A light-weight implementation of latch-based true random number generator

Naoki Fujieda
Department of Electrical and Electronics Engineering,
Faculty of Engineering,
Aichi Institute of Technology,
Toyota, Aichi, Japan
nfujieda@aitech.ac.jp

Hitomi Kishibe and Shuichi Ichikawa
Department of Electrical and Electronic
Information Engineering,
Toyohashi University of Technology,
Toyohashi, Aichi, Japan
kishibe@ccs.ee.tut.ac.jp (Hitomi Kishibe),
ichikawa@ieee.org (Shuichi Ichikawa)

***Abstract*—A TRNG (True Random Number Generator) is an important component for security in mobile computing and IoT. This paper presents an light-weight FPGA implementation of a previously proposed latch-based TRNG, while keeping the quality of generated random numbers. By accumulating the generated random numbers fifteen times with an XOR operation for each output word, the proposed TRNG with 16 latches passed the NIST SP 800-22 test suite, whereas the original TRNG required 248 latches. An important finding of this work is that the quality of random sequence is greatly improved by XOR-ing temporally interleaved series of bits.**

***Index Terms*—Random number generation, Field programmable gate arrays, System-on-chip**

I. Introduction

A true random number generator (TRNG) utilizes unpredictable physical phenomena to produce random numbers. It is often used for security applications such as obtaining a cryptographic key in an unpredictable way. In mobile computing and IoT (Internet of Things) applications where the resource is strongly limited, the hardware cost of TRNG is a critical factor in addition to the quality of generated random numbers.

One of the phenomena adopted in TRNGs is the metastability of a latch or a flip-flop [2], [6], [7], [12]. The authors previously proposed a latch-latch composition of a TRNG [6], shown in Fig. 1, which is an improved version of an RS-latch TRNG based on FPGAs (Field Programmable Gate Arrays) [7]. Each latch in [6] is implemented as a soft macro, which is more suitable for the latest design tool of Xilinx FPGAs. In the previous work [6], the latch-latch TRNG circuit was evaluated as a stand-alone circuit with a FIFO (First-In First-Out) buffer and a serial controller. It passed the diehard test [10] with 224 latches and the NIST SP 800-22 test suite [11] with 320 latches. Though it was smaller than a soft macro implementation of the past RS-latch TRNG [7], it was still much larger than the result of the hard macro implementation on an older FPGA [7].

In this paper, we propose a method to reduce the required amount of hardware of the latch-based TRNG while keeping the quality of output random numbers. The output word of the TRNG is accumulated by an exclusive OR (XOR) operation and harvested at certain minimum intervals. The point is that the accumulation is performed by word, instead of bit. An inportant finding of this study is that the quality of random numbers can be greatly improved by this by-word accumulation when temporally adjacent output bits are correlated. To prove this through detailed evaluation, the TRNG is embedded as a peripheral of an FPGA SoC (System on a Chip), which runs a Linux distribution called Xillinux [14]. There, the statistical properties of generated random bitstrings can be checked immediately in the processing system of the SoC.

II. Post-processing of TRNG output

Various post-processing methods for TRNGs have been proposed to improve the statistical properties of their output. This is because TRNGs may have a bias, or an imbalance of appearance frequency of zero and one. A trivial solution to eliminate the bias is to apply a cryptographic hash function to the output, however it is left out of the discussion because it requires an excessively large amount of hardware. Post-processing methods with minimal hardware are categorized into two classes: entropy compression and scrambling.

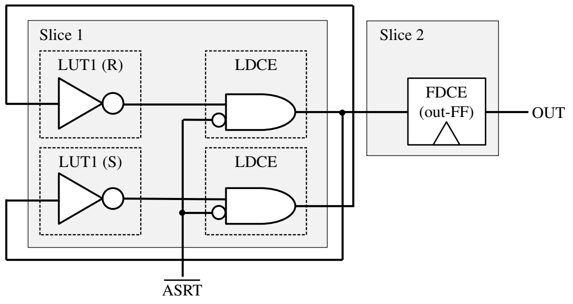

Fig. 1. A latch-latch composition of an RS latch-based TRNG [6].

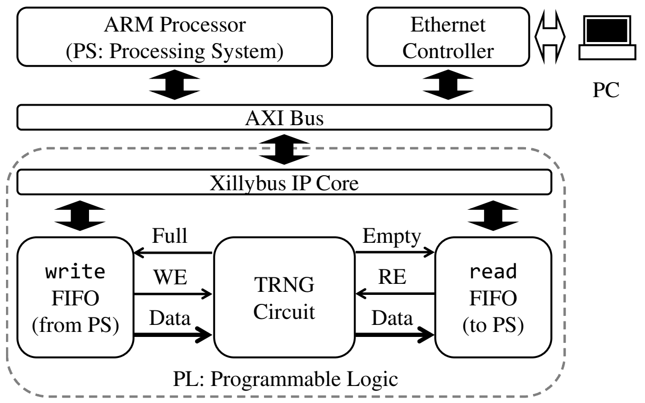

Fig. 2. Block diagram of the evaluation system.

Entropy compression transforms an n-bit random bitstring with a bias into an m-bit random bitstring with a smaller (possibly zero) bias. There are two well-known compression methods: the von Neumann corrector [13] and the XOR corrector [4]. The von Neumann corrector outputs ‘0’ or ‘1’ when two successive input bits are “01” or “10,” respectively. Input bits of “00” and “11” are simply discarded. If the input bits are independent of each other, the bias will be entirely eliminated. One of the shortcomings of the von Neumann corrector is its poor generation rate m/n: it is only 25% at most. To balance the reduction of the bias and the generation rate, methods using an (n, m) BCH (Bose–Chaudhuri–Hocquenghem) code were also studied [8]. The XOR corrector performs XOR operation to multiple biased bits to generate a single bit with a smaller bias. Our previous work [6] adopted it spatially: output bits generated at the same time by hundreds of latches were XOR-ed to obtain a single TRNG output bit.

Scrambling is a method to generate an unbiased bitstring of the same length as the input by a feedback of the output with a shift register. It is often used to post-process TRNG output [1], [9]. In general, an output bit of a TRNG is XOR-ed by one or more internal bit(s) of the shift register and fed as its input. In other words, it mixes the output of TRNG with the input of an LFSR (Linear Feedback Shift Register). When a sufficient amount of TRNG output is given, the bias will converge to zero [1].

Note that the scrambling only improves the statistical properties of the output random bitstring and the predictability of the output bit due to the bias cannot be reduced [5]. If the post-processing algorithm and the output bitstrings are known, back calculation of the TRNG output might be possible. Once the bias of the TRNG output is revealed, the next output bit might be predictable with a probability depending on the bias.

Our method accumulates a random word generated by a TRNG to the previous output word once or multiple times. If the method accumulates a word only once for each output word, it will correspond to scrambling. On the other hand, if the accumulation performs multiple times, it will be categorized into entropy compression as it requires multiple words to obtain a single output word.

III. System description

A. Overall system

To evaluate the latch-latch TRNG in detail, we built an evaluation system based on a demo configuration of the Xillybus boot partition kit [14]. Xillybus is an FPGA IP core that provides a simple FIFO-based interface between a user logic and a CPU core, along with its software driver. The software driver is available on Xillinux, a Linux distribution for FPGA SoCs based on Ubuntu LTS. Its boot partition kit includes the HDL (Hardware Description Language) source code of a demo configuration of Xillybus. The versions of the boot partition kit and Xillinux we chose were 2.0a and 1.3, respectively.

Figure 2 depicts the block diagram of the evaluation system. We use it as a standalone Linux server where a user can login via `ssh`. In general, an FPGA SoC has the processing system part and the programmable logic part, which communicate each other via the AXI bus. In Xillybus, a user circuit is connected to input and output FIFOes and implemented on the programmable logic part. We replaced a sample loopback circuit in the demo configuration with our TRNG circuit, described in Section III.B.

From a software point of view, these FIFOes are abstracted to device files `/dev/xillybus_read_32` and `/dev/xillybus_write_32`. To obtain random numbers from the TRNG peripheral, a 32-bit parameter of the circuit is first written via the `write` device file, which is valid until the file is closed. After that, the generated random numbers are sequentially readable via the `read` device file. Source codes of statistic test suites are slightly modified to deal with these device files, which means that the statistical properties of the random bitstrings generated by the TRNG can be evaluated immediately in the processing system part of FPGA SoC. In this manner, the evaluation system can be used as a practical embedded Linux system, which runs any application that requires a TRNG.

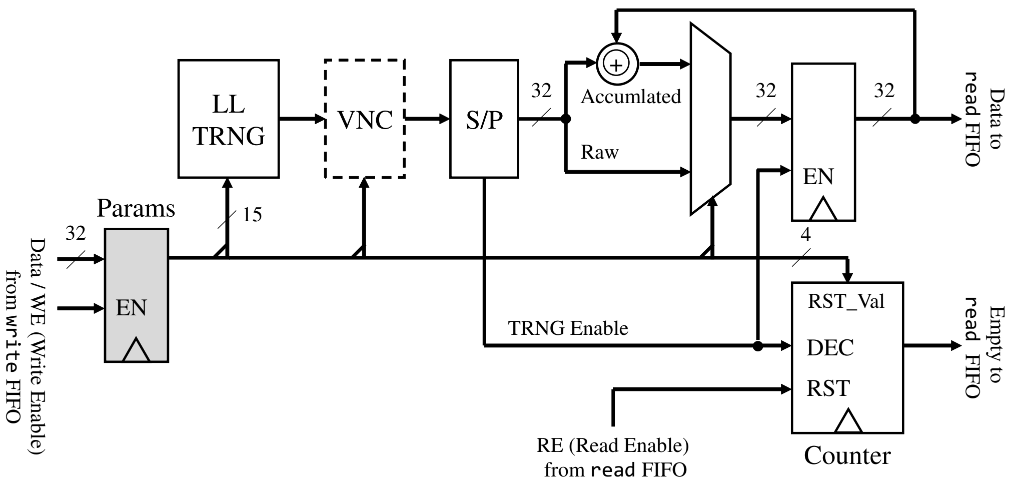

Fig. 3. Block diagram of the proposed TRNG circuit.

B. Proposed TRNG circuit

Figure 3 depicts the block diagram of the proposed TRNG circuit, connected to Xillybus FIFOes as a user logic. There are two major differences from the evaluation system in the previous work [6]. First, the output of the latch-latch TRNG (LL TRNG), converted from serial to parallel (S/P), is sent to the `read` FIFO of Xillybus. Second, the circuit can generate a post-processed output (*Accumulated* in Fig. 3) by accumulating the *Raw* output with an XOR operation. An additional down-counter circuit controls how many times the output word must be accumulated at least. Post-processing of the serial output of the TRNG by the von Neumann corrector (VNC) will also be presented for comparison.

Note that this accumulation is performed by a 32-bit word, instead of a single bit. If the accumulation width was one bit, the derived circuit would be simply a serial implementation of the XOR corrector, where each output bit would be XOR-ed to the previous bit. The accumulation width of 32 bits means that each bit is XOR-ed to the 32nd bit from the last. This temporal interval greatly improves the quality of derived random numbers, as shown in Section IV.B. It also enables software to trade the quality of random numbers for their generation rate. If it reads the `read` device file slowly, the number of times of accumulation becomes large and the quality of random numbers might be better. One of the possible shortcomings of this accumulation is that it requires 32 XOR gates. It might affect the amount of hardware of the TRNG circuit, which will be evaluated in Section IV.D.

C. Parameterization of TRNG

To evaluate the TRNG with various numbers of latches without resynthesis of the system, we implemented the latch-latch TRNG component as shown in Fig. 4. Eight RS latches are gathered into a latch set and their outputs are XOR-ed to obtain the output of the set. The outputs of up to thirty-two sets are then XOR-ed again to obtain the TRNG output with 8, 16, ..., 256 latches. The TRNG component includes three individual TRNGs. The output of the selected TRNG is supplied to the S/P converter (or VNC).

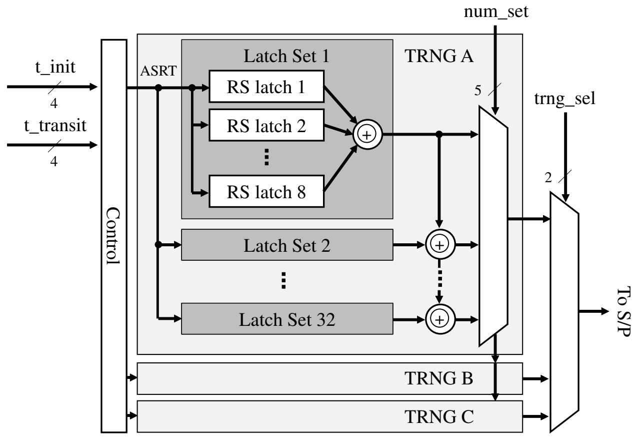

Fig. 4. A Latch-latch TRNG with a variable number of latches.

A 32-bit integer from the processing system via the `write` FIFO (Params in Fig. 3) configures the following parameters of the TRNG and its post-processing method:

- **t_init and t_transit** (4 bits each) determines the cycle time of the TRNG;
- **num_set** (5 bits) selects the number of sets to obtain the TRNG output;
- **trng_sel** (2 bits) selects one of the three individual TRNGs;
- **use_vnc** (1 bit) determines if the VNC is used for post-processing; and
- **num_accum** (4 bits) sets the minimum number of times of accumulation.

When the num_accum parameter is set to zero, accumulation is disabled, and the *Raw* output is selected as TRNG output.

TABLE I
THROUGHPUT OF THE TRNG CIRCUITS.

TRNG Variant	Throughput [Mbit/s]
Raw	20.000
Accumulated-1	20.000
Accumulated-2	10.000
Accumulated-4	5.000
Accumulated-15	1.333
VNC	4.962

The remaining 12 bits are reserved.

IV. EVALUATION

A. Methodology

We conducted both the diehard test [10] and the NIST SP 800–22 test suite [11] on the evaluation system, which was implemented on an Avnet ZedBoard that includes a Zynq-7000 XC7Z020-CLG484-1 FPGA SoC. The logic circuits are synthesized and implemented by Vivado 2017.3 with default parameters. The diehard test [10] is used to briefly check the quality of random numbers. It consists of 18 types of tests and requires about 100 Mbits of random bitstring. It takes about one minute to complete the tests. The NIST SP 800–22 test suite [11] is used for an advanced check, which requires 1 Gbit of random bitstring and about five hours of computation. We adopt the same criteria as the previous work [6] for passage of the tests.

Evaluated TRNG variants are categorized into three groups: Raw, Accumulated, and VNC. The labels *Raw* and *Accumulated* correspond to the output shown in Fig. 3. The number $n \, (n = 1, 2, ..., 15)$ after the label *Accumulated* stands for the number of times the output must be accumulated before being sent out. For example, in the TRNG variant *Accumulated-2*, the output word will be accumulated at least twice. We also evaluate the use of von Neumann corrector (*VNC*) [13]. The cycle time of the TRNG is set to 50 ns: the shortest possible time that keeps the quality of TRNGs according to the previous work [6].

Table I summarizes the throughput of the evaluated TRNG circuits with 128 latches, measured from the processing system by reading 256 bytes in a single read. It was exactly equivalent to the theoretical limit calculated from the cycle time in the *Raw* and *Accumulated* variants, which meant that Xillybus and other interconnections did not become a bottleneck of the system. The throughput with *VNC* was slightly lower than the theoretical limit, which came from a bias of the *Raw* output.

B. Effect of by-word accumulation

First, we conducted a simulation on the *Raw* bitstrings to compare the accumulation methods. We collected the *Raw* output of 400 Mbits for each of three TRNGs, as shown in Fig. 4. They were post-processed by each of the by-word accumulation (i.e. the proposed circuit) and the by-bit accumulation (i.e. serial implementation of XOR corrector). The width of

TABLE II
THE NUMBER OF FAILURES IN THE DIEHARD TEST OF BITSTRINGS DERIVED BY SIMULATED ACCUMULATION.

#latch	Accum. by	Raw	Accumulated $n=1$	Accumulated $n=2$	Accumulated $n=4$
64	Word	37	31	16	0
	Bit		35	32	7
128	Word	15	10	0	0
	Bit		10	9	0

a word in the by-word accumulation is 32 bits. The derived random bitstrings, along with the *Raw* bitstrings, are evaluated by the diehard test. The point is that the same *Raw* bitstrings are given to both accumulation methods. The effect of the accumulation method, rather than the effect of variation of the quality of the TRNGs themselves, is emphasized by this simulation. The number of latches is set to 64 or 128, while the number of times of accumulation, n, is set to 1, 2, or 4.

Table II shows the number of failed tests (out of 54) for each set of bitstrings derived by the simulated accumulation from the same *Raw* bitstrings. The rows Word and Bit stand for the by-word and by-bit accumulation methods, respectively. The by-word accumulation almost always gave fewer failures than the by-bit accumulation. In particular, it passed all the tests only with the by-word accumulation in two cases: 64 latches with $n = 2$ and 128 latches with $n = 4$.

This result implies that a bitstring generated by a latch-based TRNG has the property that its temporally adjacent bits are somewhat correlated and the correlation ceases as the time interval increases. A simple XOR corrector may generate a poor output sequence because the XOR corrector supposes that its input bits are independent of each other. Therefore, the proposed by-word accumulation, shown in Fig. 3, is a reasonable design choice.

C. Result on the quality of random numbers

Figure 5 plots the evaluation results with the diehard test for each number of latches. The x-axis represents the number of latches, while the y-axis represents the number of failed tests out of 54 (18 types of tests $\times$ 3 TRNGs). Note that the throughput of the *Raw* variant is equivalent to the *Accumulated-1* variant and the *VNC* variant has almost the same throughput as *Accumulated-4*, as shown in Table I. When comparing these pairs in the number of failed tests, the *Accumulated* variants showed the better results especially when the number of latches are relatively large.

Figure 5 also showed that sometimes one of the tests failed even though the number of latches was large enough (e.g. *Accumulated-1* with 224 latches). In all the cases, a failure occurred in either the OPERM5 or the overlapping sums test. It has been reported that these two tests might have bugs, which result in a higher possibility of giving false positives [3]. Therefore, we exclude these tests from the subsequent discussions.

Figure 6 summarizes the result of the diehard test of all the TRNG variants. The y-axis, the number of latches to pass the

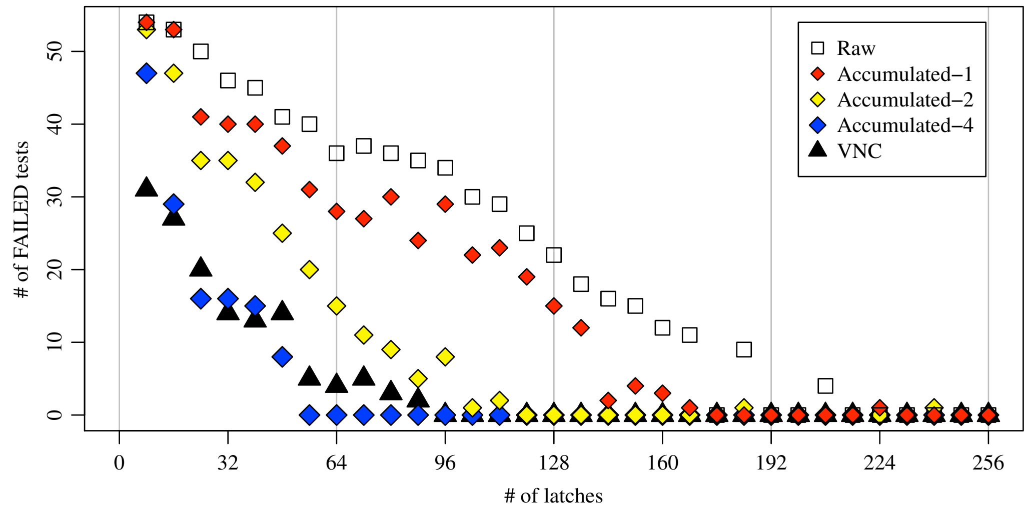

Fig. 5. The change of the number of failures in the diehard test with the number of latches.

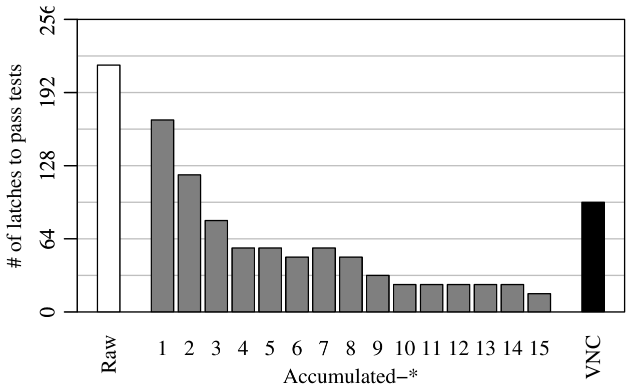

Fig. 6. The number of latches to pass the diehard test.

TABLE III
THE RESULTS OF THE NIST TEST SUITE.

TRNG	#Latch	Result
Raw	216	FAILED (CumulativeSums)
	240	FAILED (CumulativeSums)
	248	PASSED
Accumulated-1	168	FAILED (BlockFrequency)
	192	FAILED (Universal)
	200	PASSED
Accumulated-15	**16**	PASSED
	24	PASSED
	32	PASSED

test, is obtained by the maximum number of latches that failed in at least one test (except OPERM5 and overlapping sums) plus 8. The *Raw* variant required 216 latches and even with the *VNC* it required 96 latches. On the other hand, as also shown in Fig. 5, it reduced to 168 and 56 latches by *Accumulated-1* and *Accumulated-4*, respectively. It continued to reduce as the minimum number of times of accumulation increased. At last, by accumulating the output fifteen times, the TRNG with only 16 latches passed the diehard test.

Finally, the *Raw*, *Accumulated-1*, and *Accumulated-15* variants were tested by the NIST test suite. Only one of the three TRNGs (TRNG A in Fig. 4) was evaluated. Table III shows the summary of the results. In the *Raw* and *Accumulated-1* variants with the number of latches that passed the diehard test (i.e. 216 and 168 latches, respectively), the TRNG failed one of the tests as its acceptance rate (97.8% in the both cases) was slightly lower than the threshold (98.0%). The required numbers of latches to pass the NIST test suite were 248 and 200, respectively. The *Accumulated-15* variant passed the NIST test suite with 16 latches.

D. Result on the amount of hardware

Finally, the TRNG circuit shown in Fig. 3 is separately synthesized and implemented to evaluate the amount of hardware. The numbers of LUTs (Look-Up Tables) and register elements (i.e. flip-flops and latches) of the circuit after implementation are recorded. To evaluate only one TRNG (TRNG A in Fig. 4), other TRNGs (TRNG B and C) are explicitly removed. Also, with a fewer number of latches, redundant sets of latches are explicitly removed. The TRNG variants that passed the NIST test suite in Table III are evaluated.

Table IV summarizes the amount of hardware, along with area-delay product, when implementing the TRNG circuit on a Zynq-7000 FPGA SoC. We estimated the amount of hardware of the latch-latch TRNG as 2.2 LUTs and 3 registers per latch in the previous work [6]. This estimation well explains the result. The numbers of additional LUTs to the estimation were 12 $(= 558 - 2.2 \times 248)$ in the *Raw* variant and 44 $(= 484 - 2.2 \times 200)$ in the *Accumulated* variants. The extra 32 LUTs were likely to be used in the accumulation circuit.

To compare the amount of hardware in a single measure, we calculated LUT equivalent as $L + R/2$, where L and R are the

TABLE IV
COMPARISON OF AREA-DELAY PRODUCT. ACCUMULATED VARIANTS ARE ABBREVIATED AS ACCUM-n.

	Raw 248 latches	Accum-1 200 latches	Accum-15 16 latches
# of LUTs	558	484	80
# of Registers	828	683	126
LUT equivalent	972	825.5	143
Cycle Time [ns]	50	50	750
Area-delay product [x10^3]	48.6	41.3	107.3

numbers of LUTs and registers, respectively. This definition is based on the fact that a slice in Zynq-7000 FPGA SoC has four LUTs and eight registers.

The *Accumulated-1* variant reduced LUT equivalent and area-delay product by 15.1%. Although it was considered as a positive result, it should be noted that this variant is considered as scrambling, as we explained in Section II. Scrambling does not improve the unpredictability of the output bit. Therefore, this variant will be preferred when fast generation of random numbers is required and only the statistical properties matter.

The LUT equivalent of the *Accumulated-15* variant was 6.8 times smaller than the *Raw* variant, while its area-delay product was 2.2 times larger because of additional circuits such as the S/P converter. By compressing entropy in an effective way, the required hardware was greatly reduced. As a result, the proposed TRNG circuit became small enough and consumed less than 0.2% of logic elements available in the XC7Z020 FPGA SoC (53,200 LUTs and 106,400 registers).

Although accumulating output more than 15 times might further reduce the number of required latches, it should be noted that it also increases the risk of picking "useless" latches, which generate constant (or almost constant) output. In our previous work [6], the proportion of latches with constant output was about 2/3. We also confirmed that, through a preliminary evaluation with the evaluated Zynq-7000 FPGA SoC, 40.4% of the latches had constant or almost constant (with 99.99% or more probabilities) output. As a result, extremely poor TRNG will be made with a probability of p^l, where p is the proportion of such latches and l is the number of latches in the TRNG. It will rapidly increase with the extremely small number of latches. An extensive evaluation, probably with multiple FPGA SoC chips, is required to evaluate such a risk, though we leave this as future work.

V. CONCLUSION

We have proposed a new light-weight latch-latch composition for RS latch-based TRNGs. This paper presented an accumulation method that further reduced the number of required latches. The TRNG circuit with the proposed method reduced the number of latches for passing the NIST test suite from 248 to 16. As a result, the required hardware elements of FPGA became 80 LUTs and 126 registers, which were small enough to be adopted in mobile computing and IoT applications. It can also offer a trade-off between the generation rate and the quality of random numbers.

Our future work includes a detailed evaluation, especially with multiple FPGA devices, to show the availability of the latch-latch TRNG and the proposed accumulation method. Sensitivity studies with regard to operational conditions, such as temperature and supply voltage, are also important for the practicality.

ACKNOWLEDGMENT

A part of this work was supported by JSPS Grants-in-Aid for Scientific Research (KAKENHI) Grant Number 16K00072.

REFERENCES

[1] T. Addabbo, M. Alioto, A. Fort, S. Rocchi, and V. Vignoli, "Efficient post-processing module for a chaos-based random bit generator," in *Proc. 13th IEEE Int'l. Conf. on Electron., Circ. and Syst.*, 2006, pp. 1224–1227.

[2] M. J. Bellido, A. J. Acosta, M. Valencia, A. Barriga, and J. L. Huertas, "Simple binary random number generator," *Electron. Lett.*, vol. 28, no. 7, pp. 617–618, 1992.

[3] R. G. Brown, D. Eddelbuettel, and D. Bauer. (2018) Dieharder: a random number test suite version 3.31.1. [Online]. Available: https://webhome.phy.duke.edu/~rgb/General/dieharder.php

[4] R. B. Davies. (2002) Exclusive OR (XOR) and hardware random number generators. [Online]. Available: http://www.robertnz.net/pdf/xor2.pdf

[5] M. Dichtl, "Bad and good ways of post-processing biased physical random numbers," in *Proc. 14th Annual Fast Software Encryption Workshop*, 2007, pp. 137–152.

[6] 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. 20 180 386:1–20 180 386:12, 2018.

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

[8] S.-H. Kwok, Y.-L. Ee, G. Chew, K. Zheng, K. Khoo, and C.-H. Tan, "A Comparison of Post-Processing Techniques for Biased Random Number Generators," in *Proc. 5th Workshop in Inf. Secur. Theory and Pract.*, 2011, pp. 175–190.

[9] W. Mao, Y. Li, C.-H. Heng, and Y. Lian, "Zero-bias true random number generator using LFSR-based scrambler," in *Proc. 2017 IEEE Int'l. Symp. on Circ. and Syst.*, 2017, pp. 1–4.

[10] G. Marsaglia. Diehard battery of trests of randomness (mirror). [Online]. Available: https://github.com/reubenhwk/diehard

[11] A. Rukhin, J. Soto, 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*, NIST Special Publication 800–22, Rev. 1a, 2010.

[12] N. Torii, D. Yamamoto, and T. Matsumoto, "Evaluation of latch-based physical random number generator implementation on 40 nm ASICs," in *Proc. 6th Int'l. Workshop on Trustworthy Embedded Devices*, 2016, pp. 23–30.

[13] J. von Neumann, "Various techniques used in connection with random digits," *Monte Carlo Method, National Bureau of Standards Applied Mathematics Series 12*, pp. 36–38, 1951.

[14] Xillybus Ltd. Xillybus IP cores and design services. [Online]. Available: http://xillybus.com/