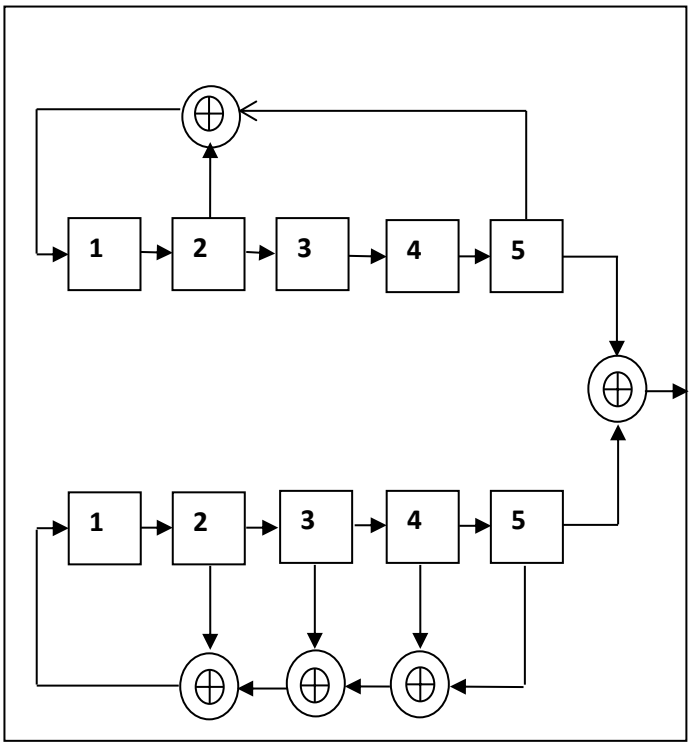

Fig. 4. Block diagram of Gold Code generator

5. EL-PRNG:

EL-PRNG is the novel lightweight PRNG which we had proposed. In order to implement EL-PRNG we used recursive hash function. In [10], Umar Mujahid *et.al* proposed a novel ultra-lightweight authentication protocol RCIA using recursive hash. It is described that only three major operations are used in RCIA protocol: bitwise AND, XOR and left rotation $Rot(A, B)$. It also consists of a new non-triangular function *Recursive Hash (R_h)*. The computation of *Recursive Hash (R_h)* is described as follows.

Consider a "*n*" bit string S,

$$S = s_1 s_2 \dots s_n, \qquad s_i \in \{0,1\}, i = 1,2, \dots, n$$

Then three steps are performed to compute $R_h(S)$:

1) Split the string S into "K" number of chunks which is same number of bits "l" per chunk.

2) Calculate a seed (index of chunk) for recursive hash by computing $R = n_1 \oplus n_2$ then the seed is equal to $wt(R)\ mod\ K$.
3) This seed will choose the chunk "K_i" from the decimated string S, then XOR operation is performed between the selected chunk "K_i" with the rest of chunks except the selected chunk. Then K_i will left rotate with itself: "$Rot(K_i, wt(K_i))$".

In this way, we can calculate the *Recursive Hash (R_h)* function. The "$Rot(A,B)$" performs the cyclic left rotation of A with respect to the number of one's in B. If there is no bit in B which is equal to one then no operation would performed. So in $Rot(A, wt(B))$, $wt(B)$ represents the hamming weight of B. We can generate a random number by using recursive hash function. The random seed will produce a random number. The algorithm is given in figure 5. In this figure, a 16-bit string S is assumed. This string is decimated into four chunks l. As the seed is 2 so the second chunk k_2 is selected for the rotation purpose. Now the XOR operation will perform between the selected chuck k_2 and all rest of other chunks except itself. This k_2 will be left rotated with respect to the number of ones in it. This procedure computes the recursive hash function *(R_h)*.

Assume $S = 1001001010110111$, $seed = 2$, $n = 16$, $l = 4$, $K = k_1, k_2, \dots, k_4$

Step1:

k_1	k_2	k_3	k_4
1001	0010	1011	0111

Step2: As $seed = 2$, so k_2 chunk will be selected for R_h.

Step3: Perform XOR between k_2 and all other chunks except itself and left rotate k_2, $Rot(k_3, wt(k_3))$

1001 0010 1011 0111

$\oplus$ 0010 0010 0010

1011 - 1001 0101

$Rot(k_3, wt(k_3))$ 0100

$R_h(S)$ 1011 0100 1001 0101

Fig. 5. Algorithm of Recursive Hash

III. HARDWARE SCHEMATICS OF LIGHTWEIGHT PRNGS

In this section, we illustrated the hardware schematics of the lightweight PRNGs which have been described above. The hardware of all lightweight PRNGs has been implemented on XILINX ISE Design suite 14.5 using the Verilog language. The sequences of PRNGs are generated according to these schematic diagrams.

1. Linear Congruential Generator (LCG):

The figure 6 shows the schematic diagram of LCG. On the left portion, there is memory which is introduced with the values of a, c, m and Y_n. There is also a sequencer which controls the access to the memory. In the middle of the diagram, we have the Arithmetic Logic Unit which performs the operations of multiplication, addition and modulus. The Arithmetic Logic Unit has two inputs. One input is from the stored values in the memory and the other is control 1 between the Y_n and the value stored in the register. There is another control- 2 which decides the operation that should be used from the Arithmetic Logic Unit (ALU). The result obtained after the execution of the equation (1) stores in register and is a random number. We have implemented 32-bit architecture of this generator.

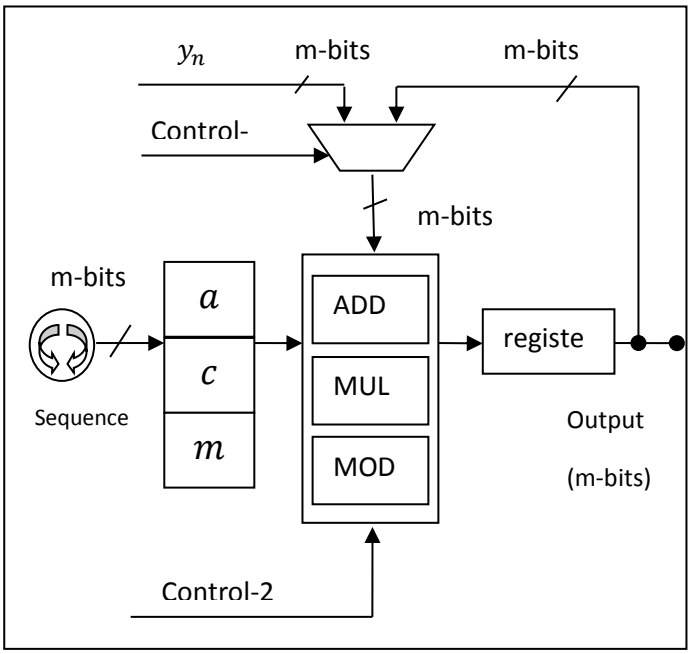

Fig. 6. Schematic diagram of LCG

2. Linear Feedback Shift Register (LFSR):

The hardware schematic of LFSR is given in figure 7. It is shown that LFSR performs two operations that are XOR and shifting. There are five registers in which values stores and shift to next register. The initial seed introduces with "CLK", the sequencer controls the access to the registers. There are two inputs which are given to the ALU. The control-1 selects between the initial seed and output value stored in reg4. There is also a clock-2 which is given so that the operations will trigger every time with it. In every clock, XOR and shift operations perform simultaneously. The sequence of clock pulses controls the shift register operation. The output is obtained by reg4 and it is a random number.

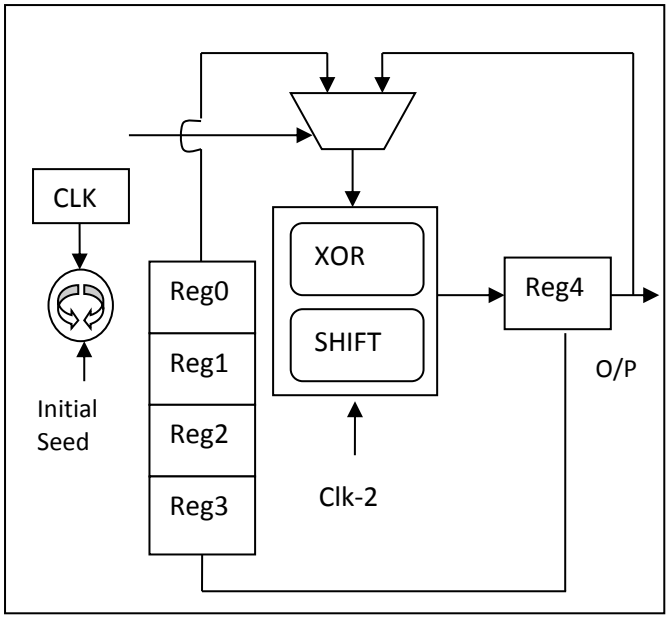

Fig. 7. Schematic diagram of LFSR

3. *AKARI-X:*

In figure 8, the general schematic diagram of AKARI-X series has been shown. In the figure, it is shown that there is a sequencer, memory, register and Arithmetic Logic Unit (ALU). Input seed introduces in sequencer and this is controlled by a clock. The ALU performs the operation of multiplication, addition, OR, XOR and left and right rotation. One input of ALU decides the operation which is to be performed. The other input is control-1 which selects between the stored value in the register and input seed from the sequencer. The output is the half of the n- bits which are the bits of initial seed. So the lower half bits of the output is taken as a random number.

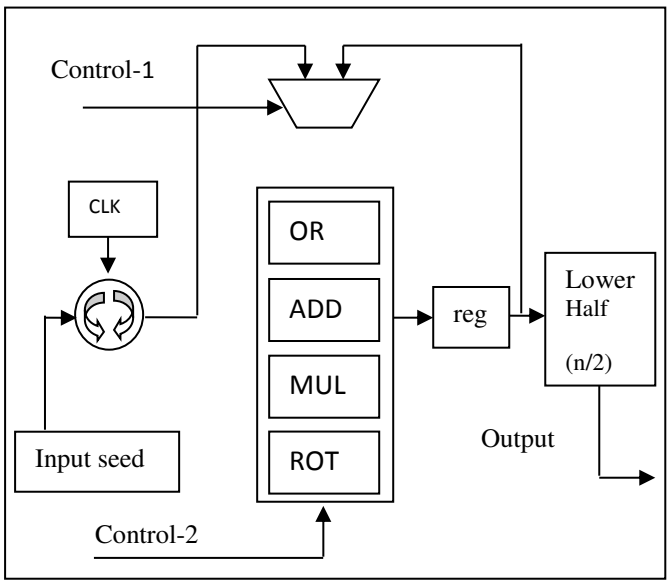

Fig. 8. Schematic diagram of AKARI-X

4. *Gold code generator:*

The basic functionality of a gold code generator is similar to LFSR. In gold code generator, the sequences of two LFSRs are XORed. Then one sequence of first LFSR performs XOR with the shifted sequence of second LFSR. In this way, the output of gold code generator is generated. The schematic of gold code can be seen in figure 9. In figure 9, two LFSR blocks are given which are the inputs of multiplexer. The control-1 selects between the two inputs. ALU consists of XOR and shift operation. Clock is given to ALU which triggers both operation turn by turn.

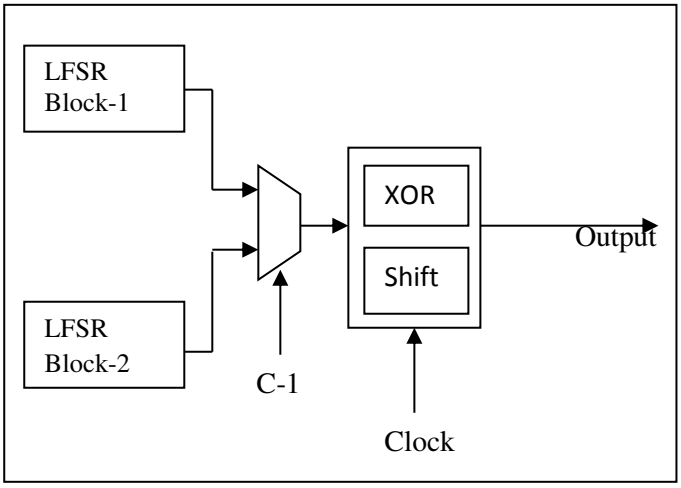

Fig. 9. Schematic diagram of gold code generator

5. *EL-PRNG:*

A novel generator EL-PRNG used recursive hash function. In recursive hash two main operations are performing: XOR and left rotation. Below in figure 10, is the schematic of EL-PRNG. A string decimated into chunks and through the calculated seed, a chunk is selected which performs XOR operation with rest of chunks. Control-1 selects between the selected chunk and rest of chunks and then the required operations would perform. Clock is given to ALU to trigger it.

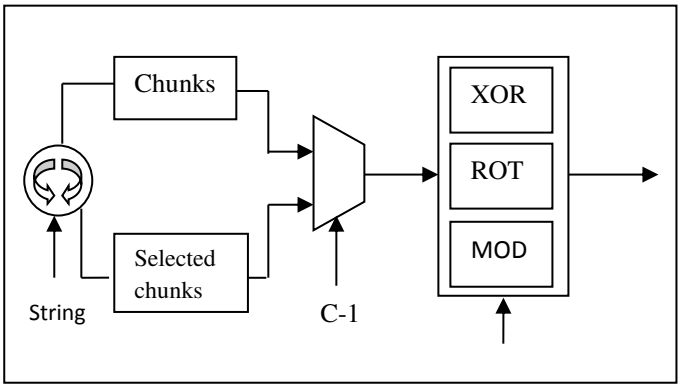

Fig. 10. Schematic diagram of EL-PRNG

IV. RANDOMNESS TEST ANALYSIS

The statistical properties of the output sequences which are generated by the lightweight PRNGs are also checked from the three batteries of statistical randomness test suites. These tests include: ENT [6], DIEHARD [7] and NIST [8]. The result of analysis is given in table 1.

4.1: ENT

The ENT test consists of six tests. ENT represents the quality of randomness in a given system. If the more entropy is fed into a value then that value will be more random.

The randomness analysis of PRNGs displays the statistical properties of generators.

V. RESULTS

Section V describes the results of the implemented lightweight PRNGs. We have designed the 32-bits architectures of PRNGs and done a detailed analysis on the synthesized circuits. After the implementation, design summary of each generator is obtained. All the PRNGs are compared on the basis of these design summaries. In table 2, the design summaries of all lightweight PRNGs are reviewed.

Table 1: Randomness analysis with ENT, NIST and DIEHARD of PRNGs

Randomness tests	LCG	LFSR	AKARI-1	AKARI-2	GOLD CODE	EL-PRNG
Entropy	7.99999	7.742279	7.99997	7.99997	7.8899	7.9988
Compression rate	0%	2.9%	0%	0%	2%	0.1%
Arithmetic mean	127.5040	124.1337	127.4929	127.5098	124.2238	127.5174
Monto carlo estimation π	3.14058	3.22583	3.14320	3.14236	3.2269	3.1424
NIST	pass	pass	pass	pass	Pass	Pass
DIEHARD	pass	pass	pass	pass	pass	Pass

4.2: DIEHARD

DIEHARD test consists of 18 test suite and each test has some P value. The value of P is different for each test. Some of it is to be measured to analyze the random behaviour for the entire sequence. Diehard test run on file of at least 80 million of bits.

4.3: NIST

The NIST batteries test consists of 15 tests, developed by random number sequence to test the random binary sequence produced by hardware for cryptographic application based random number or Pseudo random number generator.

As it is shown in table 1 that all the implemented PRNGs have passed the randomness tests, this shows that they all are random in nature. It means that they produce random sequences. Entropy, compression rate, arithmetic mean and Monto Carlo estimation π are included in the ENT test suite.

The comparison is done on the basis of logic utilization. The number of occupied slices, number of slices flip-flops, number of 4-input LUTs and number of bonded IOBs are taken as logic utilization. It can be seen that LCG used 279 numbers of slices, 43 numbers of flip-flops, 528 LUTs and 98 IOBs. In contrast to it, the number of slices, flip-flop, LUTs and IOBs which are used by LFSR is 20, 32, 32 and 66 respectively. AKARI-1 and AKARI-2 both used more amounts of devices. AKARI-1 used 261 slices, 166 flip-flops, 449 LUTs and 162 IOBs, while AKARI-2 used 641, 255, 1220 and 162 slices, flip-flop, LUTs and IOBs respectively. It is noted that the gold code generator consumes 42 slices, 64 flip-flops, 68 LUTs and 134 IOBs. The novel generator named EL-PRNG took 146 slices, 341 flip-flops, 341 LUTs and 67 IOBs. According to table 2, EL-PRNG utilized much less devices. This shows that this could be an extremely lightweight PRNG.

Table 2: Design Summary of lightweight Pseudorandom Number Generators

Logic utilization	LCG	LFSR	AKARI-1	AKARI-2	GOLD CODE	EL-PRNG
No. of slices	279	20	261	641	42	146
No. of slices flip-flop	43	32	166	255	64	341
No. of 4-input LUTs	528	32	449	1220	68	341
No. of bonded IOBs	98	66	162	162	134	67

VI. CONCLUSION

In this paper, we have introduced a novel lightweight pseudorandom number generator (EL-PRNG) using primitive recursive hash function. This proposed PRNG proves that it is extremely random in nature. In EL-PRNG, there are two main operations used: XOR and left rotation. According to the results, we have proved that EL-PRNG is the most lightweight in nature. As it is extremely lightweight, ultimately the cost of generator would also be reduced. Due to its sufficient features, EL-PRNG becomes the best choice for a very low cost RFID tags.

REFERENCES

[1] Zulfikar, Hubbul Walidainy, "Design and Implementations of Linear Congruential Generator into FPGA ", International Journal of Electronics Communication and Computer Engineering, Vol. 5, No. 4, pg (809-813), 2014.

[2] Honorio Martin, Enrique San Millan, Luis Entrena, Pedro Peris-Lopez and Julio Cesar Hernandez Castro. "AKARI-X: a Pseudorandom number generator for secure lightweight systems", 17th International On-line Testing Symposium (IOLTS), pg (228-233), 2011.

[3] Mohamad Merhi, Julio Cesar Hernandez- Castro and Pedro Peris- Lopez, "Studying the Pseudorandom Number Generator of a low-cost RFID tags", IEEE International Conference on RFID-Techonology and Applications, pp.(381-385), 2011.

[4] Honorio Martin, Enrique San Millan, Pedro Peris-Lopez and Juan E. Tapiador, "Efficient ASIC Implementation and Analysis of Two EPC-C1G2 RFID Authentication Protocols", IEEE Sensors Journal, VOL. 13, NO. 10, Pg (3537-3547), October, 2013.

[5] Purushottam Y. Chawke, R.V.Kshirsagar, "Design of 8 and 16 bit LFSR with maximum length feedback polynomial using Verilog HDL", Proceedings of 13th IRF International Conference, Pg (105-107), India, 2014.

[6] J. Walker. Randomness Battery [Online]. Available: http://www.fourmilab.ch/random/

[7] G. Marsaglia. The Marsaglia Random Number CDROM Including the Diehard Battery of Tests of Randomness [online]. Available: http://stat.fsu.edu/pub/diehard.

[8] A. Rukhin, J. Soto, J. nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Hecket, J. Dray. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, Gaitherburg, MD, USA [Online]. Available: http://csrc.nist.gov/rng/

[9] William Stallings, Wireless Communications, 2nd edition, 2002.

[10] Umar Mujahid, M. Najam-ul-Islam, M. Ali Shami, "RCIA: A New Ultralightweight RFID Authentication Protocol Using Recursive Hash", International Journal of Distributed Sensor Networks, Volume 2015.

[11] Umar Mujahid, M. Najam-ul-Islam, Qurat-ul-Ain, Yusra Mehmood, "A novel Lightweight Pseudorandom Number Generator for Passive RFID systems", 17th IEEE International Multi Topic Conference (INMIC), 2014.

[12] Umar Mujahid, M. Najam-ul-Islam "Ultralightweight Cryptography for Passive RFID systems", International Journal of Communication Networks and Information Security, Vol.6. no.3, 2014.

[13] Umar Mujahid, M. Najam-ul-islam. et al. ,"A novel pseudorandom number generator for passive RFID systems", 17th IEEE-International multi topic conference (INMIC-2014), Karachi Pakistan.