PAPER *Special Section on Forefront Computing*

Design and Implementation of an On-Line Quality Control System for Latch-Based True Random Number Generator

Naoki FUJIEDA[†a)], ***Member***, **Shuichi ICHIKAWA**[††b)], ***Senior Member***, **Ryusei OYA**[†], ***and*** **Hitomi KISHIBE**[††], ***Nonmembers***

SUMMARY This paper presents a design and an implementation of an on-line quality control method for a TRNG (True Random Number Generator) on an FPGA. It is based on a TRNG with RS latches and a temporal XOR corrector, which can make a trade-off between throughput and randomness quality by changing the number of accumulations by XOR. The goal of our method is to increase the throughput within the range of keeping the quality of output random numbers. In order to detect a sign of the loss of quality from the TRNG in parallel with random number generation, our method distinguishes random bitstrings to be tested from those to be output. The test bitstring is generated with the fewer number of accumulations than that of the output bitstring. The number of accumulations will be increased if the test bitstring fails in the randomness test. We designed and evaluated a prototype of on-line quality control system, using a Zynq–7000 FPGA SoC. The results indicate that the TRNG with the proposed method achieved 1.91–2.63 Mbits/s of throughput with 16 latches, following the change of the quality of output random numbers. The total number of logic elements in the prototype system with 16 latches was comparable to an existing system with 256 latches, without quality control capabilities.

key words: *FPGA, true random number generator, randomness test*

1. Introduction

Unpredictable random numbers are essential for many security applications [1]. They are, for example, used as encryption keys for cryptographic algorithms and nonces of challenge-response protocols. A TRNG (True Random Number Generator) obtains them from a physical phenomenon, which cannot be predicted in principle. There are many types of TRNGs according to physical phenomena on which they rely [2], [3].

An on-the-fly randomness test circuit is a key component for TRNGs to deal with the variation of quality of random numbers. It can be used in three ways: anomaly detection, autocalibration, and quality control/assurance. Since the quality may vary with operating conditions such as supply voltage and temperature, an attacker may interfere with it. It is known that such an attack to, for example, a TRNG based on ring oscillators is possible [4]. A sign of failure should be detected as an anomaly to stop the generation or warn the system about it [5]. Some types of TRNGs can control their behavior by a parameter input. They need an autocalibration mechanism to find a proper parameter for each individual device and/or operating condition [6], [7]. Some other types of TRNGs are designed to improve its randomness at the cost of increasing power consumption or decreasing bit rate of generation. On-line quality control techniques have been recently proposed for them to make a better trade-off [8], [9].

This paper presents a design and an implementation of an on-line quality control method for a TRNG, along with on-the-fly randomness test circuits. Our method is based on a latch-based TRNG implemented on an FPGA (Field-Programmable Gate Array) [10], [11]. By increasing the number of temporal accumulation of the output of latches, it can improve the randomness quality at the sacrifice of bit rate of generation [12].

The proposed method has two important differences from other quality control methods. First, it adopts a relatively complicated randomness test, called the count-the-ones test, as one of the on-the-fly tests. Although there have been FPGA implementations of complicated randomness tests [13], [14], they require too many logic elements to be used as a component of a quality control method. By a careful selection of test and an optimized design of dataflow, we successfully implemented a complicated test with a hardware cost not much different from simple tests. Second, the proposed method distinguishes the random bitstrings to be tested from those to be output. The test bitstring is generated to contain less entropy. This is done by controlling the number of accumulations for the test bitstring to be smaller than that for the output bitstring. This enables the output bitstring to have an enough margin in entropy, so that the TRNG can continue outputting random numbers even though the test bitstring fails at an on-the-fly test. We developed a prototype of the proposed quality control system and evaluated it using PYNQ–Z1 FPGA SoC development boards [15] to validate this idea.

The content of this paper is based on two previously presented studies: one was presented in the IEEJ Technical Meeting on Innovative Industrial System held in March 2021 [16]; the other was presented in the 10th International Symposium on Computing and Networking (CANDAR 2022) [17]. Major differences from the previous studies are summarized as follows:

Manuscript received December 12, 2022.
Manuscript publicized March 24, 2023.

[†]The authors are with Department of Electrical and Electronics Engineering, Faculty of Engineering, Aichi Institute of Technology, Toyota-shi, 470–0392 Japan.
[††]The authors are with Department of Electrical and Electronic Information Engineering, Toyohashi University of Technology, Toyohashi-shi, 441–8580 Japan.
a) E-mail: nfujieda@aitech.ac.jp
b) E-mail: ichikawa@ieee.org

DOI: 10.1587/transinf.2023PAP0001

- a new evaluation system, explained in Sect. 3, has been developed based on the PYNQ platform for improved generality and reliability;
- statistical tests are analyzed in more detail in the preliminary evaluation described in Sect. 4;
- an implementation of a short-term test, which complements the count-the-ones test, is introduced in Sect. 5.3; and
- a prototype of the proposed quality control system is described and evaluated in Sects. 6 and 7, respectively.

This paper is organized as follows. In Sect. 2, the latch-based TRNG is briefly explained and the existing hardware implementations of on-the-fly testing of randomness quality are reviewed. Our evaluation system and preliminary evaluation results are presented in Sects. 3 and 4, respectively. Section 5 describes hardware implementations of on-the-fly randomness tests used as components of the proposed method. In Sect. 6, we present the design of the proposed on-line quality control method and the implementation of TRNG core module with it. We evaluate the prototype system in Sect. 7 and then conclude the paper in Sect. 8.

2. Background

2.1 Latch-based TRNG

The latch-based TRNG utilizes the metastability of an RS latch. Figure 1 depicts an RS latch, where both of the input ports (R and S) are connected to the same signal, CTRL. When the CTRL signal is set to '0,' output of the both NAND gates becomes '1.' When CTRL rises to '1,' the latch transits into the metastable state and starts to oscillate [18]. It eventually settles at one of the stable states and the output of one of the NAND gates, Q, becomes either '0' or '1.' Ideally, the probability that Q becomes '1' is 1/2 because which stable state the latch will go is determined by noises (such as thermal noise) amplified in the ring of the latch. The latch thus outputs Q as a random bit. After that, CTRL is set to '0' again to start a new cycle of random bit generation.

Although an ideal, completely balanced RS latch would generate random bits with perfect randomness, there exists an imbalance in the ring of the latch in practice and the probability usually becomes far from 1/2. This means that the amount of information (or entropy) from one cycle of one latch becomes much less than one bit. Therefore, latch-based TRNG aggregates information from multiple latches using XOR (exclusive OR) operation and obtains (almost) one bit of entropy per one output bit. This technique is called XOR correction.

XOR correction can be applied either spatially or temporally (Fig. 2). Spatial implementation (Fig. 2 (a)) aggregates the output of multiple latches with an XOR gate [10], [11]. Spatial/temporal implementation (Fig. 2 (b)) additionally accumulates the results of the multiple cycles [12]. In the latter implementation, the output becomes valid every time a predefined number of cycles are completed. This means the bit rate of generation is divided by the number of accumulations. Temporal correction can be done in a unit of multiple bits. For example, an existing implementation [12] conducts accumulation by a 32-bit word.

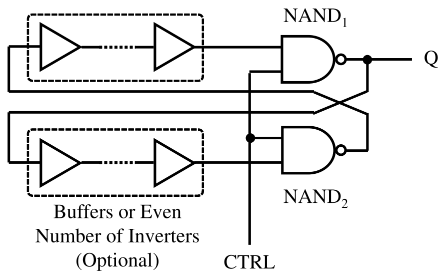


Fig. 1 RS latch where two input ports become in common.

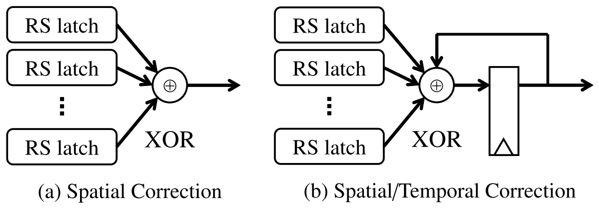


Fig. 2 Implementations of XOR corrector.

Another type of TRNGs that utilizes the metastability of an RS latch is called TERO (Transition Effect Ring Oscillator) [19], [20]. It focuses on the number of times of oscillation before the latch transits into a stable state, rather than the final output of the latch. The design principle of the TERO-based TRNG is different from the latch-based one. It has to let the latch stay at the metastable state for a while in order to increase the variation of the number of times of oscillation. To this end, some TERO implementations add buffers (or even number of inverters) to the ring [2].

The most serious weakness of the original TERO was that its behaviour varied widely with devices and placement of logic elements [2]. This problem has recently been overcome by making the path of the ring configurable by an input parameter [21]. The organization of the ring was inspired from a TRNG based on coherent sampling (COSO) [6] and an autocalibration mechanism in the COSO-based TRNG might also be adopted. However, the mechanism only estimates the likeliness to have enough entropy by measuring the difference of oscillating frequencies (in the case of COSO) or the average number of times of oscillation (in the case of TERO): the randomness of the output is not checked directly.

2.2 Hardware Implementation of Randomness Test

Table 1 enumerates recent studies on hardware implementations of randomness tests. The column labeled *Aim* represents the main target of each circuit. The column labeled # describes the number of tests implemented. Some implementations adopted tests defined in standards such

Table 1 List of hardware implementations of randomness tests.

Authors	Year	Aim	#	Origin
Santoro *et al.* [13]	2009	AD	1	AIS–31
Varchola & Drutarovský [22]	2009	AD	4	FIPS 140–2
Vaskova *et al.* [14]	2011	SU	4	Diehard
Veljković *et al.* [23]	2012	AD	8	NIST 800–22
Suresh *et al.* [24]	2013	AD	6	NIST 800–22
Yang *et al.* [25]	2015	AD	9	NIST 800–22
Yang *et al.* [26]	2015	AD	4	None
Cao *et al.* [27]	2016	AD	4	None
Hussain *et al.* [28]	2016	AD	7	NIST 800–22
Grujić *et al.* [29]	2017	AD	3	None
Martin *et al.* [30]	2018	AD	7	FIPS/NIST
Gonzalez *et al.* [9]	2018	QC	3	Yang *et al.* [26]
Carreira *et al.* [7]	2020	AC	4	None
Gantel *et al.* [31]	2020	QC	3	NIST 800–90B

* AD = Anomaly Detection
* SU = Speed-Up processing of test
* QC = Quality Control
* AC = AutoCalibration

as AIS–31 [32], FIPS 140–2 [33], NIST 800–22 [34], and NIST 800–90B [35]. Another implementation targeted tests in the well-known diehard test suite [36]. The origin of implemented tests are summarized in the column labeled *Origin*.

Most of early studies used the implemented circuits for anomaly detection: they detect the loss of entropy in a TRNG due to malicious attacks and to alarm systems. Only some latest studies adopt an on-the-fly test circuit as a component of autocalibration and quality control systems. We briefly introduce such studies in the following paragraphs.

Gonzalez *et al.* [9] proposed a method to control a TRNG using the results of on-line tests. They used two types of TRNGs based on ring oscillators and self-timed rings, respectively, and turned off some components according to the test results. In short, the purpose of their study is power reduction, which is different from our study.

Carreira *et al.* [7] presented an autocalibration method for a TRNG, based on a variant of TERO [20]. This type of TRNG might give biased output in case of taking too long time to generate random bits. To filter out such a case, the method checks the number of clock cycles for generation, in addition to generated output itself, when it searchs for an appropriate parameter. The idea of estimating entropy from the behavior of circuit is also seen in a quality control system proposed by Chen *et al.* [8]. The motivation of their work is quite similar to ours, but their system did not test the output directly.

Gantel *et al.* [31] proposed a validation and post-processing platform for TRNGs. Test results are used to determine if the output of the TRNG needs to be post-processed. Although the XOR correction (explained in Sect. 2.1) can be considered as one of the post-processing methods, our method is designed to let the TRNG output have enough entropy and require no additional post-processing.

In this study, we propose an on-line quality control method designed for the latch-based TRNG. A short-term, simple tests, adopted in the most of the recent studies, might overlook a small bias of random numbers. A long-term, complicated tests generally requires a large number of logic elements. We selected the count-the-ones test in the diehard suite [36] as the implementation target. It has a strong capability of detecting a bias in output of the latch-based TRNG, and moreover, it can be implemented with a minimal amount of hardware. We demonstrate them in the following sections of this paper.

To the best of our knowledge, Vaskova *et al.* [14] have presented the only hardware implementation of tests from the diehard suite. However, the goal of their research is to accelerate the processing of time-consuming tests. The amount of hardware of their implementation was too large to be used for on-the-fly testing of a TRNG.

3. Evaluation System

3.1 System Description

Figure 3 describes the simplified block diagram of our evaluation system, which collects random bitstrings with various numbers of latches. The system runs on a Digilent PYNQ–Z1 development board, which includes a Xilinx Zynq XC7Z020 FPGA SoC, an Ethernet port, and a MicroSD slot. A Zynq SoC consists of processing system (PS) and programmable logic (PL). The PS is a dual-core Arm Cortex-A9 microprocessor that can run Linux, while the PL is an FPGA fabric where the TRNG is implemented. The PS and the PL usually communicate each other via AXI (Advanced eXtensible Interface) interconnects.

The TRNG is packaged into an IP (Intellectual Property), whose internal structure is depicted in the lower half of Fig. 3. Generated random numbers are sent out as a stream, which means the TRNG core does not care where they shall be stored in the main memory. It is managed by an existing AXI DMA (Direct Memory Access) core. Both the TRNG and the DMA cores are controlled by the PS. A reset signal generater, which is also present in the PL, is omitted from Fig. 3.

The TRNG IP consists of a TRNG core, an AXI (Lite) interface, a stream controller, and a stream FIFO. The TRNG to be evaluated is required to have specific ports and instantiated as a TRNG core. The AXI interface receives control signals and arguments, which include the number of bytes to send, a parameter for the TRNG core, etc., from the PS. The stream controller manages the number of sent bytes and packs data from the TRNG core into a stream of 32-bit words as needed. The words are stored to the stream FIFO and sent out as AXI stream.

Figure 4 depicts the internal block diagram of the TRNG core used in the preliminary evaluation in Sect. 4. Eight latches compose a set of latches, where their outputs are corrected by XOR. The TRNG has 32 sets of latches and XOR output of either of 1–32 set(s) is selected by a 32-to-1 multiplexer. As a result, XOR output of either of 8, 16, . . . , or 256 latches is obtained. The cycle time of the input of

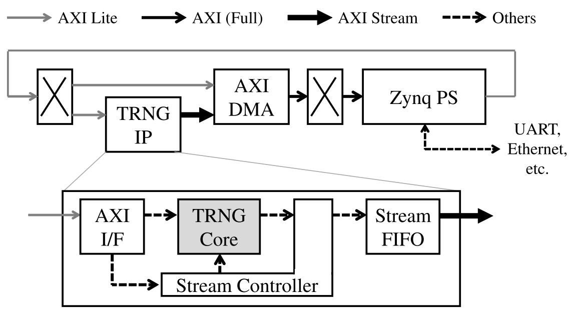


Fig. 3 Simplified block diagram of the evaluation system.

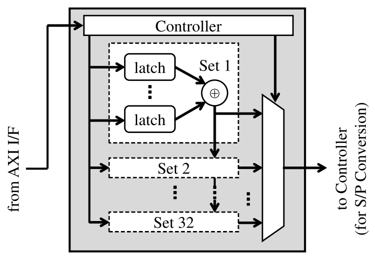


Fig. 4 Block diagram of the TRNG core for the preliminary evalution.

latches is set to 80 ns.

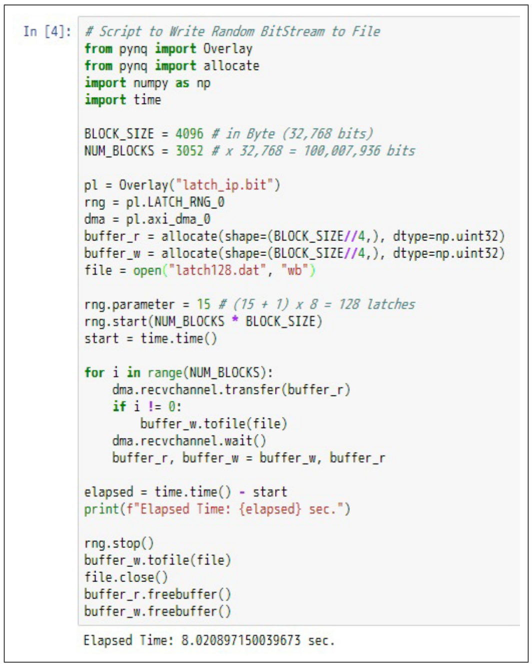
We run the evaluation system on the PYNQ platform [15], which includes Linux, an execution environment of Python, and libraries for circuits on the PL. After copying the programming (.bit) file of the evaluation system to the specific directory, we reconfigure the PL and drive the cores using a Python script written on a Web browser (Jupyter Notebook). Collected random numbers are saved in the PYNQ's file system built on a MicroSD card. The source code of the evaluation system is available at a GitHub repository [37].

3.2 Validation

Figure 5 is a screenshot of Jupyter Notebook running on the PYNQ platform. This Python script reconfigures the PL side, initializes the TRNG and DMA cores, and writes the TRNG output to a file using double buffering. Since the cycle time of the TRNG is set to 80 ns, the ideal throughput of the TRNG is approximately 12.5 Mbit/s. The script took about 8.021 s to obtain and write 100 Mbit of TRNG output, which meant the measured throughput was about 12.47 Mbit/s. As long as used in this study, a bottleneck does not exist in data transfer, which means generated random numbers are not discarded due to limitations of the system.

4. Preliminary Evaluation

This section presents a preliminary evaluation using the system described in Sect. 3. The purpose of the preliminary evaluation is to select randomness tests that have a high capability of detecting a flaw of output of the latch-based


```
In [4]: # Script to Write Random BitStream to File
        from pynq import Overlay
        from pynq import allocate
        import numpy as np
        import time

        BLOCK_SIZE = 4096 # in Byte (32,768 bits)
        NUM_BLOCKS = 3052 # x 32,768 = 100,007,936 bits

        pl = Overlay("latch_ip.bit")
        rng = pl.LATCH_RNG_0
        dma = pl.axi_dma_0
        buffer_r = allocate(shape=(BLOCK_SIZE//4,), dtype=np.uint32)
        buffer_w = allocate(shape=(BLOCK_SIZE//4,), dtype=np.uint32)
        file = open("latch128.dat", "wb")

        rng.parameter = 15 # (15 + 1) x 8 = 128 latches
        rng.start(NUM_BLOCKS * BLOCK_SIZE)
        start = time.time()

        for i in range(NUM_BLOCKS):
            dma.recvchannel.transfer(buffer_r)
            if i != 0:
                buffer_w.tofile(file)
            dma.recvchannel.wait()
            buffer_r, buffer_w = buffer_w, buffer_r

        elapsed = time.time() - start
        print(f"Elapsed Time: {elapsed} sec.")

        rng.stop()
        buffer_w.tofile(file)
        file.close()
        buffer_r.freebuffer()
        buffer_w.freebuffer()

        Elapsed Time: 8.020897150039673 sec.
```

Fig. 5 A screenshot of Jupyter Notebook on PYNQ in collecting a random bitstring.

Table 2 Evaluation criteria for *p*-values.

Decision	Condition (for diehard)	Condition (for NIST)
P (pass)	$0.005 \leq p < 0.995$	$0.01 \leq p$
W (weakly pass)	$10^{-6} \leq p < 0.005$ or $0.995 \leq p < 1 - 10^{-6}$	$2 \times 10^{-6} \leq p < 0.01$
F (fail)	$p < 10^{-6}$ or $1 - 10^{-6} \leq p$	$p < 2 \times 10^{-6}$

TRNG. They will be the candidates for being adopted by the proposed quality control method.

4.1 Methodology

The quality of random numbers can be evaluated by statistical tests. Basically, the randomness quality of the latch-based TRNG is degraded when the number of latches or the number of accumulations (or both) is small. To let an on-line quality control method work effectively, we have to find an appropriate statistical test, where a small defect will result in a *p*-value extremely close to 0 or 1.

For this reason, we conduct a preliminary evaluation using the statistical tests in the diehard test suite [36] and the NIST 800–22 test suite [34]. The diehard and NIST suites consist of 18 and 15 kinds of tests, respectively, each of which gives 1–148 *p*-values per random bitstring. If ideal random numbers are given to the test, *p*-values will be uniformly distributed in the range of $[0, 1)$.

In the preliminary evaluation, we group *p*-values into three categories: pass (P), weakly pass (W), and fail (F), according to the conditions shown in Table 2. Since the expected occurrence probability of F is 2×10^{-6}, an F means that the generated random numbers are likely to be far from ideal. On the other hand, it is a normal behavior that a W is occurred in about 1% of probability because the expected probability of W is $0.01 - 2 \times 10^{-6}$. Categorization criteria are slightly different between the two suites, which comes from whether the tests assume one-sided or two-sided.

In a similar way to the preceding study [10], we determine pass or fail of each kind of test in the following way. If a test outputs four or less p-values, their categories are checked. When they include one or more Fs, the result of the test is F. When they include no Fs but one or more Ws, the result is W. When all p-values are classified into P, the result is also P. If a test gives five or more p-values, their uniformity is checked by the Kolmogorov–Smirnov (KS) test and the test result is determined by the category of the resulting p-value.

The preliminary evaluation was conducted with a single PYNQ-Z1 board. We generated 1.5 Gbit (15×100 Mbit) of random bitstring for each number of latches using our evaluation system described in Sect. 3. We then simulate the temporal XOR correction by software and obtained a 100 Mbit of bitstring for each number of accumulations. Bitstrings without temporal correction are also tested; in this case, the number of accumulations is regarded as zero. The total number of bitstrings to be tested was 256, because we have 16 kinds of the number of latches (8, 16, . . . , 128) and 16 kinds of the number of accumulations (0, 1, . . . , 15). The quality of each bitstring is checked by the test suites. In the NIST tests, only the first 1 Mbit of each bitstring was tested.

4.2 Results

Figure 6 plots the proportion of the p-values generated from 256 bitstrings for each test, categorized into F and W. It clearly shows that the likelihood of failing at a test varies widely with the kind of test.

It is also observed that some diehard tests gave larger proportion of F than NIST tests. In the NIST suite, a p-value of an individual attempt of a test is not used for determining a pass or fail of a TRNG. Instead, a set of p-values with multiple bitstrings for each test are used by a final analysis, as we will explain later in Sect. 7.1. In short, a p-value extremely close to 0 or 1 for each attempt is not necessary in the NIST tests.

There are four tests with excessively high proportion of W: the overlapping 5-permutation test (9.2%), the overlapping sums test (6.6%), the random excursions test (6.7%), and the random excursions variant test (27.6%). Regarding the first two tests from the diehard suite, it has been reported that their reliability is low: they have higher possibility of giving false positives than expected [38]. For the last two tests from the NIST suite, probably it was not appropriate to conduct the KS test to resultant p-values. They have another problem that they cannot be conducted to all bitstrings. For these reasons, we exclude these four tests and focus on remaining 29 tests in the subsequent discussions.

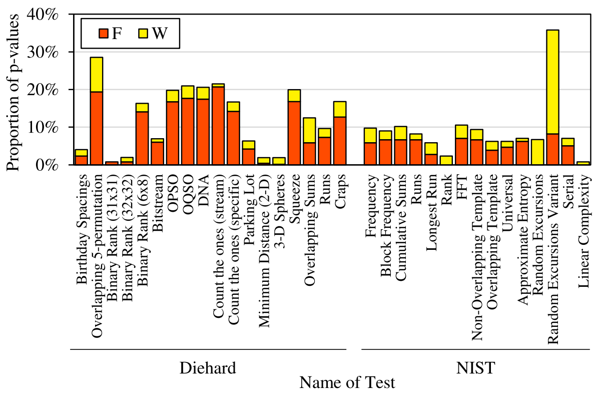


Fig. 6 Proportion of the results for each test.

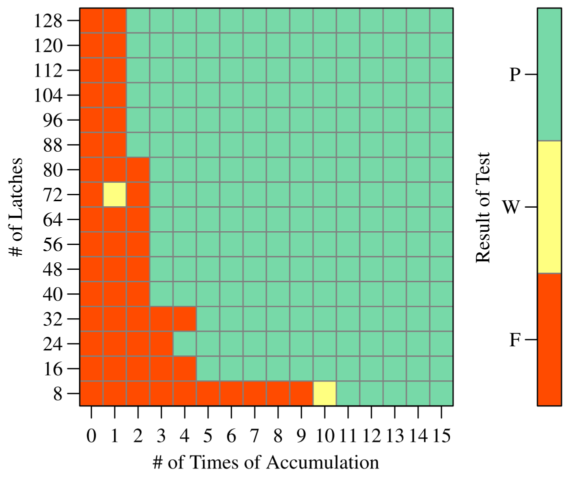


Fig. 7 Heat map of the results of the count-the-ones (stream) test.

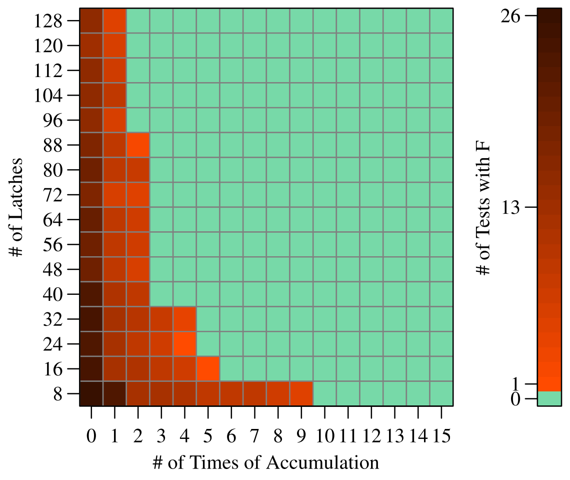


Fig. 8 Heat map of the overall results of the selected 29 tests.

4.3 Discussion

Figures 7 and 8 describe the results of the count-the-ones (stream) test and all of the selected tests, respectively, for each number of accumulations (X-axis) and each number of latches (Y-axis). In Fig. 7, results of P, W, and F is colored in green, yellow, and red, respectively. In Fig. 8, results without Fs correspond to green, while results with one or more Fs correspond to red. The more tests fail at, the darker the color.

These heat maps show that the results of the count-the-ones (stream) test well represent the overall results of the tests. In other words, this test has a high capability of detecting a flaw of output of the latch-based TRNG. There are some other types of tests gave similar results in the diehard suite: the binary rank (6x8) test, the OPSO (overlapping pairs sparse occupancy) test, the OQSO (overlapping quads sparse occupancy) test, the DNA test, the count-the-ones (specific) test, the squeeze test, and the craps test. By carefully considering the ease of implementation by hardware for them, we decided to adopt the count-the-ones (stream) test as one of the on-the-fly randomness tests for the proposed quality control method. In the following sections, we simply refer the count-the-ones (stream) test to as the count-the-ones test.

5. Randomness Testing Circuits

5.1 Analysis of Count-the-Ones Test

We briefly explain the count-the-ones test here. The test uses 256,004 bytes of random numbers. It first translates each input byte into an alphabet by the number of ones in the byte, according to Table 3. It then forms sets of 256,000 overlapping 5-letter and 4-letter words, counts the frequencies of each word, and calculates the χ^2 value for each set. The distribution of the difference between the χ^2 values, represented as *chsq* in the rest of this paper, follows the χ^2 distribution with degree of freedom of 2,500: its mean and standard deviation becomes 2,500 and $\sqrt{5{,}000}$, respectively.

Our considerations on hardware implementation of the count-the-ones test are threefold. The first is an efficient use of ROMs. The calculation of the χ^2 value requires the expected number of occurrence, *ev*, for each word. It can be calculated in advance from the number of total words (256,000) and the occurrence probabilities of the alphabets in the word (see Table 3). This means the *ev* values can be stored in a ROM. Using the fact that 'A' and 'E', 'B' and 'D' have the same probability, respectively, the size of the ROM can be reduced [17]. Since *ev* is also used as a divisor, another ROM for its reciprocal, *ev_r*, is prepared to transform a division into a multiplication.

The second consideration is the use of fixed-point arithmetic, applied to the calculation of the *chsq* value. More specifically, the variables *ev* and *chsq* are stored as fixed-point numbers. Since the bit widths of the fractional portions of the variables affect the computational precision, a trade-off between the error and the amount of hardware has to be made [17].

The last consideration is the exclusion of the calculation of the *p*-value from hardware implementation. Note that the *p*-value of the test is not necessarily calculated: if we only have to know whether the *p*-value is within a certain range, we can check if the *chsq* value is within the corresponding range. For example, the test result being an F corresponds to the *chsq* value meeting $|chsq - 2500| \geq 237$. For this reason, we let the *chsq* value be the test result, instead of the *p*-value.

Table 3 Mapping of alphabets in the count-the-ones test.

# of ones	0–2	3	4	5	6–8
Alphabet	A	B	C	D	E
Probability	37/256	56/256	70/256	56/256	37/256
Encoding	`000`	`010`	`100`	`011`	`001`

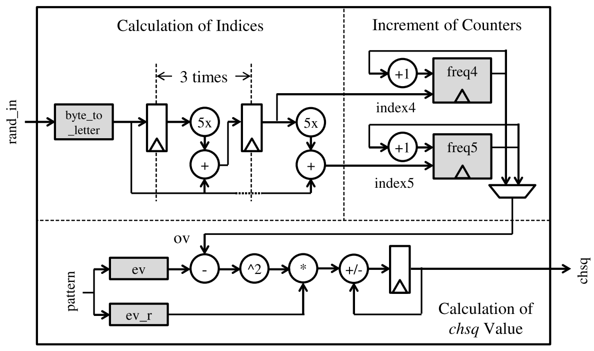


Fig. 9 Block diagram of our implementation of the count-the-ones test [17].

5.2 Implementation of Count-the-Ones Test

Figure 9 depicts a block diagram of our implementation of the count-the-ones test. Since we coded the test in C++ and used high-level synthesis (HLS) to obtain the test circuit, pipeline registers inserted automatically by an HLS tool are omitted from the figure. RAMs and ROMs are shown in gray. The *byte_to_letter* ROM translates a byte into the corresponding alphabet. The *freq*4 and *freq*5 RAMs store the numbers of occurrence of 4-letter and 5-letter words, respectively.

The diagram has three sections: calculation of indices, increment of counters, and calculation of the *chsq* value. If the test circuit receives an input byte, the indices of RAMs are calculated in the first section and the corresponding counters are incremented in the second section, in a pipelined structure. This pipeline proceeds once in 3 cycles because the number of occurrence in the RAMs must be read, incremented, and written. The total number of cycles to complete these sections is estimated at 768,012.

The *chsq* value is calculated in the third section. A counter value is read from one of the RAMs as *ov*. The expected value of the corresponding word and its reciprocal are read from ROMs as *ev* and *ev_r*, respectively. From them, a term of χ^2 value, $(ov - ev)^2 \times ev\_r$, is calculated. It is added to or subtracted from the *chsq* value, according to which RAM was read. After the accumulation for all of the words, the circuit halts and outputs the final *chsq* value. This section approximately takes 3,750 cycles because the num-

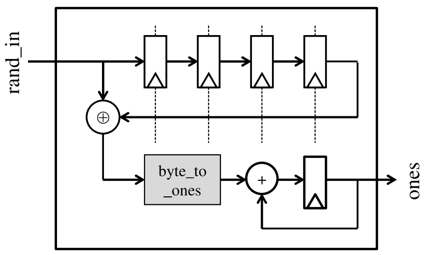


Fig. 10 Block diagram of our implementation of short-term test, a variant of monobit frequency test.

ber of possible 4-letter and 5-letter words is $5^4 + 5^5 = 3{,}750$. In addition, RAMs must be reset to zero as an initialization. This takes approximately 3,125 cycles.

In summary, the number of cycles to complete the test is estimated at 774,887. This means the throughput of the test circuit becomes approximately 2.64 bits/cycle ($\simeq 256{,}004 \times 8/774{,}887$). According to a report after HLS, the actual number of cycles became very close to this estimation [17]. Since the throughput of the test circuit is much larger than that of the latch-based TRNG, the test circuit cannot be a performance bottleneck.

5.3 Implementation of Short-Term Test

In addition to the count-the-ones test for checking long-term quality of random numbers, we implement a short-term test based on the monobit frequency test. There are two goals of the short-term test. One is to deal with a characteristic of the latch-based TRNG that the decline of the randomness quality occurs suddenly and successively [10]. The other is to terminate the random number generation when the latches do not generate random bits at all for some reasons.

The test gets the difference between adjacent 32-bit words by XOR and applies the monobit test (i.e., counting the number of ones in a block of a fixed length) to them. When the latches constantly generates zeros or ones, the accumulated output word becomes the same as the previous word or its inversion, respectively. In the case of outputting inversion, the occurrence probability of both zeros and ones becomes 1/2. This case cannot be detected by simply applying the monobit test to the accumulated output. This is why the difference between adjacent words is required.

Figure 10 depicts the block diagram of the implemented short-term test circuit. To let the circuit be integrated with the count-the-ones test, it has a 8-bit data input. The latest four bytes of data are stored in a shift register and the difference from the previous word is obtained by an XOR. The number of ones in the difference is accumulated. This procedure is repeated for 2,000 bytes (16,000 bits) and the sum of the number of ones is output as a result. It approximately follows a normal distribution with a mean of 8,000 and a standard deviation of $\sqrt{4{,}000}$.

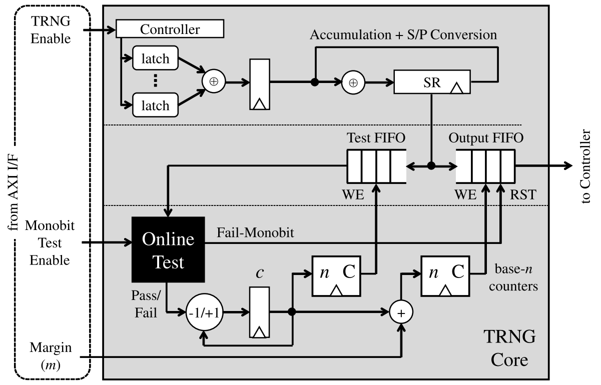


Fig. 11 Block diagram of the TRNG core with the proposed control method.

6. Proposed TRNG with On-Line Quality Control

6.1 Control Method

The basic strategy of the proposed on-line quality control method is that the number of accumulations for output random numbers is sufficiently larger than the maximum number of accumulations with which the on-line test fails. Figures 7 and 8 showed that the results of the individual test was similar to the overall results, but not the same. On the boundary between F and the others, the test result might go back and forth between them over time.

Figure 11 describes the block diagram of the TRNG core that adopts the proposed method. The proposed core is roughly separated into three parts. The upper part includes the latch-based TRNG, accumulation, and serial-to-parallel (S/P) conversion. Since the output of the TRNG is accumulated by word, the S/P conversion circuit is moved from the stream controller to the TRNG core. The center part has a pair of internal FIFOes. One of them is for output and the other is for the on-line test. The size of the output FIFO is larger than a block size of the short-term test and it does not send out data until it becomes full. In other words, the input of the short-term test remains unsent in the core until the test is passed. The bottom part includes the online test circuit and a FIFO control circuit according to the test result.

The accumulated word is valid only when a predefined number of accumulations are completed. This is done by two base-n counters that give write enable (WE) signals of the FIFOes. The number of accumulations for the test FIFO is stored in a register as c, while that for the output FIFO is the sum of $c + m$, where m is a margin given as a parameter of the core. When the count-the-ones test is completed, it assesses whether the result is within a certain range or not. If it is (i.e., passing the test), c is decremented; otherwise (i.e., failing at the test), c is incremented. When the short-term test is completed and the result is a fail, c is incremented and the output FIFO is reset, to avoid the failed random number being output. The value of c cannot be less than one. If c exceeds a predefined maximum number (31 in our evaluation), random number generation is terminated by considering that the latches cannot generate random bits any more. For comparison, the short-term test can be disabled by a parameter.

In this paper, the range of the result of each test to be considered as a pass is set to 2500 ± 336 in the count-the-ones test and 8000 ± 400 in the short-term test. The probability of occurrence of a false positive is 2.0×10^{-6} and 2.7×10^{-10}, respectively. Since a fail in the short-term test comes with a reset of the output FIFO, its significance level is set to be smaller.

6.2 Implementation of Proposed TRNG Core

The online test circuit is described as a C++ function and synthesized with the Xilinx Vitis HLS tool. An `HLS interface axis` pragma is added to each of the arguments, or the input data and the output result, to use AXI Stream. An `HLS interface ap_ctrl_none` pragma is given to the function itself to start the circuit automatically and repeatedly. These changes make it easy to be integrated with the other part of the core. The synthesized circuit is exported as Verilog files and instantiated from the core.

In the fixed-point arithmetic of the count-the-ones test, the lengths of the fractional portions of the *chsq* and *ev* values are set to 18 and 0, respectively. This pair of parameters have been referred to as the resource-saving candidate [17], which has about 1.5% of error in the *chsq* value on average but can be implemented with a minimal hardware. After the change of interfaces, the amount of hardware became slightly smaller: the required number of LUTs and flip-flops were 358 and 414, respectively.

The short-term test, described in Sect. 5.3, is included in the same circuit. It is conducted repeatedly while the number of occurrence of words is counted in the count-the-ones test. Since the lengths of input of the count-the-ones and the short-term tests are 256 kB and 2 kB, respectively, 128 results of the short-term tests are obtained during one iteration of the count-the-ones test. After the integration of the short-term test, the number of LUTs and flip-flops became 423 and 483, respectively. This meant the additional logic elements for the short-term test were 65 LUTs and 69 flip-flops.

7. Evaluation

7.1 Methodology

Our evaluation of the proposed method was conducted using three PYNQ–Z1 boards, which were different from one used in the preliminary evaluation. The TRNG core in the evaluation system, explained in Sect. 3, was replaced with the proposed core. Unlike the preliminary evaluation, the number of latches is fixed at logic synthesis and cannot be changed as a parameter opened to software. This means we synthesized a different hardware overlay for each number of latches. The version of the PYNQ system was 2.7, based on Ubuntu 20.04. Xilinx Vitis HLS 2021.1 and Xilinx Vivado 2020.2 were used for HLS and logic synthesis, respectively. The version of Vivado is slightly older than Vitis HLS because it is recommended by PYNQ 2.7.

We obtain 10^9 bits (1 Gbits) of random bitstring for each combination of the following parameters:

- the identifier of PYNQ board (ID): A, B, and C;
- the number of latches (l): 8, 16, 32, and 64;
- the margin of the number of accumulation(m): 1, 2, and 3; and
- the short-term test (Monobit): enable and disable.

If the bitstring is successfully obtained, it is divided into 1,000 10^6-bit substrings and assessed by the NIST 800–22 test suite [34]. The generation bit rate is also calculated and recorded from the elapsed time. If the random number generation has been terminated, or the number of accumulation has reached the upper limit, the event is recorded instead.

When the NIST test is conducted with many substrings, there are two criteria for determining pass or fail. One is the proportion of the number of substrings with p-values of 0.01 or more. If it is within a range of 99% ± 3σ, the test is considered as a pass. Its significance level is about 0.27%. The other is the p-value of a test on the flatness of the distribution of the obtained p-values. If this "p-value of p-values" is 0.0001 or more, the test is considered as a pass. The significance level of this criterion is 0.01%.

We summarize the overall result of the NIST suite by classifying it into three categories: PASS, *fail*, and **FAIL**. We mark it *fail* if the bitstring fails based on *either of* the criteria in any of the tests; we mark it **FAIL** if the bitstring fails based on **both of** the criteria.

7.2 Results on Random Number Generation

Table 4 summarizes the results on random number generation. Each row corresponds to the board identifier (ID) and the number of latches (l). Each column corresponds to the margin of accumulation (m) and the use of the short-term test (Monobit). The first line in each cell is the overall result of the NIST suite. The case that the random number generation is terminated is depicted as *Abort*. The second line represents the generation bit rate in Mbit/s.

If the short-term test is enabled, the bitstrings basically passed the test even with $m = 1$. If disabled, the bitstrings sometimes failed in the test with $m = 1$. Regarding the bit rate of generation, enabling the short-term test had almost the same effect as increasing m by one or two.

To examine the behavior with the short-term test in more detail, we monitored the time variation of the number of accumulation, or c. Figure 12 plots its result in the case of $l = 16$ and $m = 1$, using the board A. The X-axis represents the position of corresponding output bitstring. The value of c basically fluctuated around three or four but rapidly increased three times in the range of 600–800 MB. When a rapid decline of randomness quality is detected, the proposed method increases the value of c to a range of expected

Table 5 The number of logic elements used in the PL (FPGA) part of the evaluation system.

	Preliminary (l = 256)				Proposed (l = 16)			
	LUT	FF	BRAM	DSP	LUT	FF	BRAM	DSP
TRNG IP Core	780	1,071	1.0	0	784	906	5.5	2
AXI-Lite I/F	71	111	0.0	0	78	117	0.0	0
TRNG and Controller	676	902	0.0	0	631	669	4.5	2
Stream FIFO	64	58	1.0	0	68	58	1.0	0
AXI DMA	792	1,219	1.0	0	781	1,219	1.0	0
AXI Interconnects	860	1,041	0.0	0	849	1,026	0.0	0
Reset Generator	17	33	0.0	0	17	33	0.0	0
Total	2,448	3,364	2.0	0	2,430	3,184	6.5	2

Table 4 Summary of results on random number generation and the NIST test suite to generated bitstrings.

Monobit		Enable			Disable		
ID	l	$m = 1$	$m = 2$	$m = 3$	$m = 1$	$m = 2$	$m = 3$
A	8	*Abort* -	*Abort* -	*Abort* -	*Abort* -	*Abort* -	*Abort* -
	16	PASS 2.44	PASS 1.99	PASS 1.74	**FAIL** 2.94	PASS 2.54	*fail* 2.20
	32	PASS 2.29	PASS 1.93	PASS 1.66	PASS 3.15	PASS 2.48	PASS 1.93
	64	*fail* 3.89	PASS 2.92	PASS 2.31	PASS 5.02	PASS 3.45	PASS 2.75
B	8	PASS 1.70	*fail* 1.56	*fail* 1.40	**FAIL** 2.14	PASS 1.81	PASS 1.71
	16	PASS 1.91	PASS 1.67	*fail* 1.51	**FAIL** 2.69	PASS 2.35	PASS 1.92
	32	*fail* 2.13	PASS 1.84	*fail* 1.61	*fail* 3.34	*fail* 2.59	PASS 2.19
	64	PASS 4.10	PASS 3.09	PASS 2.47	PASS 5.67	*fail* 3.84	*fail* 2.96
C	8	PASS 1.01	PASS 0.94	PASS 0.99	**FAIL** 1.92	PASS 1.69	*fail* 1.37
	16	PASS 2.63	PASS 2.10	PASS 1.64	PASS 3.31	PASS 2.80	*fail* 2.17
	32	PASS 1.93	PASS 2.00	PASS 1.76	PASS 3.20	PASS 2.76	PASS 2.25
	64	PASS 3.94	PASS 3.02	*fail* 2.40	*fail* 5.19	PASS 3.78	PASS 2.85

Table 6 Comparison of the number of logic elements and the bit rate of generation among various TRNG cores.

Type		LUT	FF	Bit Rate
TC-TERO [21]		48	30	2.03
Configurable COSO [6]		69	33	1.37
Latch w/o accumulation [10]		676	902	12.50
Latch w/ accumulation [12]		69	95	0.83
This work	$l = 8$	613	645	1.01–1.70
	$l = 16$	631	669	1.91–2.63
	$l = 32$	664	717	1.93–2.29
	$l = 64$	735	813	3.89–4.10

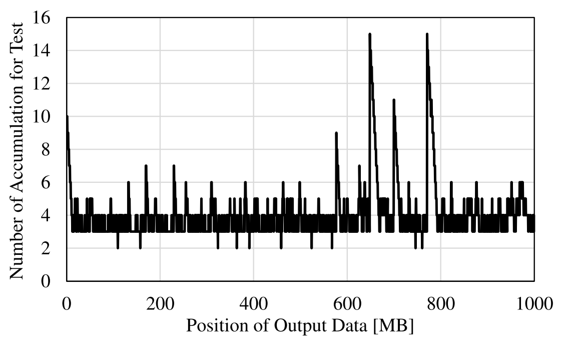


Fig. 12 Time variation of the number of accumulation when the short-term test is enabled.

safety, while discarding the output of the TRNG. After that, it decreases c gradually, confirming that the tested random numbers pass the count-the-ones test. If it could not follow a rapid decline of randomness quality, some of the random substrings would have a bias and exhibit a poor p-value. We think this is why some of the bitstrings failed in the NIST suite when the short-term test is disabled and $m = 1$.

However, a small decline of randomness quality might still be overlooked. There are a larger number of failed bitstrings based on the criterion of proportion only than expected. Since we conducted 15 kinds of tests for each of 66 random bitstrings, we expect two or three false positives ($66 \times 15 \times 0.27/100 \simeq 2.7$). The number of bitstrings marked $fail$ was much larger. Perhaps a very small number of substrings still have a bias and the proportion of substrings with p-values of 0.01 or more is slightly smaller than 99%. This result suggests that an adjustment of some parameters, such as the range of the result considered as a pass the implemented tests, is still needed, though we leave it for future work.

7.3 Hardware Resource Usage

Table 5 compares the number of logic elements — LUTs, flip-flops (FF), block RAMs (BRAM), and DSP units — in the FPGA part of the system, from the one used in the preliminary evaluation. The TRNG in the preliminary evaluation basically corresponds to the original latch-based TRNG [10] and up to 256 latches are available, whereas neither accumulation nor quality control features is supported. The case of 16 latches (l = 16) is shown in the table. The proposed system used a comparable number of LUTs and flip-flops, though additional 4.5 (36-kbit) block RAM and 2 DSPs were required. Only for the TRNG IP core, it only used less than 2% of available LUTs and flip-flops of PYNQ–Z1, a low-end FPGA SoC development board.

Table 6 enumerates various types of TRNGs implemented as TRNG cores of our evaluation system. The bit rate of generation is shown in Mbit/s. Note that existing TRNGs do not include hardware for quality control and assurance. Due to accumulation, the bit rate of generation with the proposed method decreased to an order of Mbit/s. However, it is larger than the case where the number of accumulation is fixed [12] and still comparable to other types of recently proposed TRNGs [6], [21]. The increase of the number of logic elements per latch was about 2.2 LUTs and 3 flip-flops, as estimated in the original latch-based TRNG [10].

To the best of our knowledge, there were no quality control methods for TRNGs that consider both short-term and long-term tendencies of the TRNG output. We have realized the first one with a minimal amount of hardware, by a careful design of randomness tests and a control method.

8. Conclusion

In this paper, we presented a design and an implementation of an on-line quality control method for the latch-based TRNG, in order to increase the throughput while keeping the quality of output random numbers. According to the evaluation with a prototype of the control system, the latch-based TRNG with the proposed method achieved 1.91–2.63 Mbit/s of throughput with 16 latches. We also confirmed that the method can follow the change of the quality of output random numbers quickly.

We are planning to conduct further experiments of, for example, seeing if the method is tolerant of changes of operating conditions or external attacks. We will also have to find causes of a rapid decline of randomness quality of the latch-based TRNG and a way to fundamentally solve the problem. Further improvements on reliability and throughput to our method may be achieved by examining new on-the-fly tests. It is also left for future studies to apply our approach to other types of TRNGs, such as TERO and COSO.

Acknowledgments

A part of this study was supported by JSPS Grants-in-Aid for Scientific Research (KAKENHI) Grant Number 21K12164 and 20K11733.

References

[1] Ç. K. Koç, ed., Cryptographic Engineering, Springer, Berlin, 2009.

[2] O. Petura, U. Mureddu, N. Bochard, V. Fischer, and L. Bossuet, "A survey of AIS-20/31 compliant TRNG cores suitable for FPGA devices," Proc. 26th International Conference on Field Programmable Logic and Applications, Lausanne, Switzerland, pp.1–10, 2016.

[3] B. Sunar, "True random number generators for cryptography," Cryptographic Engineering, ed. Ç.K. Koç, Berlin, pp.55–73, Springer, 2009.

[4] A.T. Markettos and S.W. Moore, "The Frequency Injection Attack on Ring-Oscillator-Based True Random Number Generators," Proc. Workshop on Cryptographic Hardware and Embedded Systems 2009, Lausanne, Switzerland, vol.5747, pp.317–331, 2009.

[5] B. Yang, V. Rožić, N. Mentens, W. Dehaene, and I. Verbauwhede, "TOTAL: TRNG on-the-fly testing for attack detection using lightweight hardware," Proc. 2016 Design, Automation & Test in Europe Conference & Exhibition, Dresden, Germany, pp.127–132, 2016.

[6] A. Peetarmans, V. Rožić, and I. Verbauwhede, "A highly-portable true random number generator based on coherent sampling," Proc. 29th International Conference on Field Programmable Logic and Applications, Barcelona, Spain, pp.218–224, 2019.

[7] L.B. Carreira, P. Danielson, A.A. Rahimi, M. Luppe, and S. Gupta, "Low-Latency Reconfigurable Entropy Digital True Random Number Generator With Bias Detection and Correction," IEEE Trans. Circ. Syst. I, vol.67, no.5, pp.1562–1575, 2020.

[8] T. Chen, Y. Ma, J. Lin, Y. Cao, N. Lv, and J. Jing, "A lightweight full entropy TRNG with on-chip entropy assurance," IEEE Trans. Comput. Aided Des. Integrated Circ. Syst., vol.40, no.12, pp.2431–2444, 2021.

[9] H.M. Gonzalez, E.S.M. Heredia, and L.E. Arrontes, "Dynamic control of entropy and power consumption in TRNGs for IoT applications," IEEE Electron. Express, vol.15, no.2, pp.20171157:1–20171157:11, 2018.

[10] N. Fujieda and S. Ichikawa, "A latch-latch composition of metastability-based true random number generator for Xilinx FPGAs," IEICE Electron. Express, vol.15, no.10, pp.20180386:1–20180386:12, 2018.

[11] H. Hata and S. Ichikawa, "FPGA implementation of metastability-based true random number generator," IEICE Trans. Info. Syst., vol.E95-D, no.2, pp.426–436, 2012.

[12] N. Fujieda, H. Kishibe, and S. Ichikawa, "A light-weight implementation of latch-based true random number generator," Proc. 15th International Wireless Communication and Mobile Computing Conference, Tangier, Morocco, pp.901–906, 2019.

[13] R. Santoro, A. Tisserand, O. Sentieys, and S. Roy, "Arithmetic operators for on-the-fly evaluation of TRNGs," Proc. SPIE 7444, Mathematics for Signal and Information Processing, vol.7444, no.74440S, San Diego, CA, pp.1–12, 2009.

[14] A. Vaskova, C. López-Ongil, E.S. Millán, A. Jiménez-Horas, and L. Entrena, "Accelerating secure circuit design with hardware implementation of Diehard Battery of tests of randomness," Proc. IEEE 17th International On-Line Testing Symposium, Athens, Greece, pp.179–181, 2011.

[15] Xilinx Inc., "PYNQ: Python productivity." http://www.pynq.io/, accessed Nov. 7, 2022.

[16] H. Kishibe, S. Ichikawa, and N. Fujieda, "An investigation of latch-based lightweight TRNG," IEEJ Technical Meeting on Innovative Industrial System, no.IIS–21–012, pp.1–6, 2021 (in Japanese).

[17] R. Oya, N. Fujieda, and S. Ichikawa, "An HLS implementation of on-the-fly randomness test for TRNGs," Proc. 10th International Symposium on Computing and Networking, Himeji, Japan, pp.151–157, 2022.

[18] L.M. Reyneri, D. Del Corso, and B. Sacco, "Oscillatory metastability in homogeneous and inhomogeneous flip-flops," IEEE J. Solid-State Circ., vol.25, no.1, pp.254–264, 1990.

[19] M. Varchola and M. Drutarovský, "New high entropy element for FPGA based true random number generators," Proc. Workshop on Cryptographic Hardware and Embedded Systems 2010, Santa Barbara, CA, vol.6225, pp.351–365, 2010.

[20] K. Yang, D. Blaauw, and D. Sylvester, "An all-digital edge racing true random number generator robust against PVT variations," IEEE J. Solid-State Circuits, vol.51, no.4, pp.1022–1031, 2016.

[21] N. Fujieda, "On the feasibility of TERO-based true random number generator on Xilinx FPGAs," Proc. 30th International Conference on Field Programmable Logic and Applications, Göthenburg, Sweden, pp.103–108, 2020.

[22] M. Varchola and M. Drutarovský, "Embedded Platform for Automatic Testing and Optimizing of FPGA Based Cryptographic True Random Number Generators," Radioengineering, vol.18, no.4, pp.631–638, 2009.

[23] F. Veljković, V. Rožić, and I. Verbauwhede, "Low-cost implementations of on-the-fly tests for random number generators," Proc. 2012 Design, Automation & Test in Europe Conference & Exhibition, Dresden, Germany, pp.959–964, 2012.

[24] V.B. Suresh, D. Antonioli, and W.P. Burleson, "On-chip lightweight implementation of reduced NIST randomness test suite," Proc. 2013 IEEE International Symposium on Hardware-Oriented Security and Trust, Austin, TX, pp.93–98, 2013.

[25] B. Yang, V. Rožić, N. Mentens, W. Dehaene, and I. Verbauwhede, "Embedded HW/SW platform for on-the-fly testing of true random number generators," Proc. 2015 Design, Automation & Test in Europe Conference & Exhibition, Grenoble, France, pp.345–350, 2015.

[26] B. Yang, V. Rožić, N. Mentens, and I. Verbauwhede, "On-the-fly tests for non-ideal true random number generators," Proc. 2015 IEEE International Symposium on Circuits and Systems, Lisbon, Portugal, pp.2017–2020, 2015.

[27] Y. Cao, V. Rožić, B. Yang, J. Balasch, and I. Verbauwhede, "Exploring active manipulation attacks on the TERO random number generator," Proc. IEEE 59th International Midwest Symposium on Circuits and Systems, Abu Dhabi, UAE, pp.1–4, 2016.

[28] S.U. Hussain, M. Majzoobi, and F. Koushanfar, "A built-in-self-test scheme for online evaluation of physical unclonable functions and true random number generators," IEEE Trans. Multi-Scale Comput. Syst., vol.2, no.1, pp.2–16, 2016.

[29] M. Grujić, V. Rožić, B. Yang, and I. Verbauwhede, "Lightweight prediction-based tests for on-line min-entropy estimation," IEEE Trans. Embed. Syst. Lett., vol.9, no.2, pp.45–48, 2017.

[30] H. Martin, G. Di Natale, and L. Entrena, "Towards a dependable true random number generator with self-repair capabilities," IEEE Trans. Circ. Syst. I, vol.65, no.1, pp.247–256, 2018.

[31] L. Gantel, A. Duc, L. Steiner, F. Vannel, A. Upegui, and F. Gluck, "A FPGA-based post-processing and validation platform for random number generators," 2020 IEEE International Parallel and Distributed Processing Symposium Workshops, New Orleans, LA, pp.123–126, 2020.

[32] W. Killmann and W. Schindler, A proposal for: functionality classes for random number generators, version 2.0, Federal Office for Information Security, Bonn, Germany, 2011.

[33] National Institute of Standard Technology, Gaithersburg, MD, FIPS PUB 140–2 Security Requirements for Cryptographic Modules, 2001.

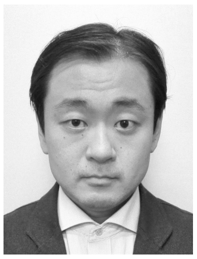
[34] A. Rukhin, J. Sota, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo, A statistical test suite for random and pseudorandom number generators for cryptographic applications, 2000.

[35] M.S. Turan, E. Barker, J. Kelsey, K. McKay, M. Baish, and M. Boyle, "Recommendation for the entropy sources used for random bit generation," National Institute of Standard Technology (NIST) Special Publication 800–90B, Gaithersburg, MD, 2018.

[36] G. Marsaglia, "Diehard battery of tests of randomness (archived)." https://web.archive.org/web/20160125103112/http://stat.fsu.edu/pub/diehard/, accessed Nov. 7, 2022.

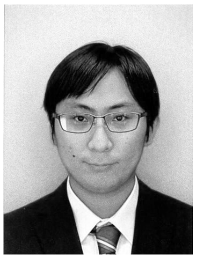
[37] N. Fujieda, "Versatile TRNG IP core for evaluation on PYNQ platform." https://github.com/nfproc/TRNG_IP, accessed Nov. 7, 2022.

[38] R.G. Brown, D. Eddelbuettel, and D. Bauer, "Dieharder: a random number test suite version 3.31.1."



Naoki Fujieda received his D.E. degree in 2013 from the Department of Computer Science of Tokyo Institute of Technology. He worked for Toyohashi University of Technology during 2013–2019. He is presently an associate professor of Aichi Institute of Technology. His research interests include processor architecture, applied FPGA systems, embedded systems, and digital systems education. He is a member of IPSJ, IEICE, and IEEE.

Shuichi Ichikawa received his D.S. degree in Information Science from the University of Tokyo in 1991. He has been affiliated with Mitsubishi Electric Corporation (1991–1994), Nagoya University (1994–1996), Toyohashi University of Technology (1997–2011), and Numazu College of Technology (2011–2012). Since 2012, he is a professor of the Department of Electrical and Electronic Information Engineering of Toyohashi University of Technology. His research interests include parallel processing, high-performance computing, custom computing machinery, and information security. He is a member of IEEE (senior member), ACM, IEICE (senior member), IEEJ (senior member), and IPSJ.



Ryusei Oya received his B.E. degree in 2022 from the Department of Electrical and Electronics Engineering of Aichi Institute of Technology.

Hitomi Kishibe received her M.E. degree in 2021 from the Department of Electrical and Electronic Information Engineering of Toyohashi University of Technology.