Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2020) 28: 288 – 301

doi:10.3906/elk-1906-194

A novel S-box-based postprocessing method for true random number generation

Erdinç AVAROĞLU[1,*], **Taner TUNCER**[2]

[1]Department of Computer Engineering, Faculty of Engineering, Mersin University, Mersin, Turkey
[2]Department of Computer Engineering, Faculty of Engineering, Fırat University, Elazığ, Turkey

Received: 29.06.2019 • **Accepted/Published Online:** 04.10.2019 • **Final Version:** 27.01.2020

Abstract: The quality of randomness in numbers generated by true random number generators (TRNGs) depends on the source of entropy. However, in TRNGs, sources of entropy are affected by environmental changes and this creates a correlation between the generated bit sequences. Postprocessing is required to remove the problem created by this correlation in TRNGs. In this study, an S-box-based postprocessing structure is proposed as an alternative to the postprocessing structures seen in the published literature. A ring oscillator (RO)-based TRNG is used to demonstrate the use of an S-box for postprocessing and the removal of correlations between number sequences. The statistical properties of the numbers generated through postprocessing are obtained according to the entropy, autocorrelation, statistical complexity measure, and the NIST 800.22 test suite. According to the results, the postprocessing successfully removed the correlation. Moreover, the data rate of the bit sequence generated by the proposed postprocessing is reduced to 2/3 of its original value at the output.

Key words: Ring oscillator, random number, postprocessing, S-box, statistical test

1. Introduction

Randomness is the absence of a specific pattern or predictability in phenomena. Systems that are not random are deterministic. Random numbers are defined as numbers that have no relation to each other, whose occurrence probabilities are equal, and that are defined over a specific interval. Random numbers are used in many areas, such as simulation, sampling, numerical analysis, entertainment, and cryptography. Random numbers are especially important in cryptographic systems. In cryptography, random numbers are needed for key generation and distribution, for the generation of starting vectors, for identity verification protocols, for prime numbers, and for password generation. The security of a cryptographic system relies on the true randomness of the numbers obtained. For this reason, random number generators form the most critical aspect of cryptography [1]. Random number generators are divided into 2 classes, pseudorandom number generators (PRNGs) and true random number generators (TRNGs). PRNGs are algorithms that create number sequences with characteristics similar to random numbers. These number sequences are calculated deterministically using a initial value called the seed. No PRNG generates true random numbers because all PRNGs use a deterministic algorithm. For this reason, PRNGs are not suitable for cryptography applications. In TRNGs, a nondeterministic source of entropy is used to generate random numbers. In TRNG designs, sampling is achieved by digitizing the signals obtained from a source of entropy. After the sampling process, generated bit sequences are postprocessed in order to make the bit sequences independent of each other and remove any correlation between them. As shown

*Correspondence: eavaroglu@mersin.edu.tr

in Figure 1, a TRNG is formed by 3 basic units. These are, in order, the source of entropy, the sampler, and the postprocessing [1, 2].

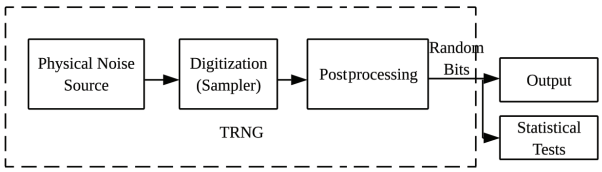

Figure 1. Basic units of TRNGs [3].

The true randomness of TRNGs is completely dependent on the source of entropy. If the source of entropy is of high quality, numbers generated by the TRNG have good statistical properties. Human-driven interaction and physical processes such as noise based on thermal, shot, and atmospheric sources, and metastability-containing circuits, jitter, and Brownian motion are used as sources of entropy in the published literature [1]. In the sampling unit, random signals generated by the source of entropy are transformed into digital signals. For the sampling unit, periodic sampling, mixing, or aperiodic sampling circuits are used in the published literature [1]. Postprocessing is usually used to remove correlation to the source of entropy and increase the randomness of the signal. Another advantage of postprocessing is that it makes the system more robust against environmental changes and side-channel attacks. The security of the generator is increased by removing correlation due to the postprocessing algorithm. However, postprocessing decreases the data rate of generated numbers as a result of removing correlation in the bit sequences. Postprocessing is used to remove correlation in the source of entropy in TRNG systems. Most of the postprocessing algorithms proposed in the published literature reduce the data rate of the TRNG. For example, the data rate of an XOR corrector, Von Neumann corrector, H function, and resilient function are reduced by a 1/2 [1], a 1/4 (approximate) [4], a 1/2 [5], and by 1/16, respectively. In the present study, a novel postprocessing algorithm is proposed that removes the statistical weaknesses and reduces the data rate of a TRNG to 2/3 of the original. The proposed postprocessing is performed by substitution boxes (S-boxes), which are used in the safe design of block encryption systems. S-boxes are nonlinear bijective (one-to-one and surjective) transformations that change an n-bit input value to an m-bit output value. In the present study, an RO-based TRNG system is defined and S-box postprocessing is used to remove correlation between the sequences of numbers generated by this number generator. The RO-based TRNG uses the Sunar structure [6]. Statistical tests on the numbers are conducted using the NIST 800.22 test suite. The rest of this study is organized as follows: section 2 describes related work about postprocessing in the published literature. The proposed postprocessing algorithms are explained in section 3. Section 4 outlines the proposed use of an S-box for postprocessing. In section 5, statistical test results, scale index, and autocorrelation results are produced by the NIST 800.22 test suite in order to demonstrate the usability of random numbers obtained in real time for cryptographic systems. In the conclusion section, interpretation of the statistical tests of the proposed system is discussed by mentioning the advantages and disadvantages of the proposed system.

2. Related work

Various postprocessing algorithms have been proposed to mitigate the statistical shortcomings of numbers generated by TRNG systems. Some of these are the XOR corrector Von Neumann corrector [4], H function [5], SHA-1[2], and resilient functions [6]. The SHA-1 postprocessing algorithm, together with Von Neumann, was

[1]Davies RB (2002) Exclusive OR (XOR) and hardware random number generators [online]. Website http://www.robertnz.net/pdf/xor2.pdf [accessed 14 April 2019]

[2]Kocher PB (1999) The intel random number generator [online]. Website https://www.rambus.com/wp-content/uploads/2015/08/IntelRNG.pdf [accessed 30 April 2019]

used in an Intel TRNG, one of the very first TRNG designs. A thermal noise source was used as the source of entropy in the Intel TRNG. The statistical properties of numbers obtained by sampling from the system were not good. For this reason, the randomness properties of these numbers were improved through processing the numbers first with Von Neumann postprocessing and then the SHA-1 algorithm. Numbers obtained from this system successfully passed the FIPS 140-1 test. In another study, a resilient function was used for postprocessing in an RO-based TRNG system proposed by Sunar [6]. In this TRNG design 114 ring oscillators (ROs), each having 13 inverters, were used as the source of entropy. Using too many RO circuits in the system caused various problems. These problems included high energy consumption, a reduction in the quality of randomness, and correlation in the output because the ROs were not independent of each other. Therefore, the obtained bit sequence was postprocessed using the resilient function. The data rate of the TRNG was reduced to 1/16 due to the resilient postprocessing used in the system. There have been RO-based TRNG designs, used as a source of entropy, and various postprocessing methods have been used in these designs. In the TRNG system developed by [7], an oscillator ring with two transparent latches, a buffer, and an inverter was used. The output of the TRNG was postprocessed using an XOR function. For this reason, the data rate was low, and the obtained rate was less than 1 Mbit/s. In [8], a TRNG that used a Galois ring oscillator (GRO) and Fibonacci ring oscillator (FRO) was proposed. Outputs obtained from the FRO and GRO were unified using XOR, and random numbers were generated by sampling the XOR output with a D-type flip-flop. The output bit sequence was postprocessed by linear feedback shift register (LFSR)-based postprocessing. A bit rate of 12.5 Mbit/s was achieved with this design. In [9], the design described in [6] was realized on a Xilinx Virtex II Pro FPGA (field programmable data array) using an RO and the inverter numbers (110, 3), (110, 13), and (210, 3). The strongest source of entropy was identified as (210, 3) during tests. The outputs obtained were postprocessed using a resilient function and the final output rate was 2 Mbps. In [10], the logistics map is used to improve the statistical properties of numbers that are obtained using an RO-based TRNG. There is no bit rate reduction in this postprocessing. Furthermore, successful results were obtained in the NIST tests with the resulting chaotic map. However, the biggest disadvantage of this system is the high energy consumption and low data rate. In [11], performance differences between the conventional TRNG method that used a chaotic system and recently designed FPGA-based chaotic systems have been compared. In [12], the authors proposed a novel type of high-speed TRNG based on chaos and ANN implemented in an FPGA chip. The generated random numbers have been tested with the NIST 800.22 and FIPS 140-1 test suites. The high quality of generated numbers has been confirmed by passing all randomness tests. In [13], the authors proposed a new dual entropy core discrete time chaos-based TRNG structure that has been implemented on an FPGA. The system has been synthesized in an FPGA chip. The data rates of the designed dual entropy core TRNG units range between 390 and 464 Mbps. In [14], the Sundarapandian–Pehlivan chaotic system was modeled and simulated to generate random numbers. The frequency of the proposed system is 293.815 MHz and the system can calculate 1,000,000 data in 0.201 s. The proposed system is successful according to the FIPS-140-1 and NIST-800-22 test suites. In [15], a true random number generator is formed by uniformly sampled ring oscillators and using the Hash function for postprocessing. Both the generator and SHA-256 are realized on a 5-core Virtex FPGA. As a result of the experiments, it was shown that at least 8 ring oscillators are required to make the system pass all statistical tests. In [16], true random bit generation is achieved by using the random environmental noise signals coming from the microphone port on a computer sound card. As a postprocessing method, a novel procedure for the distribution of bits, "Mixing Bits in Steps and XORing of Adjacent Bits" (MiBiS&XOR), was proposed. The proposed procedure divides input bits that are consecutive and specifically correlated by separating them in a

simple and efficient manner. This procedure decreases the cumulative autocorrelation. Experimental statistical randomness tests performed on the bit sequences obtained by this procedure validated the perfect quality of TRNG outputs. In [17], an improvement by applying the N-bit Von Neumann (VN_N) postprocessing technique is shown. A general algorithm that explains N-bit VN_N and the application of 4-bit VN_4 at the circuit level are shown. With VN_4, an output rate of 40.6% was achieved. A new waiting strategy was proposed to further improve the output rate. VN_4+ waiting and VN_8+ waiting achieved 46.9% and 62.5% rates, respectively. When compared with the original Von Neumann (VN_2) technique, they gave results improved by 1.88 and 2.50 times, respectively.

3. Postprocessing

Postprocessing is usually used to increase the randomness in the signal. Postprocessed signal values are uniform compared to the original signal. The second purpose of the postprocessing, which has gained much importance as a countermeasure to combat side-channel attacks, is to make number generators more robust against tampering and environmental changes. The security of the generator will increase depending on the postprocessing algorithm. In the published literature, various postprocessing algorithms such as XOR corrector, Von Neumann corrector, extractor function, H function, and resilient functions are used [4–6]. These postprocessing methods are described below.

3.1. XOR corrector

In the XOR corrector shown in Figure 2. The bit sequence obtained is separated into n-bit ($n = 2$) blocks in order to produce an output bit. Each separated block is processed on its own with the XOR function. This procedure causes the output bit efficiency to decrease $1/n$ times while it removes bias in the output bit. However, bias in the output bit sequence will only decrease on the condition that the input bits are independent. The advantage of this corrector is that it is simple and can achieve a constant output bit rate[1].

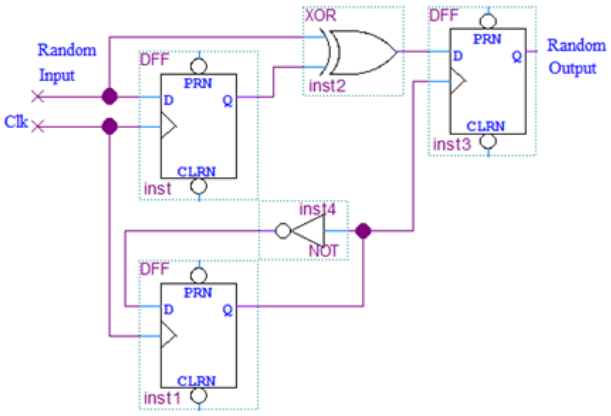

Figure 2. XOR postprocessing ($n = 2$).

[1]Davies RB (2002) Exclusive OR (XOR) and hardware random number generators [online]. Website http://www.robertnz.net/pdf/xor2.pdf [accessed 14 April 2019]

3.2. Von Neumann corrector

This is the oldest and simplest method to remove irregularities in the bit sequence. As shown in Table 1, Von Neumann generates uniform 0 and 1 bits. It takes into consideration the synchronous pairs coming from the TRNG. If the bit sequence is (1, 0), it produces a 1. If the bit sequence is (0, 1), it produces a 0. Bit sequences (0, 0) and (1, 1) are discarded. The change in the bit rate is shown in Table 2. This corrector makes the entropy close to the ideal value of 1 and hence contributes to its improvement. However, since some bit sequences coming from the TRNG are discarded, the Von Neumann output bit rate relies on the TRNG output and is therefore not constant. The bit rate is reduced to 1/4 of the input bit rate [4].

Table 1. Von Neumann corrector.

Input Bit Pairs	Von Neumann Output
00	No output
01	0
10	1
11	No output

Table 2. Change in the bit rate due to Von Neumann.

0	1	0	0	1	1	1	0	1	0	0	0	1	1	0	1	1	0	0	1	0	0	1	0	1	1	Pure Bits from TRNG
0		-		-		1		1		-		-		0		1		0		0		1		-		Von Neumann Output

3.3. H function

The H postprocessing function shown in Figure 3 is proposed by Dichtl to prevent bias in a TRNG [5]. It is based on the quasigroup sequence transformation. The postprocessing algorithm uses 16 bits of the source randomness to obtain an 8-bit output. The input bits for the postprocessing are $a_0, a_1, ..., a_{15}$. The 8-bit $b_0, b_1, \ldots, b_7$ is defined as $b_i = a_i \oplus a_{i+1} mod 8$. There are only 128 possible values for $b_0, b_1, \ldots, b_7$. When the input is a perfect random number generator, one bit of the entropy is removed. The output of the postprocessing function $c_0, c_1, \ldots, c_7$ is defined as $c_i = b_i \oplus a_{i+8}$. This function is called the H function. The 16-bit sequence in the input is divided into two: $A_1 = a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7$ and $A_2 = a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15}$. The A_1 byte is shifted left 1 bit by $rotateleft(A_1, 1)$.

$$b_i = a_i \oplus a_{i+1} mod 8 = A_1 \oplus rotateleft(A_1, 1) \tag{1}$$

$$c_i = H(A_1, A_2) = A_1 \oplus RL(A_1, 1) \oplus A_2 \tag{2}$$

3.4. Extractor function

The extractor function was proposed by Barak et al. [18] to make the TRNG more robust against changing environmental conditions. This is a very strong function with measurable properties. This strong stateless function with measurable properties was first developed as a tool for complexity theory. In [18], a mathematical model was proposed to capture the impact of an attacker on the randomness source. This provides an open structure based on the correlation of the summary functions proven for the output properties with the input

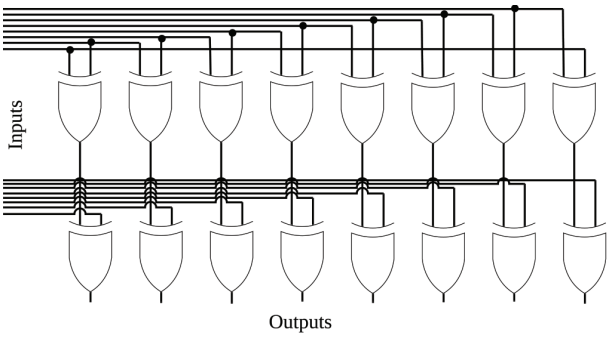

Figure 3. H postprocessing function [17].

through unknown causes, should such a thing exist. The definition of an extractor is provided below [18]: An extractor is a function $E : \{0,1\}^n \times S \to \{0,1\}^m$ for some data set S. It is denoted by $E^\pi(\cdot) = E(\cdot, \pi)$, a single-input function that is the result of fixing the parameter π to the extractor E. It is desirable for the output $E(X, \pi) = E^\pi(X)$ to be (close to) uniformly distributed, where $X \in \{0,1\}^n$ is the output of the high-entropy source and $\pi \in S$ is the public attribute. Defining security, we consider the following ideal setting:

1. An adversary chooses 2^t distributions $D_1, \ldots, D_{2^t}$ over $\{0,1\}^n$, such that $min - Ent(D_i) > k$ for all $i = 1, \ldots, 2^t$.
2. A public attribute π is chosen at random and is independent of the choices for D_i.
3. The adversary is given π and selects $i \in \{1, \ldots 2^t\}$.
4. The user computes $E^\pi(X)$, where X is drawn from D_i.

3.5. Resilient function

A resilient function involves the filtering of deterministic bits. In a study by Sunar et al., the affected deterministic bits were used to investigate the tolerance of the properties of the resilient function [6]. The high degree of flexibility and the expected number of deterministic bits in the sample determines the amount of tolerance of the TRNG of an active enemy. The $(n, m, t)-$resilient function is defined as follows:

Definition 1 *An (n, m, t)-resilient function is a function of the type $F(x_1, x_2, ..., x_n) = (y_1, y_2, ..., y_m)$. For Z_2^n to Z_2^m it has the property that, for any t coordinates $i_1, ..., i_t$, for any constants $a_1, ..., a_t$ from Z_2, and any element y of the codomain: Prob $[F(x) = y | x_{i1} = a_1, ..., x_{it} = a_t] = 1/2^m$. In computing this probability, all x_i for $i \notin \{i_1, ..., i_t\}$ are viewed as independent random variables, each of which takes on the value 0 or 1 with a probability of 0.5.*

4. Proposed postprocessing

The name S-box comes from the English word substitution. S-box usually appears in methods that use one symmetric encryption variant called block encryption. The goal is to determine the topology of a table using the table. S-box is the only nonlinear component in block encryption systems such as DES and AES and provides the property of confusion. That is why, to be able to design strong encryption systems, S-boxes with good

cryptological properties must be developed[3]. Regardless of the S-box type or category created, an S-box must display specific attributes in order to be effective. It is not possible to design a good S-box by combining any arbitrary substitution schemes. One way to generate these inputs in S-boxes is to create a nonlinear Boolean function that maps the n-bit input to an m-bit output. A special Boolean function set, called bent functions, can be used to achieve maximum nonlinearity[4]. In an $n \times m$ S-box, S is a mapping where $S : \{0,1\}^n \rightarrow \{0,1\}^m$. S can be represented as 2^n $m-$bit numbers, denoted $r_0, , r_{2^n-1}$, in which case $S(x) = r_x, 0 \leq x \leq 2^n$ and r_i are the rows of the S-box. Alternatively, $S(x) = [c_{m-1}(x), c_{m-2}(x), \ldots, c_0(x)]$, where c_i are fixed Boolean functions $c_i : \{0,1\}^n \rightarrow \{0,1\}$ $\forall i$; these are the columns of the S-box. Finally, S can be represented by a $2^n \times m$ binary matrix M, where the i, j entry is bit j of row i. A linear combination of two functions $f, g : \{0,1\}^n \rightarrow \{0,1\}$ is defined to be: $(f \oplus g)(x) = f(x) \oplus g(x)$, where $\oplus$ denotes $modulo2$ addition. Let V_n denote the set of functions mapping $\{0,1\}^n \rightarrow \{0,1\}$. Let Ln denote the set of linear functions mapping $\{0,1\}^n \rightarrow \{0,1\}$. Let An denote the set of affine functions mapping $\{0,1\}^n \rightarrow \{0,1\}$[5]. There are other criteria that must be met in the design of an S-box. These criteria are confusion and diffusion. Generally, the design criteria below must be met for a good S-box. **A. Strict avalanche criterion** The strict avalanche criterion was first published by Webster and Tavares [19]. A function satisfying the strict avalanche criterion means that changing a single input bit has a probability of changing half of the output bits. **B. Bit independence criterion (BIC)** The bit independence criterion requires the output bits to move independently. In other words, a statistical model or dependency between output vectors and output bits must not exist [19] For a function $f : \{0,1\}^n \rightarrow \{0,1\}^n$, where $i, j, k \in \{1, 2, ..., n\}$ and $j \neq k$, for all i, j, k values, when the input bit i is inverted, if the output bits j and k can be changed independently, BIC is satisfied [19]. **C. Nonlinearity criterion** An S-box is not a linear mapping from input to output. If it was, that would make the cryptosystem more vulnerable to attacks. When an S-box is formed by Boolean functions with maximum nonlinearity, it will be incompatible with linear functions and therefore it becomes harder to break the cryptosystem. **D. Balance** This means that every Boolean vector responsible for the S-box contains the same amount of 0s and 1s. It is not possible to satisfy all these criteria simultaneously. A compromise must be made for some of the criteria. For example, the bit independence criterion conflicts with the nonlinearity criterion, while the nonlinearity criterion conflicts with the balance criterion at the same time. The S-box that is designed for DES is listed in Table 3 . The S-box designed for DES uses a 6-bit input and a 4-bit output. As shown in Figure 4, the most and the least significant bits of the 6-bit input determine the rows, and the other bits determine the columns. The integer at the chosen row and column is used as the output. In the present study, the binary equivalent of this integer is used as the random number generated [20]. For example, suppose that the bit sequence 011000010001011110... is produced by the number generator. The first 6 bits of this sequence are 011000 and let the S-box that is going to be used for S1 be like Figure 4. The most and the least significant bits, 00, will be used for determining the row, and the other bits, 1100, will be used for determining the column. As shown in Table 4, the determined output for the row and column chosen according to the S-box S1 is the integer 5. The binary representation of this number is 0101. As a result, the number 0101 will be produced by the generator using S-box postprocessing for the 6-bit number 011000. An example bit sequence generated via S-box and the pseudocode is given below.

[3]Özkaynak F, Özer AB. Rasgele seçim tabanlı s-box yapıları. Cryptodays [online]. Website http://mcs.bilgem.tubitak.gov.tr/cryptodays/files/posters/Fatih%20Ozkaynak_Ozer.pdf [accessed 25 April 2019]

[4]Modern Cryptography: Applied Mathematics for Encryption and Information Security [Online]. Website https://www.oreilly.com/library/view/modern-cryptography-applied/9781259588099/. [accessed 02 May 2019].

[5]Practical S-Box Design [Online]. Website https://www.researchgate.net/publication/2584910_Practical_SBox_Design. [accessed 02 May 2019].

Table 3. S-boxes used for DES [20]

S1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S2															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S3															
13	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S4															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S5															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S6															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S7															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S8															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	7

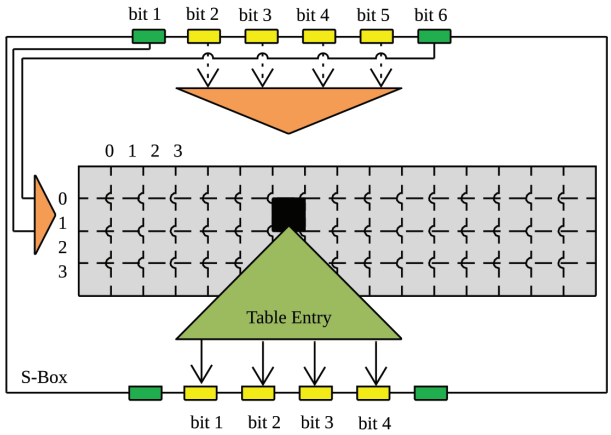

Figure 4. S-box rule [21].

Pseudocode:

Step1. 48 bits are taken from the generated random bit sequence

011000 010001 011110 111010 100001 100110 010100 100111

Step2. The 48-bit block is divided into 8 separate parts, each having 6 bits (S1, S2, S3, S4, S5, S6, S7, S8)

$$\underbrace{011000}_{S1}\ \underbrace{010001}_{S2}\ \underbrace{011110}_{S3}\ \underbrace{111010}_{S4}\ \underbrace{100001}_{S5}\ \underbrace{100110}_{S6}\ \underbrace{010100}_{S7}\ \underbrace{100111}_{S8}$$

Step3. The most and the least significant (the first and the last) bits of S1 are taken and used to determine the row.

Step4. The remaining bits determine the column.

Step5. The new value is taken by looking up the specified row and column of the S-box.

S_1																
	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
01	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
10	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
11	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Step6. The new bit sequence is obtained by writing the new value in a binary representation.

Step7. Go to Step 2 unless S8 has been reached.

Step8. A 32-bit block is generated by S1, S2, S3, S4, S5, S6, S7, and S8.

$$\underbrace{011000}_{S1}\ \underbrace{010001}_{S2}\ \underbrace{011110}_{S3}\ \underbrace{111010}_{S4}\ \underbrace{100001}_{S5}\ \underbrace{100110}_{S6}\ \underbrace{010100}_{S7}\ \underbrace{100111}_{S8}$$

$$0101\quad 1100\quad 1000\quad 0010\quad 1011\quad 0101\quad 1001\quad 0111$$

Step9. The End

In the present study, in order to demonstrate the use of S-boxes as a postprocessing method, random bit sequences obtained from the RO-based TRNG system were proposed as a source of entropy by Sunar. In the published literature, Sunar performed random number generation and obtained successful results with a TRNG [10, 22]. In [10], bit sequences obtained without postprocessing are used. According to the results obtained from [10], an RO-based TRNG does not pass the NIST test without postprocessing.

The TRNG system shown in Figure 4 has 114 ROs and 13 inverters in each RO. Random signals obtained from the RO are unified by the XOR function. After the XOR function, the signals obtained are sampled using a D type flip-flop. Sunar used the resilient function in postprocessing to mitigate the statistical weaknesses occurring in this design [6]. In the present study, instead of the resilient function, the S-box that is proposed for postprocessing is shown in Figure 5.

As shown in Figure 5, an FPGA application is realized in order to apply S-box postprocessing to bits obtained from the RO-based TRNG. Figure 6 shows the hardware used for the S1 box. The Rom data module shows the memory unit that stores the raw bits obtained from the RO-based random number generator. The most and the least significant bits of the 6-bit number that will be read from the first address are given to the Dec decoder module as input. The value 00 ensures the $0th$ row, 01 the $1st$ row, 10 the $2nd$ row, and finally 11 the $3rd$ row of the S1 box is chosen, each of these values being input to this module. The other 4 bits choose the column for the chosen row. Memory modules $Sbox11$, $Sbox12$, $Sbox13$, and $Sbox14$ are used. The outputs of these 4 memory units are the random numbers generated and they are recorded into a Ram module with the help of a Mux in order to perform statistical tests. C_data and Dec1 modules are only used to perform type conversions. For example, in the C_data module, the 6-bit vector is converted to a 4-bit vector and 2×1 bit output.

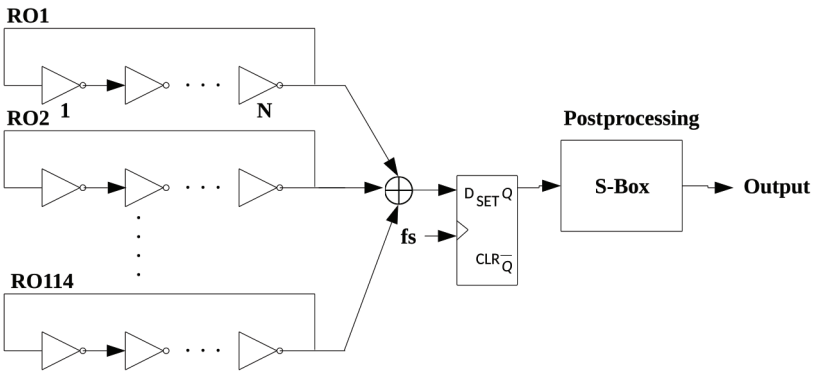

Figure 5. The postprocessing model based on an S-Box (ring oscillator)

5. Results obtained from S-box-based postprocessing

Bit sequences generated by random number generators may show statistical weaknesses. The generated bit sequence is therefore subject to postprocessing in order to remove these weaknesses. Postprocessing applications cause a reduction in the bit rate while removing these weaknesses. Table 4 provides a comparison between the proposed S-box-based postprocessing and other postprocessing methods in the published literature.

The proposed S-box based postprocessing algorithm in the present study reduces the data rate of the TRNG to 2/3 of the original, and so using an S-box causes less data rate reduction than other postprocessing methods. In order to use the developed system in cryptographic applications in a secure manner, randomness

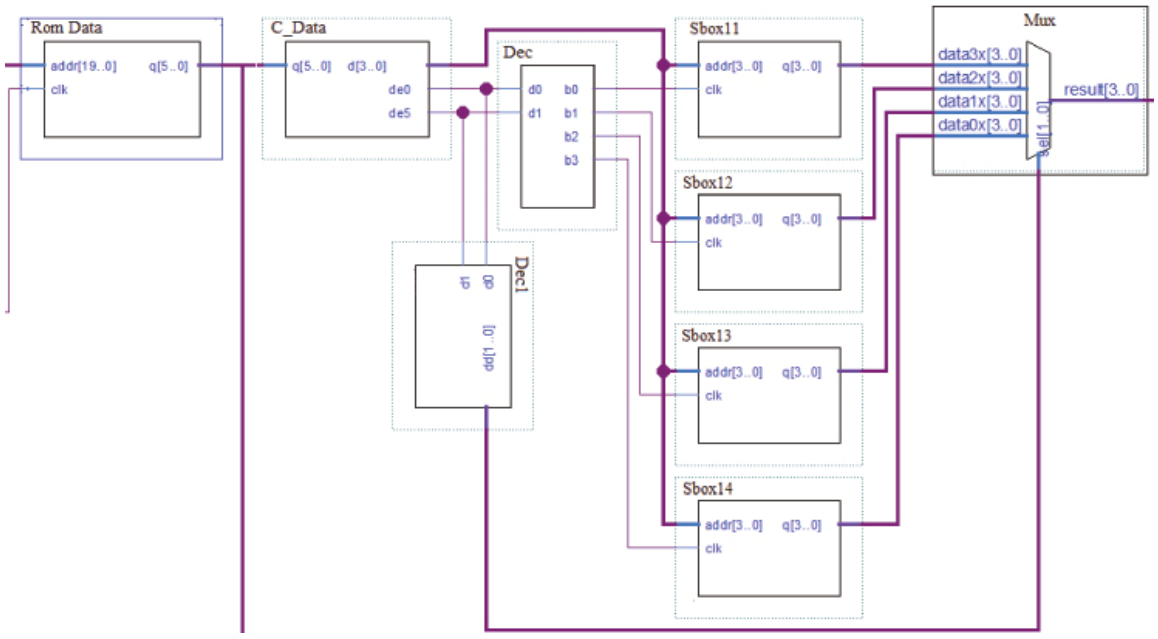

Figure 6. Realization of the S-box S1 on an FPGA.

Table 4. Bit reduction rates for the postprocessing methods.

Postprocessing	Bit rate
Van Neumann	About 1/4
XOR	1/2
H function	1/2
Resilient/Extractor	1/16
Proposed S-box postprocessing	2/3

tests must be applied to it. Many test suites have been developed to analyze the randomness of generated numbers. One of these tests is the statistical NIST 800.22 test suite. In this test suite, there are 16 tests in total and the parameters of each test are explained in detail [23]. According to this test suite, statistical tests on the generated numbers by the system are performed by the developed software [24]. The statistical test results performed by the NIST 800.22 test suite are given in Table 5 and Table 6 below. Table 5 shows the statistical test results of the S-box-based postprocessed bit sequence obtained from the Sunar TRNG system. Successful results are achieved by postprocessing the source of entropy. Table 6 shows a comparison between the proposed S-box postprocessing method and the other postprocessing algorithms. While the entropy of the pure bits generated by the RO-based TRNG is 0.989, with the proposed S-box postprocessing it is increased to 1.0. Entropy values of both the S-box and the other postprocessing algorithms are given in Table 7. These results, together with the NIST test results, show that S-box postprocessing can be used in TRNG systems because it can produce bit sequences with no statistical correlation between the sequences. Autocorrelation test results are given in Table 8. Correlation shows the linear relationship between two or more variables. It takes a value between $+1$ and -1. If it is 0 or close to 0, there is no linear relationship between these variables. As shown in Table 8, for values 8 and 15 for D, successful results are achieved by the S-box postprocessing [25]. In Table 9,

the statistical complexity measure is shown. For aperiodic sequences, the statistical complexity measure must be zero or close to zero. Table 9 shows that successful results are achieved for the S-box method [25].

Table 5. Statistical test results of the enhanced Sunar system with and without postprocessing.

	Sunar Design without postprocessing		Sunar Design with S-box postprocessing	
Test	P-Value	Result	P-Value	Result
Frequency (Monobit) Test	-	Not passed	0.639	Passed
Frequency Test within a Block	-	Not passed	0.108	Passed
Runs Test	-	Not passed	0.177	Passed
Test for the Longest Run of Ones in a Block	-	Not passed	0.123	Passed
Binary Matrix Rank Test	0.424	Passed	0.564	Passed
Discrete Fourier Transform Test	-	Not passed	0.147	Passed
Nonoverlapping Template Matching Test	-	Not passed	0.114	Passed
Overlapping Template Matching Test	-	Not passed	0.413	Passed
Maurer's Universal Statistical Test	-	Not passed	0.552	Passed
Linear Complexity Test	0.615	Passed	0.265	Passed
Serial Test	-	Not passed	0.430	Passed
	0.879	Passed	0.709	Passed
Approximate Entropy Test	-	Not passed	0.137	Passed
Cumulative Sums Test	-	Not passed	0.318	Passed

Table 6. Statistical test results of the Sunar system processed by the other postprocessing methods.

Test	Sunar Design without postprocessing	Von Neumann	XOR	H Function
Frequency (Monobit) Test	-	0.365	-	0.241
Frequency Test within a Block	-	0.596	-	0.924
Runs Test	-	0.083	-	0.302
Test for the Longest Run of Ones in a Block	-	0.382	-	0.038
Binary Matrix Rank Test	0.424	0.980	0.795	0.332
Discrete Fourier Transform Test	-	0.021	0.184	0.692
Nonoverlapping Template Matching Test	-	0.013	-	-
Overlapping Template Matching Test	-	0.322	0.272	0.156
Maurer's Universal Statistical Test	-	0.101	0.735	0.240
Linear Complexity Test	0.615	0.970	0.567	0.337
Serial Test	-	0.435	-	0.634
	0.879	0.126	0.178	0.883
Approximate Entropy Test	-	0.154	0.012	0.126
Cumulative Sums Test	-	0.280	-	0.192

6. Conclusions

The randomness of the numbers obtained from TRNGs depends on the source of entropy. There is a correlation in the generated bit sequences due to the sources of entropy being affected by environmental changes. In the present study, in order to remove the correlation problem, an S-box-based postprocessing method was proposed. The proposed S-box postprocessing algorithm was applied to the pure bits obtained from the RO-based TRNG

Table 7. The change in the entropy of postprocessing methods.

Method	Entropy
Pure bit	0.989
XOR	0.992
H Function	1.0
Von Neuman	1.0
Proposed Sbox Postprocessing	1.0

Table 8. Autocorrelation test results.

	D Value	Pure Bit	Von Neumann	XOR	H Function	S-box
Autocorrelation	8	−2.875	0.208	1.193	0.088	0.526
	15	−3.633	1.491	1.569	0.187	0.698

Table 9. Statistical complexity measure result.

	Pure Bit	Von Neumann	XOR	H Function	S-box
Statistical Complexity Measure	0.013	0.305	0.217	0.204	0.102

system. Pure number sequences generated by the RO-based TRNG did not pass statistical tests. As a result of applying an S-box to the pure bits, successful results were achieved with the NIST tests. Moreover, successful results were achieved with autocorrelation and complexity measures. The proposed S-box postprocessing, when compared to the other postprocessing algorithms, achieved an entropy value of 1, as for the H function and Von Neumann methods. Another advantage of S-box postprocessing is that it can achieve 2/3 of the original data rate for the generated bit sequences. Random bit sequences generated according to these results will be suitable for many applications, such as cryptography.

References

[1] Koç CK. About cryptographic engineering. In: Koç CK (editor). Cryptographic Engineering. New York, NY, USA: Springer, 2009, pp. 5-16.

[2] Avaroğlu E, Koyuncu İ, Özer AB, Türk M. Hybrid pseudo-random number generator for cryptographic systems. Nonlinear Dynamics 2015; 82(1-2):239-248.

[3] Tuncer T. Implementation of duplicate TRNG on FPGA by using two different randomness source. Elektronika Ir Elektrotechnika 2015; 21 (4):35-39. 10.5755/j01.eee.21.4.12779

[4] Suresh VB, Burleson WP. Entropy extraction in metastability-based TRNG. In: IEEE International Symposium on Hardware-Oriented Security and Trust (HOST); Anaheim, CA, USA: IEEE, 2010. pp. 135–140.

[5] Dichtl M. Bad and good ways of post-processing biased physical random numbers. In: Biryukov A (editor). Fast Software Encryption. FSE 2007. Lecture Notes in Computer Science, vol 4593. Berlin, Germany: Springer, 2007. pp. 137–152.

[6] Sunar B, Martin WJ, Stinson DR. A provably secure true random number generator with built in tolerance to active attacks. IEEE Transactions on Computers 2007; 56 (1): 109–119.

[7] Kohlbrenner P, Gaj K. An embedded true random number generator for FPGAs. In: Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays, New York, NY, USA; ACM, 2004. pp. 71–78.

[8] Golic JDJ. New methods for digital generation and post processing of random data. IEEE Transactions on Computers 2006; 55 (10): 1217–1229.

[9] Schellekens D, Preneel B, Verbauwhede I. FPGA vendor agnostic true random number generator. In: International Conference on Field Programmable Logic and Applications; Madrid, Spain; IEEE, 2006. pp. 1–6.

[10] Avaroğlu E, Tuncer T, Özer AB, Ergen B, Türk M. A novel chaos-based post-processing for TRNG. Nonlinear Dynamics 2015; 81 (1-2): 1–11.

[11] Tuna M, Fidan CB. A study on the importance of chaotic oscillators based on FPGA for true random number generating (TRNG) and chaotic systems. Journal of the Faculty of Engineering and Architecture of Gazi University 2018; 33 (2): 469-486.

[12] Alçin M, Koyuncu İ, Tuna M, Varan M, Pehlivan İ. A novel high speed artificial neural network based chaotic true random number generator on field programmable gate array. International Journal of Circuit Theory and Applications 2019; 47 (3): 365-378.

[13] Tuna M, Fidan CB, Koyuncu İ. The chaos-based dual entropy core TRNG on FPGA: VHDL Codes of Chaotic Systems, Beau Bassin, Mauritius: LAMBERT Academic Publication (LAP), 2019.

[14] Koyuncu İ, Özcerit AT. The design and realization of a new high speed FPGA-based chaotic true random number generator. Computers & Electrical Engineering 2017; 58: 203-214.

[15] Loza S, Matuszewski L, Mieczyslaw J. A random number generator using ring oscillators and SHA-256 as post processing. International Journal of Electronics and Telecommunications 2015; 1 (2): 199-204.

[16] Nikolic S, Veinovic M. Advancement of true random number generators based on sound cards through utilization of a new post-processing method. Wireless Personal Communications 2016; 91 (2): 603–622.

[17] Zhang R, Chen S, Wan C, Shinohara H. High-throughput Von Neumann post-processing for random number generator. In: International Symposium on VLSI Design, Automation and Test (VLSI-DAT) Hsinchu, Taiwan; IEEE, 2018. pp. 1–4.

[18] Barak B, Shaltiel R, Tromer E. True random number generators secure in a changing environment. In: Walter CD, Koç C, Paar C (editors). Cryptographic Hardware and Embedded Systems(CHES), Berlin, Germany: Springer, 2003, pp. 166–180.

[19] Webster AF, Tavares SE. On the design of S-Boxes. In: Advances in Cryptology — CRYPTO '85 Proceedings, Berlin, Germany: Springer, 1986, pp. 523–534.

[20] Paar C, Pelzl J. Understanding Cryptography: A Textbook for Students and Practitioners. Berlin, Germany: Springer, 2010.

[21] Stallings W. Cryptography and Network Security Principles and Practices. Upper Saddle River, NJ, USA: Pearson, 2005.

[22] Wold K, Tan CH. Analysis and enhancement of random number generator in fpga based on oscillator rings. In: International Conference on Reconfigurable Computing and FPGAs; Cancun, Mexico; IEEE, 2008, pp. 385–390.

[23] Bassham LE, Rukhin AL, Soto J, Nechvatal JR, Smid ME et al. A statistical test suite for random and pseudo random number generators for cryptographic applications. Special Publication (NIST SP) - 800-22 Rev 1a, 2010.

[24] Avaroğlu E, Türk M. Hardware based realization of random number generator. PhD, Fırat University, Elazığ, Turkey, 2014.

[25] Avaroğlu E. Pseudorandom number generator based on Arnold cat map and statistical analysis. Turkish Journal Of Electrical Engineering & Computer Sciences 2017; 25 (1): 633–643.