Low-Cost Implementations of On-the-Fly Tests for Random Number Generators

Filip Veljković, Vladimir Rožić and Ingrid Verbauwhede
Katholieke Universiteit Leuven, ESAT/SCD-COSIC and IBBT
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
E-mail: veljcomp@yahoo.com, Vladimir.Rozic@esat.kuleuven.be, Ingrid.Verbauwhede@esat.kuleuven.be

***Abstract*—Random number generators (RNG) are important components in various cryptographic systems. Embedded security systems often require a high-quality digital source of randomness. Still, randomness of an RNG can vary due to aging effects, temperature or process conditions or intentional active attacks. This paper presents efficient, compact and reliable hardware implementations of 8 tests from the NIST test suite for statistical evaluation of randomness. These tests can be used for on-the-fly quality monitoring of on-chip random number generators as well as for fast hardware evaluation of RNG designs.**

I. Introduction

Random numbers (RN) are used in many applications ranging from stochastic simulations, statistical experiments, Monte Carlo methods of numerical analysis, to cryptography. Many embedded security systems such as smart cards, eIDs and RFID tags, require a high-quality digital random number generator (RNG). Therefore, an RNG is an essential part of a modern cryptographic system. The purpose of an RNG is to generate random numbers that are uniformly distributed on their range and statistically independent. These numbers are used for generating random session keys, signature parameters, challenges and for other cryptographic purposes.

Quality of the data produced by an RNG is evaluated using statistical tests for randomness. There are several test suites reported in literature such as NIST [1], DIEHARD [2], AIS.31 [3] and FIPS 140-1/2 [4][5]. Randomness of a bit sequence may be assessed using software implementations of these tests. Some statistical tests in these suites require a huge bitstream of test data in order to give meaningful results which makes the testing process difficult and time consuming. In addition, the quality of an RNG can vary due to aging effects, bad implementations or active attacks. In [6] a successful attack on an oscillator-based RNG was performed by injecting frequency components in a power supply, thereby reducing the range of the produced random numbers and consequently proving that RNGs are susceptible to active attacks. For these reasons, there is a need for on-the-fly evaluation and on-line monitoring of an RNG.

This work was supported in part by the European Commissions ECRYPT II NoE (ICT-2007-216676) and UNIQUE (EU FP7), by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and by the Research Council K.U.Leuven GOA TENSE (GOA/11/007).

978-3-9810801-8-6/DATE12/©2012 EDAA

Several papers addressing these issues have been published in recent years. In 2009, authors of [7] and [8] implemented four out of eight FIPS 140-2 [5] statistical tests in hardware in order to evaluate randomness of different RNGs on various operating temperatures. In [9] and [10], four statistical tests of the DIEHARD battery and two statistical tests of the NIST battery have been implemented in order to accelerate testing. Results in hardware have shown that for larger bitstreams reduction in execution time is up to 4 orders of magnitude compared to software statistical tests application. Authors of [11] proposed FPGA implementations of the NIST battery of tests using dynamic reconfiguration. Evaluation time has been decreased and communication overhead between generation and validation of the tested sequence has been eliminated. However, this design is not suitable for on-line monitoring because the architecture does not allow for tests to run while the bitstream is being generated. Also, each of the test implementations consumes too much FPGA resources to be incorporated in a low-cost device.

In this paper we present compact and efficient hardware implementations of 8 NIST tests suitable for on-line monitoring of RNGs. Our tests analyze generated bitstream in real time so that malfunctions can be detected while an RNG is in operation. One of the main design goals was to minimize test reaction time since, in a security system, it is essential to stop an RNG from generating too many statistically weak sequences before the fault is detected. Another goal was low area because these tests have applications on resource constrained devices. Important contribution of this paper is on the mathematical simplifications made on the test equations. By simplifying the arithmetic of the test statistics, we achieved low-cost hardware implementations with low cycle count, which makes them suitable for on-the-fly testing.

This paper is organized as follows. In section 2, the basics of statistical testing for RNGs are explained, the NIST test suite is described and the selection of tests suitable for hardware implementation has been made. Section 3 contains the main contribution of our paper. Interface of each test and methodology of on-line testing is shown and all implementations are analyzed. Results of those test implementations are summarized in section 4 and finally, section 5 gives perspectives and conclusion of this paper.

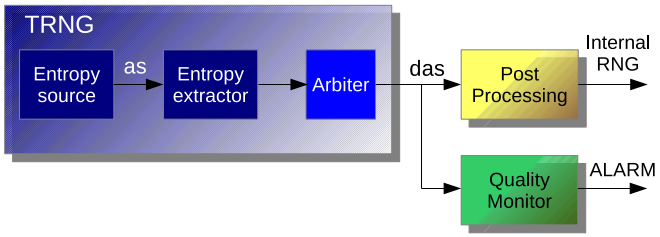

Fig. 1. Generic architecture of a TRNG.

TABLE I
RELATION BETWEEN THE DATA AND THE OUTCOME OF THE TEST.

True Situation	CONCLUSION	
	Accept H_0	Reject H_0
Data is random	No error	Type I error
Data is not random	Type II error	No error

TABLE II
THE NIST TEST SUITE

TEST	HW
1.The Frequency (Monobit) Test	Yes
2.Frequency Test within a Block	Yes
3.The Runs Test	Yes
4.Test for Longest-Run-of-Ones in a Block	Yes
5.The Binary Matrix Rank Test	No
6.The Discrete Fourier Transform (Spectral) Test	No
7.The Non-overlapping Template Matching Test	Yes
8.The Overlapping Template Matching Test	Yes
9.Maurer's "Universal Statistical" Test	No
10.The Linear Complexity Test	No
11.The Serial Test	No
12.The Approximate Entropy Test	No
13.The Cumulative Sums (Cusums) Test	Yes
14.The Random Excursions Test	No
15.The Random Excursions Variant Test	Yes

II. BACKGROUND

A. Classification of RNGs

Digital key-streams used for encryption of confidential data are generated by RNGs. Depending on the entropy source, all RNGs can be divided into 2 classes: *True* RNGs (TRNG) which produce true randomness and *Deterministic* RNGs (DRNG) also known as Pseudo RNGs (PRNG) which use a randomly chosen seed and expand it into a random-like sequence. Fig. 1 shows a generic architecture of a TRNG. These generators use a non-deterministic process as an entropy source (e.g. quantum random processes, free-running oscillators, inherent semiconductor thermal noise, chaos). This randomness source produces time-continuous analog signal (as) which is digitized after uniform time intervals. The output of a TRNG, the digitized analog signal (das), may be used directly as a random number, may need post-processing in order to improve the quality of randomness or may be used as a seed for a PRNG. The process of converting das-numbers to internal RNs, referred to as post-processing, is deterministic with pseudo-random properties. For this reason if a physical noise source completely brakes down and das-numbers become constant, post-processing will produce internal RNs which may pass statistical tests. Therefore, on-line monitoring should be applied before post-processing, i.e., on the das numbers. A good overview of RNGs and evaluation criteria can be found in [12].

B. Statistical evaluation of randomness

For cryptographic purposes, generated sequences need to be sufficiently random. In order to evaluate the quality of RNs various statistical tests are used. Each of those tests determines the presence of a pattern which, if detected, would indicate that the sequence is non-random. Probability plays a significant role in statistical evaluation of randomness. Output sequence can be either accepted as random (null hypothesis H_0 accepted) or non-random (H_0 rejected). If the sequence is truly random then a conclusion to reject H_0 should occur with small probability. This probability is known as Type I error probability (Table I). On the other hand, if the sequence is non-random, then a conclusion to accept H_0 is called Type II error. The probability of Type I error is a design parameter which is set prior to the test, and National Institute of Standards and Technology (NIST) recommends that the value of this probability, also known as the *level of significance* of the test, should lie in range $[0.001, 0.01]$. The primary goal of all tests is to minimize the probability of Type II error. Therefore, testing procedure consists of choosing a proper sample size, setting the probability of Type I error and calculating the test statistics. In this case, the test statistic is used to calculate the P_value, which is the probability that an ideal generator produces a sequence less random than the analyzed sequence. In this context, the term *less random* is used to refer to the metric which is defined in each test. For instance, in the frequency monobit test, ratio of ones in a sequence is used as a measure of randomness. For two sequences with different ratios of ones, the one whose ratio deviates more from one half is considered less random. If Type I error probability is set to 0.001, it is expected that one in 1000 sequences generated by an ideal RNG would be rejected. Therefore, for $P\_value \geqslant 0.001$ analyzed sequence would be considered random with an accuracy of 99.9% and non-random with the same accuracy if $P\_value < 0.001$.

One of the well known batteries of statistical tests for TRNGs and PRNGs is published by the National Institute of Standards and Technology (NIST). The NIST test suite consists of 15 statistical tests, each of them developed for assessing the randomness of binary sequences produced by TRNGs or PRNGs. Software implementations are using complex mathematical operations to calculate the test statistics, including complementary error function and incomplete gamma function. Important facts are that no test suite is considered complete and that a failure of any of the statistical tests would provide the conclusion that an RNG is not truly random.

The focus of this paper is on compact hardware implementations of these tests. Implementations of some algorithms

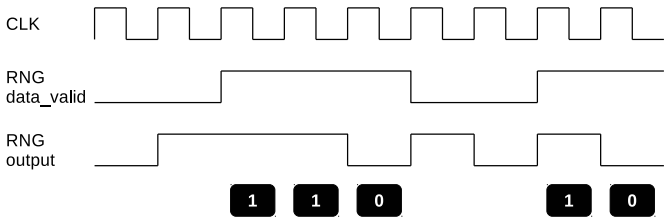

Fig. 2. Valid data produced by an RNG.

from this test suite would either consume too much in terms of arithmetic and memory units or have unacceptably high computation time. The preliminary analysis of test algorithms shows that 8 statistical tests from the NIST test suite are suitable for hardware implementation (Table II-B). The remaining tests were discarded because no solution was found for compact, low-latency implementation. It was estimated that each implementation would eather consume too much area or that it would have high latency.

The selected tests can be implemented using simple arithmetic and a small number of cycles, which makes them suitable for on-the-fly testing. Simplification of complex mathematical calculations are made in order to reduce the amount of arithmetic and memory units. Lengths of the test sequences were chosen prior to implementation, following the input size recommendations given by NIST.

III. Implementations

Each test evaluates an n-bit sequence produced by an RNG, where n differs from test to test. The test interface was designed following the assumption that RNGs generate one bit per clock cycle which is true for most TRNG designs reported in literature. Regarding the fact that low reaction time is the most important requirement for on-line monitoring, all of our tests process each bit immediately after it's generated. Therefore, the output of an RNG is fed to the input signal of the module (INP). The $data\_valid$ signal is introduced to allow testing of variable data rate RNGs. In other words, bits are considered valid only if $data\_valid$ signal is asserted high, as shown in Fig. 2.

A. General Principles

As mentioned in previous section, complex mathematical calculations are needed in order to calculate the test statistics. Software implementations of NIST tests use complementary error function (*erfc*) and incomplete gamma function (*igamc*). However, hardware implementations of these functions would lead to high latency and/or high area consumption. Therefore, simplifications such as storing the precomputed values of the inverse functions of *erfc* and *igamc* are made prior to the implementation.

$$\begin{aligned} \textit{P\_value} &= erfc(x) \\ \textit{P\_value} > \alpha &=> x < erfcinv(\alpha) \end{aligned} \tag{1}$$

$$\begin{aligned} 1 - \textit{P\_value} &= igamc(a, x) \\ \textit{P\_value} > \alpha &=> x < igamcinv(a, (1-\alpha)) \end{aligned} \tag{2}$$

During the fixed-point refinement, care has been taken that no additional Type I and Type II errors are introduced. In tests where operations on non-integer numbers are used, the decimal part is represented with enough bits so that the round-off errors do not change the outcome of the test. These numbers are determined using ad-hoc methods which will not be discussed in detail in this paper.

B. Test Implementations

Eight tests of the NIST test suite are implemented in hardware. The value $\alpha = 0.01$ has been chosen for the border for P_value. Parameters for each test are chosen considering the input size recommendations given by NIST.

1) Frequency (Monobit) Test: This is the first test of the NIST test suite. Its purpose is to determine whether the number of zeros and ones in a sequence is balanced. Sequence length of $n = 20000$ bits is assessed for each evaluation.

■ **Simplification:** According to the simplification given in (1) it is determined that the number of ones should lie in the range $[9818, 10182]$. This design is implemented using a simple counter of ones. Test reaction time is either 1 or 0 clock cycles.

2) Frequency Test within a Block: The focus of this test is on the proportion of ones within M-bit blocks. In other words, this test determines whether the number of zeros and ones in an M-bit block is approximately $\frac{M}{2}$, as would be expected under the assumption of randomness. Input sequence length of $n = 20000$ bits is divided into $N = 100$ blocks each consisting of $M = 200$ bits. The number of ones in each block (ε) is counted.

■ **Simplification:** Using simplification (2) maximal value of χ^2 for a random sequence is determined and shown in (3).

$$\chi^2(obs) = \sum_{i=1}^{N=100} (\varepsilon_i - \frac{M}{2})^2 \leqslant 6790 \tag{3}$$

Inequality (3) is implemented in hardware. This optimization reduces the amount of calculations to one substraction, one squaring and one addition per each 200b block. Squaring was implemented in a digit serial manner to save the area. The design is implemented using a counter of ones, an accumulator, adder, subtractor and a digit-serial squaring module. The highest reaction time of the test is 12 clock cycles.

3) Runs Test: The third test of the NIST test suite verifies whether the number of runs is as expected for a random sequence. A run is an uninterrupted sequence of identical bits. The goal is to determine whether the oscillation between ones and zeros is too fast or to slow. Once again, an input length of $n = 20000$ bits has been chosen for assessing. The total number of ones (ε) and the total number of runs ($V_n(obs)$) are counted.

■ **Simplification:** NIST propose the Frequency (monobit) test as the prerequisite for the Runs test. As a result, if the total number of ones is out of the range $[9818, 10182]$ the test

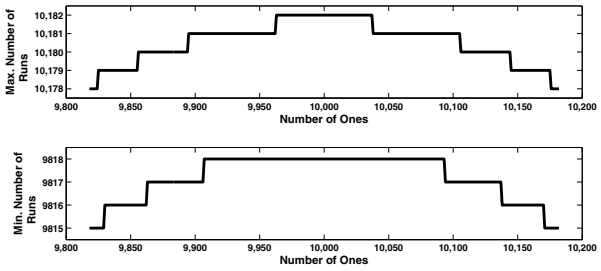

Fig. 3. Maximal and minimal value of $V_n(obs)$ depending of ε.

will fail. Furthermore, by applying the conclusion obtained in (1) it is determined that the sequence should satisfy inequality shown in (4).

$$|nV_n(obs) - 2\varepsilon(n-\varepsilon)| \leqslant \frac{2\sqrt{2n}\cdot\varepsilon(n-\varepsilon)\cdot 1.82138}{n} \quad (4)$$

However, significant simplification of (4) can be made. The graph shown in Fig. 3 presents the maximal and minimal values of $V_n(obs)$ for each value of the number of ones in a range $[9818, 10182]$. Therefore, the test statistic can be calculated using only comparison operations. Maximal latency of the test is 3 clock cycles.

4) The Longest Run of Ones in a Block: The purpose of this test is to determine whether the length of the longest run of ones within the tested sequence is consistent with the length of the longest run of ones that would be expected in a random sequence. A run of length L consists of exactly L ones and is bounded before and after with a zero. Three different choices of parameters, depending on the number of bits in a sequence, are recommended by NIST. Considering recommendations given in the NIST document, test parameters shown in Table III have been chosen for our implementations.

The longest run of ones for each block is recorded so that the blocks with the same length of the longest run could be divided into categories. Blocks are distributed into four, six or seven categories depending on the choice of K. The number of blocks in each category (ν_i) is counted. The most light-weight version of the three ($K = 3$) has been chosen for implementation in hardware.

■ **Simplification:** After applying simplification (2), the condition for passing this test is reduced to a single inequality:

$$\sum_{i=0}^{K=3} \nu_i^2(\frac{1}{\pi_i}) \leqslant 437.52 \quad (5)$$

Precomputed constants ($\frac{1}{\pi_i}$) differ for different values of K and can be found in the NIST document. Counters for all ν_i, logic for determining the longest run and multipliers are used. Latency of the test result is 2 clock cycles. Computations needed for calculating the test statistics are 12 multiplications and 4 additions. The result can be computed using one 11x11 bit multiplier and one adder.

TABLE III
TEST 4 : PARAMETERS

n	Block length-M	Num. of Blocks-N	Deg. of freedom-K
128	8	16	3
6400	128	50	5
750000	10^4	75	6

5) Non-Overlapping Template Matching Test: The purpose of this test is to detect sequences with too many occurrences of a given aperiodic pattern. Therefore, the 4-bit Template input has been added to the test interface. Sequence length of $n = 2048$ bits is divided into $N = 8$ blocks each consisting of $M = 256$ bits. For each block the number of pattern occurrences W_j is being counted. A list of non-overlapping patterns is given in the NIST standard. Each pattern can be used for testing in our design.

■ **Simplification:** By applying mathematical simplifications including the conclusion obtained in (2), the test passing condition is reduced to inequality (6). Therefore, a single counter for the number of pattern occurrences, a comparator for pattern recognition and an accumulator have been implemented in hardware. Worst case latency of the test result is 2 clock cycles.

$$\sum_{j=1}^{N=8} (16\cdot W_j - 253)^2 \leqslant 46287 \quad (6)$$

6) Overlapping Template Matching Test: The focus of this test is on the number of occurrences of pre-specified target strings. The difference between this and the previous test is in the fact that the pre-specified templates can be periodic. Therefore, patterns can overlap. For this test 9-bit template input has been added to the standard test interface. The test design allows evaluation of the input sequence for all possible 9-bit patterns. Input sequence length of $n = 1.032 \times 10^6$ bits has been chosen for assessing. The bit stream is divided into $N = 1000$ blocks each consisting of $M = 1032$ bits. For each block the number of 9-bit pattern occurrences is counted. Blocks with the same number of occurrences are distributed into six categories ($\nu_0..\nu_5$) as the value of degrees of freedom K is set to 5. A software version of this test has inequality (7) as a decision rule.

$$P\_value = igamc(\frac{1}{2}\sum_{i=1}^{K=5}\frac{(\nu_i - N\pi_i)^2}{N\pi_i}, \frac{K}{2}) \geqslant 0.01 \quad (7)$$

■ **Simplification:** In order to obtain a suitable decision rule for our hardware design, mathematical simplifications of (7) needed to be done. As a result, inequality (8) has been implemented in hardware.

$$\sum_{i=0}^{K=5} \nu_i^2(\frac{1}{\pi_i}) \leqslant 1015086.27 \quad (8)$$

In total 6 squaring operations on 10-bit numbers, 6 20x24 bit multiplications and 6 additions are needed to calculate test statistics. This can be performed by using a single 4x20 bit

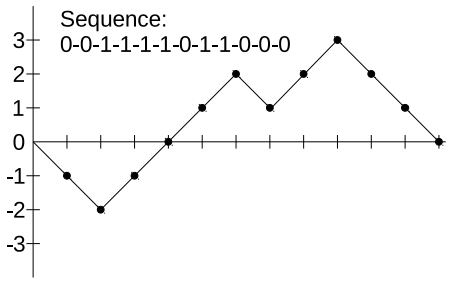

Fig. 4. Example of random walk.

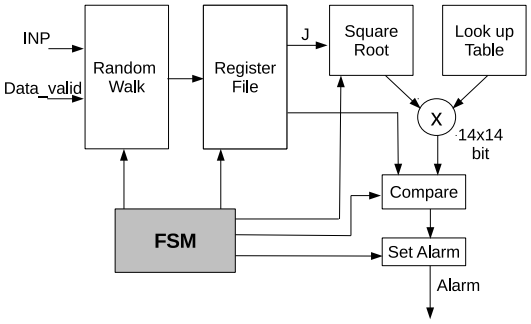

Fig. 5. Architecture of test 15.

multiplier. Each squaring operation takes 3 cycles and each multiplication 6 cycles. Latency of the result is 68 clock cycles.

7) Cumulative Sums Test: This is the thirteenth test of the NIST test suite. Its focus is on the maximal excursion from zero of the random walk defined by the cumulative sum of adjusted (-1, +1) digits in the sequence (Fig. 4). For a random bit stream, the excursions of the random walk should be near zero. Two modes of this test are implemented in hardware, both assessing the same input sequence of $n = 20000$ bits. Unlike in the forward mode shown in Fig. 4, in backward mode partial sums S_k which form the random walk are calculated from the last to the first bit of the sequence.

■ **Simplification:** By applying simplifications on the software algorithm it is concluded that the maximal excursion from zero of the random walk should be 397. Therefore, inequality $-397 \leqslant S_k \leqslant +397$ is implemented in hardware using just one counter and a register for recording the maximal excursion from zero. Latency of the forward mode result is either 2 or 0 clock cycles. The backward mode latency is within 3 clock cycles.

8) Random Excursions Variant Test: The focus of this test is the total number of times that a particular state occurs in a random walk. Its purpose is to detect deviations from the expected number of visits to various states. This test is a series of eighteen different tests, one for each of the states x: $\pm 9 \pm 8$ ... ± 1. The number of zero crossings (J) and the number of state visits ($\xi(x)$) are counted in a sequence of $n = 10^6$ bits since this is the minimal length of the test sequence recommended by NIST for this test.

■ **Simplification:** In order to achieve suitable decision rule for hardware, inequalities (9a) have been simplified by applying the inequality (1). Eighteen arbiters, one for each state, are implemented in hardware as shown in (9b).

$$SW : P\_value = erfc(\frac{|\xi(x) - J|}{\sqrt{2J(4|x| - 2)}}) \geqslant 0.01 \quad (9a)$$

$$HW : \xi(x)(\xi(x) - 2J) \leqslant 6.6349J(4|x| - 2) - J^2 \quad (9b)$$

The equation 9b can be simplified for compact HW implementation by determining the maximal and minimal border for $\xi(x)$. The test gives a positive outcome, when the number of visits to a particular state lies in the interval $(J - \triangle J, J + \triangle J)$ where $\triangle J$ is given by equation:

$$\Delta J = \sqrt{J}\sqrt{2A_x^2(4|x| - 2)} \quad (10)$$

A_x is a constant value for each of the 18 tests. The first term in (10) has to be calculated in hardware, while the second one has a constant value for each of the 18 tests. Therefore, this term can be precomputed and read from a look-up table.

The amount of computation needed to perform this test consists of one 14x14 bit multiplication per each of the 18 tests and a single square root operation. The architecture of the hardware module of test 15. is shown in Fig. 5. State of the random walk is calculated in a simple module consisting of a 19-bit shift register and a counter. The number of visits to each of the 18 states and the number of zero crossings are stored in a register file. Square root operation was implemented in a compact way by using digit-serial architecture. The latency of the test result is 119 clock cycles.

IV. Results

The proposed test modules have been implemented on a Xilinx Spartan-6 XC6SLX45 FPGA. Implementation of each test has been done without using the on-board DSP blocks. Table IV presents the design occupation statistics, the maximal working frequency and the highest latency for these implementations. All test designs, excluding the 15th, occupy less than 1% of slices on FPGA which makes them very compact and therefore suitable for on-the-fly assessing of RNG bitstreams. The operating frequency is higher than $100MHz$ for all tests, which makes them suitable for use in high-throughput FPGA applications. As presented, latency of the test result is low for all implementations which makes our proposal very reliable for on-line quality monitoring of an RNG.

In order to compare our implementations with previously published work, we have synthesized our designs on different platforms. In [11] implementation results are presented for Xilinx Virtex 2 Pro FPGA. As can be seen in Table V, our implementations consume less FPGA resources and work at higher clock frequencies. However, the comparison is not entirely fair since implementations in [11] work with sequences of 8MB and use different interface.

In [10] results for the joint implementation of the frequency test and the runs test are given for Xilinx Virtex 4 XC4VLX60

TABLE IV
THE CHIP UTILIZATION, THE MAXIMUM WORKING FREQUENCY AND THE WORST CASE LATENCY FOR XILINX SPARTAN-6 XC6SLX45 FPGA.

Test		FFs	LUTs	Occupied Slices	Max. Frequency [MHz]	Latency [clk cycles]
1		32 (1%)	44 (1%)	17 (1%)	203	1
2		81 (1%)	98 (1%)	32 (1%)	178	12
3		56 (1%)	115 (1%)	43 (1%)	136	3
4		64 (1%)	168 (1%)	49 (1%)	151	21
7		43 (1%)	153 (1%)	48 (1%)	132	2
8		243 (1%)	356 (1%)	107 (1%)	155	68
13	forward mode	48 (1%)	94 (1%)	28 (1%)	127	2
	backward mode	66 (1%)	135 (1%)	39 (1%)	158	3
15		637 (1%)	757 (2%)	237 (3%)	121	119

TABLE V
COMPARISON WITH PREVIOUS WORK [11]. THE CHIP UTILIZATION AND THE MAXIMUM WORKING FREQUENCY OF THE 6 NIST TEST IMPLEMENTATIONS FOR XILINX VIRTEX 2 PRO FPGA.

	FFs		LUTs		Occupied Slices		Max. Frequency [MHz]	
Test	[11]	This work	[11]	This work	[11]	This work	[11]	This work
1	1204	33	4117	90	2408	49	87	232
2	1355	81	2762	200	1595	110	96	184
3	2151	59	6716	297	3989	160	62	202
4	1746	50	3257	94	2000	48	93	258
7	1305	46	2782	127	1646	67	93	145
8	1573	252	3514	411	2101	225	83	128

TABLE VI
IMPLEMENTATION RESULTS OF THE FREQUENCY (MONOBIT) TEST AND THE RUNS TEST FOR XILINX VIRTEX-4 XC4VLX60 FPGA.

	[10]	This work
FFs	452	92
LUTs	828	405
Max. Frequency [MHz]	20	248

FPGA. By creating a joint implementation of these tests and performing synthesis for the same platform we obtained the results presented in Table VI. The results show that our implementation is less area consuming and works at higher clock frequencies.

V. CONCLUSION

In this paper we have presented compact and reliable hardware implementations of 8 tests from the NIST statistical test suite. Implementation results have been given for Xilinx Spartan-6 FPGA target. Designs consuming less than 1% of FPGA resources are suitable for efficient on-line monitoring of RNGs. In addition, each of these test modules can be used in order to provide simple and fast evaluation of the randomness behavior of TRNGs thereby avoiding communication overhead which is a common problem when statistical testing is done in software. Compared to previously proposed implementations [10] [11] our designs of the NIST tests are less area consuming and therefore more appropriate for low-cost embedded security systems.

Area consumption of the proposed RNG quality monitor can be improved by sharing hardware resources between different tests. In future works, we will address the topic of integrating the proposed implementations into a unified module.

Another topic for future research is using the digit-serial arithmetic and exploring the tradeoff between area and latency.

REFERENCES

[1] A. Rukhin et al., "A statistical test suite for random and pseudorandom number generators for cryptographic applications." Special-Pub:800-22 NIST, August 2008. [Online]. Available: http://csrc.nist.gov/groups/ST/toolkit/rng/documentation software.html

[2] "DIEHARD battery of tests of randomness." [Online]. Available: http://www.stat.fsu.edu/pub/diehard/

[3] W. Killman and W. Schindler, "AIS31. A proposal for: Funcionality Classes and Evaluation Methodology for Physical Random Number Generators," September 2001.

[4] "FIPS 140-1, Security requirements for cryptographic modules," 1999. [Online]. Available: http://csrc.nist.gov/publications/fips/fips140-1

[5] "FIPS 140-2, Security requirements for cryptographic modules," 1999. [Online]. Available: http://csrc.nist.gov/publications/fips/fips140-2

[6] A. T. Markettos and S. W. Moore, "The frequency injection attack on ring-oscillator-based true random number generators," in *CHES*, 2009, pp. 317–331.

[7] R. Santoro, O. Sentieys, and S. Roy, "On-line monitoring of random number generators for embedded security." in *ISCAS*. IEEE, 2009, pp. 3050–3053.

[8] ——, "On-the-Fly Evaluation of FPGA-Based True Random Number Generator," in *IEEE Computer Society Annual Symposium on VLSI, ISVLSI'09*, May 2009.

[9] A. Vaskova, C. López-Ongil, E. San Millán, A. Jiménez-Horas, and L. Entrena, "Accelerating secure circuit design with hardware implementation of diehard battery of tests of randomness," in *On-Line Testing Symposium (IOLTS), 2011 IEEE 17th International*, july 2011.

[10] A. Vaskova, C. López-Ongil, A. Jiménez-Horas, E. San Millán, and L. Entrena, "Robust cryptographic ciphers with on-line statistical properties validation," in *On-Line Testing Symposium (IOLTS), 2010 IEEE 16th International*, july 2010, pp. 208 –210.

[11] D. Hoţoleanu, O. Creţ, A. Suciu, T. Gyorfi, and L. Văcariu, "Real-time testing of true random number generators through dynamic reconfiguration," in *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, sept. 2010, pp. 247 –250.

[12] W. Schindler and W. Killmann, "Evaluation criteria for true (physical) random number generators used in cryptographic applications," in *CHES 2002. LNCS*, vol. 2523, dec. 2003, pp. 431 –449.