International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 11, Number 2, pages 154–171, July 2021

An MMCM-based high-speed true random number generator for Xilinx FPGA

Naoki Fujieda and Sogo Takashima
Department of Electrical and Electronics Engineering, Faculty of Engineering,
Aichi Institute of Technology, Toyota, Aichi, 470-0392, Japan

Received: February 5, 2021
Revised: April 19, 2021
Accepted: June 1, 2021
Communicated by Shinya Takamaeda-Yamazaki

Abstract

For a true random number generator (TRNG) on an FPGA, the use of a pair of clocking elements has an advantage of minimal usage of its logic elements. This paper presents a novel high-speed TRNG for recent Xilinx FPGAs using their clocking elements called mixed-mode clock managers (MMCMs). By following the proposed parameter selection methods, both better randomness and higher throughput of generated bitstrings can be achieved. According to our evaluation on an Artix-7 FPGA with the most promising sets of parameters, 38.2% (42 out of 110) of the sets passed AIS-31 Procedure B, which means that an appropriate parameter set can be found by ten or less trials with more than 99% probability. The average throughput of them was 2.44 Mbit/s, which was comparable to recent FPGA-based TRNGs. An initial prototype of dynamic reconfiguration of the parameters is also presented in this paper.

1 Introduction

For secure computing and networking systems, a true random number generator (TRNG) is an important component to obtain unpredictable random numbers. They are used as, for example, an encryption key and nonce of challenge-response protocols. A TRNG utilizes a physical phenomenon as a source of entropy, of which the AIS-31 standard [11] requires an appropriate stochastic model.

There are some types of TRNGs that are suitable for FPGA (field programmable gate array) implementations and compliant with the AIS-31 [16]. They use physical phenomena of internal logic or complementary elements, while thermal noise of resisters [24] or transistors [14] is often used in ASIC (application-specific integrated circuit) implementations. Coherent sampling [1, 12, 13] is one of the operating principles of them.

Coherent sampling-based TRNGs with clocking elements, such as phase-locked loop (PLL) [4] and digital clock manager (DCM) [10], have an advantage of being implementable with a minimal number of logic elements. Coherent sampling requires two clock signals that have slightly different frequencies. Instead of using two ring oscillators, they use frequency synthesized signals from one oscillator or an external clock input. They require two clocking elements but fewer logic elements. In general, an FPGA-based computing and networking system requires a large number of logic elements, while most of clocking elements remain unused. They let precious logic resources use for other parts of the system.

In this paper, we present a novel coherent sampling-based TRNG using recent clocking elements of Xilinx FPGAs called MMCMs (mixed-mode clock managers) [22]. MMCM is available for Virtex-6 and 7 series (or newer) FPGAs and a PLL for these FPGAs is its subset [22]. Although it offers finer

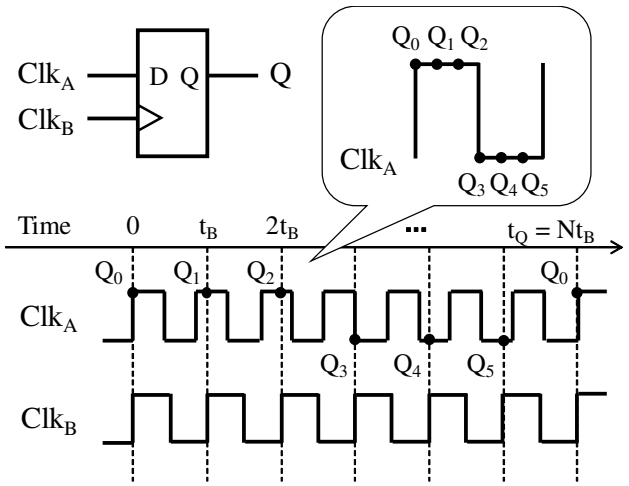

Figure 1: Example of coherent sampling where the frequency ratio is 7 : 6.

multiplying and dividing factors to be set as parameters than earlier DCMs, it was not clear how to effectively use its functionality for TRNGs. At first, we demonstrate that simply porting an existing DCM-based TRNG [10] is not enough for a newer FPGA. We then propose selection strategies of parameter sets of MMCMs and show that entropy and bit rate of generation are increased by the selected parameter sets. We also present an initial prototype of dynamic reconfiguration of the parameters. The most important contribution of this paper is to show the feasibility of TRNGs with clocking elements in a system based on recent Xilinx FPGAs.

We have presented a preliminary version of this study in the PDAA workshop of CANDAR 2020 [6]. Major differences from the preliminary version are as follows.

- An evaluation result with additional sets of parameters is presented in Section 5.
- A prototype system with dynamic reconfiguration of parameters is developed and evaluated, which is described in Section 6.
- Other types of TRNGs suitable for FPGAs are reviewed in Section 7.

The organization of this paper is as follows. A brief explanation of the basis of our research, coherent sampling and MMCM, is presented in Section 2. Section 3 describes porting of the existing DCM-based method, while Section 4 presents the proposed selection of parameter sets of MMCMs. The quality of random numbers, the generation bit rate, and the amount of hardware of the MMCM-based TRNG are evaluated in Section 5. Section 6 describes the development and evaluation of a prototype of dynamic reconfiguration. A review on other types of TRNGs suitable for FPGAs is presented in Section 7. Finally, we conclude the paper in Section 8.

2 Background

2.1 Principle of Coherent Sampling

Figure 1 depicts the operating principle of coherent sampling with an example. It requires two clock signals that have slightly different frequencies. These signals, $\mathrm{Clk_A}$ and $\mathrm{Clk_B}$, are given to the data and clock input ports of a D flip-flop (D-FF), respectively. For ease of explanation, we assume that the ratio of the frequencies is $f_A : f_B = (N+1) : N$. The example of Figure 1 shows the case of the ratio of 7 : 6. The periods of the respective clocks are denoted by t_A and t_B.

Suppose that the both clock signals rise at time zero. The time when the both clocks rise at the same time again will be $t_Q = Nt_B$. As $\mathrm{Clk_B}$ goes slightly slower than $\mathrm{Clk_A}$, it can effectively capture a waveform of a single cycle of $\mathrm{Clk_A}$ with N samples. As a result, the output of the D-FF becomes

a series of consecutive '1's and consecutive '0's. The expected value of the number of consecutive '1's is $N/2$ (if the duty cycle of $\mathrm{Clk_A}$ is 1/2).

TRNGs utilize the jitter of these clock signals. When the vicinity of edge of $\mathrm{Clk_A}$ (such as Q_0 and Q_3 in Figure 1) is captured, the output of the D-FF may vary with slight time jitter. Also, when the both edges come quite close, D-FF may fall into a metastable state due to timing violation, which results in random output. These phenomena make the actual number of consecutive '1's uncertain. Its LSB (least significant bit) can be used as a source of entropy, which is obtained by a T flip-flop (a D-FF and an XOR gate). To harvest enough entropy, the jitter σ_J should be sufficiently larger than the difference of the periods $t_d = t_B - t_A = t_A/N$. Without considering the effect of metastability, the standard deviation of the number of '1's is proportional to σ_J/t_d.

2.2 Coherent Sampling-based TRNG

One of the methods to obtain clock signals for coherent sampling-based TRNGs is to use two ring oscillators [12, 15, 19]. Even though the oscillators have the same topology, their oscillation frequencies may slightly differ because of manufacturing variation. If we got a proper period difference t_d, we would generate high-quality random numbers at a fast rate (in an order of Mbit/s). However, improper t_d results in the lack of entropy (too large t_d), or reduction of the bit rate of generation (too small t_d). It was a serious problem for practical use to properly adjust t_d. A solution for this problem has recently been proposed, which uses route-selectable ring oscillators [15]. To improve the generation bit rate, a mutual sampling method [19] was proposed, which captured $\mathrm{Clk_B}$ by $\mathrm{Clk_A}$ in addition to capturing $\mathrm{Clk_A}$ by $\mathrm{Clk_B}$. It was reported, however, that the quality of random numbers was degraded due to the correlation among the output bits [19].

Another method for clock signals is to utilize clocking elements such as PLLs [4] and DCMs [10]. Since factors of multiplication and division can be set as parameters, it is easy to get a proper frequency ratio. When an external clock source is used, no ring oscillators are required. Even when clocks must be generated internally (i.e. an external clock source is unreliable), only one ring oscillator is required. This type of TRNGs have an advantage of minimizing the number of required logic elements in exchange for two additional PLLs or DCMs. A shortcoming of this method is relatively large power consumption of clocking elements.

Our research is based on a DCM-based TRNG proposed by Johnson et al. [10], whose target is Xilinx Virtex-5. The parameters of Virtex-5 DCM are multiplier M and divisor D. They both must be integer and meet $2 \leq M \leq 33$ and $1 \leq D \leq 32$. Proper range of N is $400 \leq N \leq 1000$ (i.e. $t_A/400 \geq t_d \geq t_A/1000$) [10]. They presented 23 sets of parameters that meets these conditions [10]. For example, from an input clock of $f_{IN} = 100$ MHz, they generated $\mathrm{Clk_A}$ of $(15/31)f_{IN} \sim 48.39$ MHz and $\mathrm{Clk_B}$ of $(14/29)f_{IN} \sim 48.28$ MHz. In this case, the frequency ratio is $435 : 434$ (i.e. $N = 434$) and the expected value of the number of '1's is 217. The numbers of '1's are obtained at the rate of $48.28/434 \simeq 0.111$ Msample/s. In the DCM-based TRNG, three LSBs of the number of '1's are extracted and the generated bitstring is post-processed by the von Neumann Corrector [21]. Its bit rate of generation is, theoretically, $0.111 \times 3 \times 1/4 \simeq 0.083$ Mbit/s. Although this rate varies with parameters, the actual rate was 0.210 Mbit/s on average according to an additional evaluation with various parameters [7]. In this paper, in consideration of a requirement of AIS-31 [11] for a random bitstring without post-processing, only one LSB of the number of '1's is extracted and post-processing is not applied unless explicitly stated.

There are two methods to deal with the number of '1's: the number of consecutive '1's [13] and the sum of the number of '1's in N samples [1]. When using two ring oscillators, the latter method is not available because the frequency ratio cannot be exactly determined. With clocking elements, the both methods can be used. The DCM-based TRNG by Johnson et al. adopted the former method [10]. The former method detects the falling edge of output of the D-FF, instead of counting the number of samples, which makes hardware simpler. However, since sampling the vicinity of edge happens consecutively, '0's ('1's) may appear in consecutive '1's ('0's). When counting the number of consecutive '1's in the former method, this causes quite small counter values, which have smaller entropy than counter values near the expected value [7]. This research adopts the latter method to avoid such a negative effect.

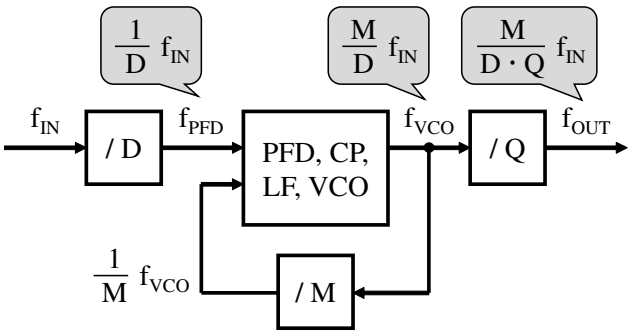

Figure 2: Simplified block diagram of a mixed-mode clock manager (MMCM).

2.3 Mixed-mode Clock Manager

Figure 2 abstracts the operation of the mixed-mode clock manager (MMCM) [22]. Functions unrelated to this research are omitted from this figure. The input clock of frequency of f_{IN} is first divided by D, and then passed phase frequency detector (PFD), charge pump (CP), loop filter (LF), and voltage-controlled oscillator (VCO). The input frequency of the PFD f_{PFD} is

$$f_{PFD} = \frac{1}{D} f_{IN}. \tag{1}$$

The output of the VCO is fed back to the PFD after divided by M. Since this feedback signal is controlled in order to have the same frequency as f_{PFD}, the following equation holds:

$$f_{PFD} = \frac{1}{M} f_{VCO}. \tag{2}$$

From Equations (1) and (2), the output frequency of the VCO f_{VCO} is multiplied by M and becomes

$$f_{VCO} = M \cdot f_{PFD} = \frac{M}{D} f_{IN}. \tag{3}$$

The output of the MMCM is obtained by dividing the VCO output by Q, whose frequency f_{OUT} is

$$f_{OUT} = \frac{M}{D \cdot Q} f_{IN}. \tag{4}$$

The parameters of the MMCM, D, M, and Q, have the following constraints:

$$1 \leq D \leq 106, \tag{5}$$
$$2 \leq M \leq 64, \tag{6}$$
$$1 \leq Q \leq 128. \tag{7}$$

D must be integer, while M and Q can be integer or fraction with 1/8 interval. These parameters enable setting of the output frequency to be finer than the earlier DCM.

These frequencies have constraints due to the characteristics of the PFD and the VCO. They are slightly different with FPGA family and speed grade. The target FPGA of our evaluation, Artix-7 of speed grade -1, has the following constraints [23]:

$$10 \leq f_{PFD} \leq 450 \text{ [MHz]}, \tag{8}$$
$$600 \leq f_{VCO} \leq 1200 \text{ [MHz]}, \tag{9}$$
$$4.68 \leq f_{OUT} \leq 800 \text{ [MHz]}. \tag{10}$$

The input frequency is set to $f_{IN} = 100$ MHz in this research. According to Equations (8) and (9) for D and M/D, respectively, the effective constraints of the parameters are as follows:

$$1 \leq D \leq 10, \tag{11}$$
$$6 \leq \frac{M}{D} \leq 12. \tag{12}$$

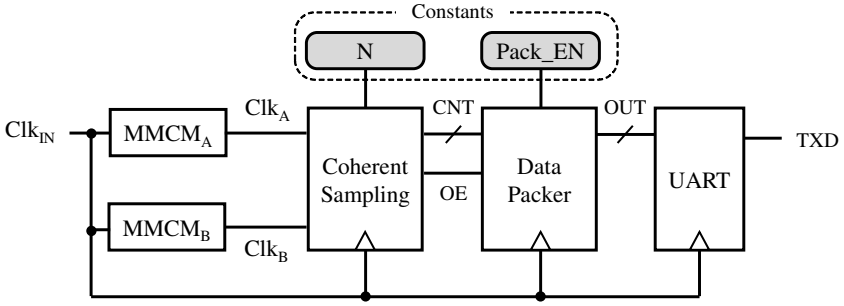

Figure 3: Block diagram of an evaluation system of TRNG.

Table 1: Parameters of MMCM to simply port the DCM-based TRNG [10].

ID	Target	M_A	D_A	Q_A	M_B	D_B	Q_B
J01	DCM	15	31	-	14	29	-
	MMCM	7.50	1	15.50	7.00	1	14.50
J02	DCM	21	22	-	20	21	-
	MMCM	10.50	1	11.00	10.00	1	10.50
J22	DCM	30	31	-	29	30	-
	MMCM	7.50	1	7.75	7.25	1	7.50
J23	DCM	31	32	-	30	31	-
	MMCM	7.75	1	8.00	7.50	1	7.75

In addition, the MMCM offers the power down mode [22] to save power consumption when it is not in use. Although it might be a solution of the shortcomings of PLL/DCM-based TRNGs referred to in Section 2.2, we do not consider it in this paper.

3 Porting of DCM-based TRNG

All of our evaluations in this paper use a Digilent Arty FPGA board, which includes an Artix XC7A35T FPGA. Circuits are synthesized by Vivado 2019.2 with the default options unless explicitly stated.

Figure 3 abstracts our evaluation system. From the 100-MHz input clock on the Arty board, two clock signals are generated by two MMCMs. A coherent sampling module counts the number of '1's for each N samples. A data packer packs the LSBs of eight counter values into a single byte. A sequence of bytes (i.e. random numbers) is sent to a PC via UART. The baud rate is set to 3 Mbps or 6 Mbps in order that the UART transmitter does not become a bottleneck.

In this evaluation system, a programming file must be generated for each set of parameters of MMCMs. In other words, their parameters are given as constants. As the number of samples N varies with the parameters, N is also given as a constant. In addition, data packing can be disabled by setting another constant, Pack_EN, to zero to send a sequence of counter values.

In this section, we simply port the DCM-based TRNG by Johnson et al. [10] to FPGAs with MMCMs. Although 23 parameter sets for DCMs were listed in their paper [10], because of the constraints on M and D, or Equations (11) and (12), these parameters cannot be directly adopted. To make the parameters comply with MMCMs, Q is used as a dividing factor instead of D, and then both M and Q are divided by two (if $M \leq 24$) or four (if $M > 24$). Table 1 shows a part of the parameter sets obtained from this strategy. We assigned numbers of J01, J02, ..., and J23 to the parameter sets of the DCM-based TRNG [10]. We use J as their initial because they were originally presented by Johnson et al. The parameters M_A, D_A, Q_A are for $\mathrm{Clk_A}$ and M_B, D_B, Q_B are for $\mathrm{Clk_B}$. We measured 10^7 counter values (the number of '1's for each N samples) for each parameter set and plotted their distribution.

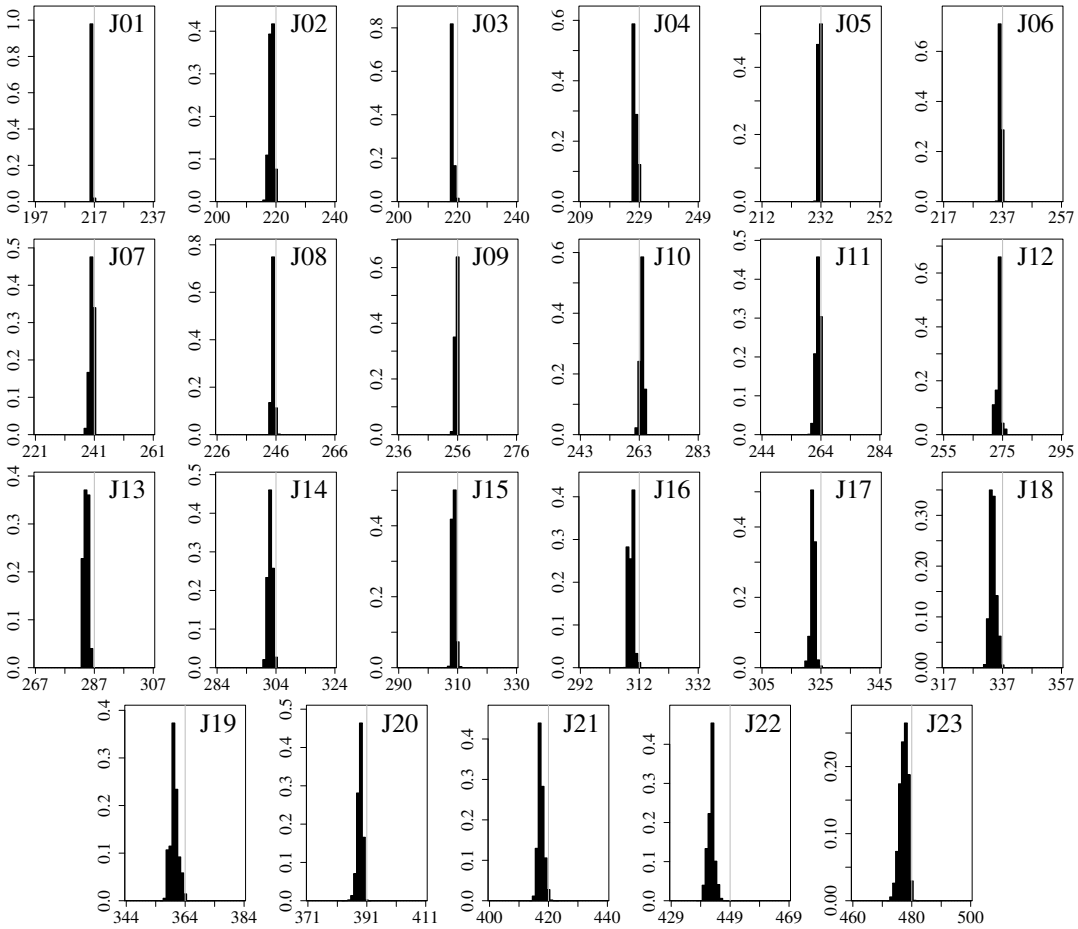

Figure 4: Distribution of counter values using the parameters shown in Table 1. For the all plots, the X-axis is the counter value and the Y-axis is the occurrence frequency.

Figure 4 depicts the distributions of counter values with all of the parameter sets. The X-axis represents the counter value and the Y-axis represents the occurrence frequency for each value. A gray vertical line in the center corresponds to the expected value of the counter, $N/2$. The variance of the counter values was much smaller than the results of the previous experiments with Virtex-5 DCMs [7]. In particular with J01 parameter set, 98% of the values were the same: 216. As we will evaluate in more detail in Section 5, TRNGs constructed from these parameters do not give enough entropy and, as a result, they fail statistical tests in most cases. It might be because logic and clocking elements of the newer FPGAs become more fault tolerant: jitter of MMCMs, setup time of D-FFs, and hold time of them might get smaller. Also, their bit rate of generation are about 0.1 Mbit/s, which is an order of magnitude slower than other type of TRNGs suitable for FPGAs [16]. The conclusion of this section is that a simple porting of the DCM-based TRNG is not enough in either randomness or throughput.

4 Enhanced Parameter Selection

4.1 Method for Larger Entropy

As we have overviewed in Section 2.1, there are two ways to increase the variance of the number of '1's: decreasing the difference of the periods t_d or increase the jitter σ_J. Decreasing the period difference should be avoided because it has a tradeoff with the generation bit rate. Instead, we

Table 2: Frequency and peak-to-peak jitter of the output of MMCM.

M	D	Q	Freq. [MHz]	Jitter [ps]
7.75	1	8.00	96.875	141.837
15.50	2	8.00		184.566
23.25	3	8.00		229.787
31.00	4	8.00		273.577
38.75	5	8.00		305.392
46.50	6	8.00		343.210
54.25	7	8.00		383.515
62.00	8	8.00		427.425

Table 3: Parameters of MMCM to obtain larger jitter.

ID	Method	M_A	D_A	Q_A	M_B	D_B	Q_B
J01	NM	7.50	1	15.50	7.00	1	14.50
	JT	**60.00**	**8**	15.50	**63.00**	**9**	14.50
J02	NM	10.50	1	11.00	10.00	1	10.50
	JT	**63.00**	**6**	11.00	**60.00**	**6**	10.50
J22	NM	7.50	1	7.75	7.25	1	7.50
	JT	**60.00**	**8**	7.75	**58.00**	**8**	7.50
J23	NM	7.75	1	8.00	7.50	1	7.75
	JT	**62.00**	**8**	8.00	**60.00**	**8**	7.75

adjust the parameters in order to increase the jitter while keeping the frequency unchanged.

Concretely speaking, we multiply both M and D by the same integer. When multiplying and dividing factors are large, the effect of internal noise becomes large and the jitter becomes increased. For example, Table 2 summarizes the peak-to-peak jitter of $\mathrm{Clk_A}$ of J23 parameter set, when M and D are multiplied by 2, 3, ..., and 8. These values can be found from the clocking wizard of Vivado [22]. The worst case jitter becomes three times larger. Although the reason of this increase of jitter is not explicitly described in Xilinx's user guide [22], it should be interpreted as the effect of clock dividers. Since a clock divider includes a counter, its bit width becomes large as the divisor increases, resulting in a large propagation delay. This implies that the multiplier for M and D should be as large as possible. The maximum multiplier, say D_{max}, is constrained by Equation (6) as follows:

$$D_{max} = \left\lfloor \frac{64}{M} \right\rfloor, \tag{13}$$

where $\lfloor x \rfloor$ is the maximum integer that is not more than x. Since M/D remains unchanged, no additional constraints come from Equation (12).

Table 3 illustrates some parameter sets modified by the above strategy. The method to obtain parameter sets shown in Section 3 is represented as NM (Normal). The method proposed here is represented as JT (Jittery). Both M and D are multiplied by D_{max} to maximize the jitter while Q is kept unchanged. This modification is applied to both $\mathrm{Clk_A}$ and $\mathrm{Clk_B}$.

Figure 5 depicts the distribution of counter values where the JT method is applied. Note that the scale of the Y-axis is not the same as Figure 4. The variation of counter values apparently got larger: their standard deviation became 2.6 times larger on average and 10 times larger at a maximum (J01). Considering the effect of quantization (that counter values must be integer), this result basically matches the increase of jitter.

With some sets of parameters, most obviously with J22, irregularity of distribution was observed. Even numbers appeared more frequently than odd numbers in more than half of the cases. We leave it for future work to find out why this phenomenon occurs, though resolving it would further improve the quality of random number.

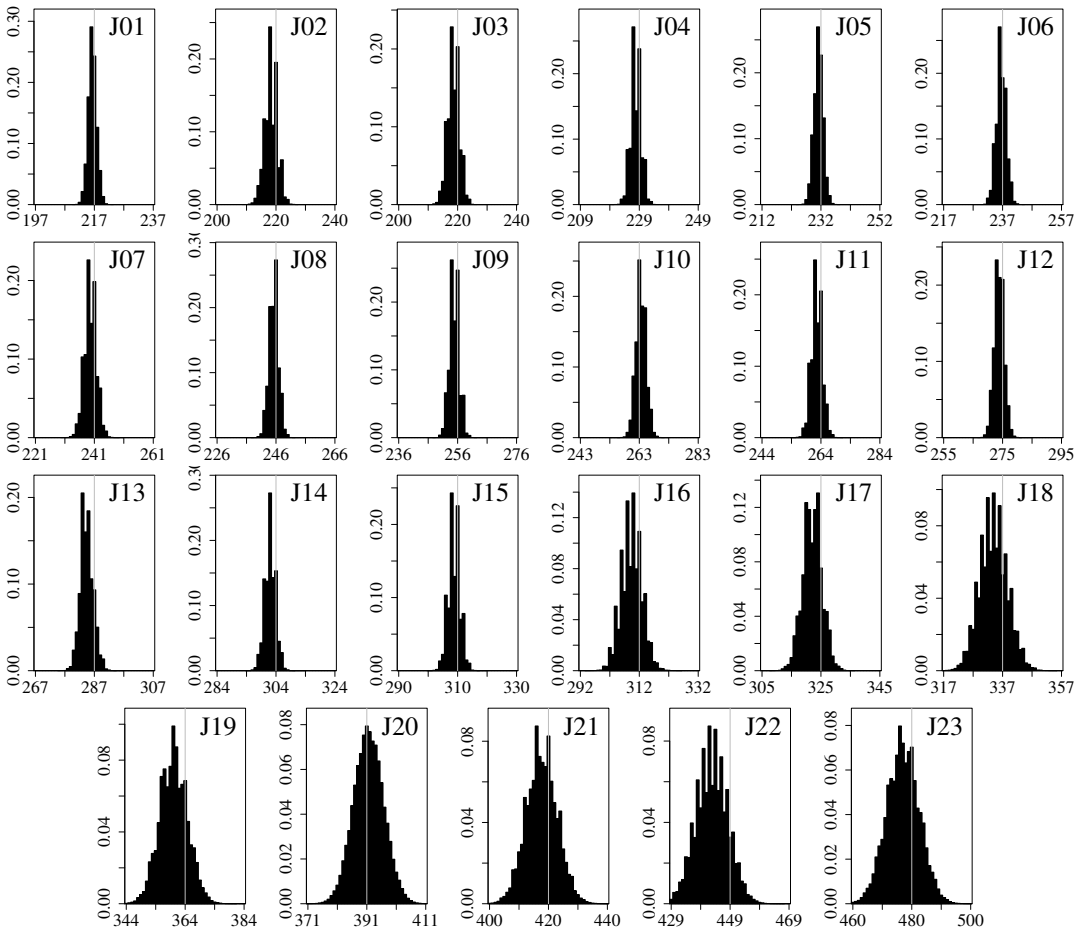

Figure 5: Distribution of counter values using the parameters shown in Table 3.

4.2 Method for Higher Throughput

In this section, we consider dividing N samples of the coherent sampling into K sections of N/K samples and counting the number of '1's for each section, where K is a natural number and N is divisible by K. This will make the generation bit rate of the TRNG be K times higher. From a lesson from mutual sampling [19], the count should be independent from each other; otherwise, unwanted correlation may occur. In this situation, each section must sample the logically same waveform. This can be done by multiplying the frequency of the sampled clock (Clk_{A}) by K or, more specifically, dividing the parameter Q_A by K. Of course, the resultant Q_A must be a permitted value as a parameter of the MMCM.

Figure 6 illustrates this idea in the case of $K = 2$, by a similar example to Figure 1. The frequency ratio is now changed from $7 : 6$ to $14 : 6$. From the principle of coherent sampling, in this case the waveform of two cycles of Clk_{A} is captured by N samples. If each of the counter values of the first half (Q_0, Q_1, and Q_2 in the example) and the last half (Q_3, Q_4, and Q_5) is counted independently, the generation bit rate can be doubled while avoiding the effect on the entropy (the variance of the counter values).

Table 4 shows some parameter sets after applying this method with $K = 2$ while keeping the JT method applied. We denote this combination of the proposed methods as CB (Combined). Since N had to be divisible by K, we applied the CB method to 13 (out of 23) parameter sets where N was even. Only Q_A is halved and the other parameters are left unchanged.

According to an evaluation, the distribution of the counter values was almost unchanged. The generation bit rate was exactly doubled because the circuit to count the number of samples is

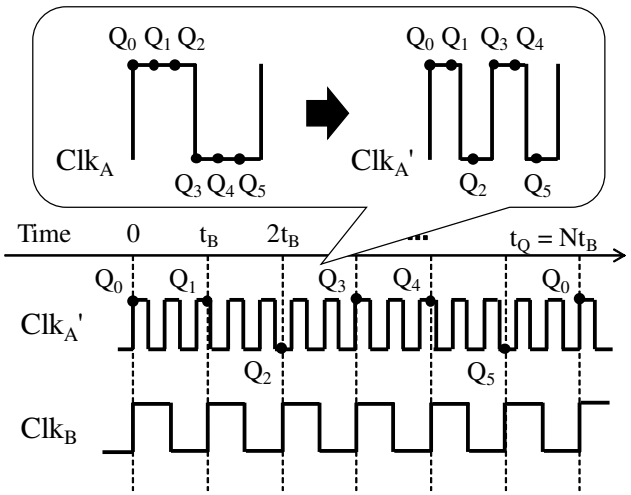

Figure 6: Example of coherent sampling where the frequency ratio is 14 : 6.

Table 4: Parameters of MMCM to obtain higher throughput.

ID	Method	M_A	D_A	Q_A	M_B	D_B	Q_B
J01	JT	60.00	8	15.50	63.00	9	14.50
	CB	60.00	8	**7.75**	63.00	9	14.50
J02	JT	63.00	6	11.00	60.00	6	10.50
	CB	63.00	6	**5.50**	60.00	6	10.50
J21	JT	58.00	8	7.50	63.00	9	7.25
	CB	58.00	8	**3.75**	63.00	9	7.25
J23	JT	62.00	8	8.00	60.00	8	7.75
	CB	62.00	8	**4.00**	60.00	8	7.75

* J22 is not available because its N is odd.

deterministic. We will conduct a detailed evaluation in Section 5.

4.3 Selection Strategy

Based on the examinations shown in Section 4.1 and Section 4.2, we consider selection of parameter sets in order to maximize the effectiveness of the proposed CB method. First, to maximize the generation bit rate,

$$Q_A \in \mathbb{N} \tag{14}$$

in the NM and JT methods and the multiplier $K = Q_A$. This means $Q_A = 1$ in the CB method. From the constraints of the frequencies of the VCO and the output, (9), (10), the range of M_A becomes

$$6 \leq M_A \leq 8. \tag{15}$$

As a result from Equations (13) and (15), D_A in the JT and CB methods becomes either 8, 9, or 10, which maximizes the jitter. For $\mathrm{Clk_B}$, although arbitrary parameters can be set as long as Equation (12) is met, we constrain the range of M_B as

$$6 \leq M_B \leq 8, \tag{16}$$

to make D_B be also either 8, 9, or 10. Finally, parameters that meets constraints on the output frequency are selected: the integer ratio of the frequencies is $(N + 1) : N$, N is within a predefined

Table 5: Grouping of the parameter sets.

Name	Condition	# of Sets
A	$150 \leq N < 220$	77
B	$220 \leq N < 300$	64
C	$300 \leq N < 400$	46
D	$400 \leq N < 600$	67
E	$600 \leq N \leq 1000$	65
All	$150 \leq N \leq 1000$	319

Table 6: Parameters of MMCM that can maximize jitter and throughput.

ID	Method	M_A	D_A	Q_A	M_B	D_B	Q_B
A01	NM	6.000	1	7.000	6.500	1	7.625
	JT	**60.000**	**10**	7.000	**58.500**	**9**	7.625
	CB	60.000	10	**1.000**	58.500	9	7.625
A02	NM	6.000	1	7.000	7.250	1	8.500
	JT	**60.000**	**10**	7.000	**58.000**	**8**	8.500
	CB	60.000	10	**1.000**	58.000	8	8.500
A03	NM	6.250	1	7.000	6.875	1	7.750
	JT	**62.500**	**10**	7.000	**61.875**	**9**	7.750
	CB	62.500	10	**1.000**	61.875	9	7.750
E63	NM	7.875	1	15.000	6.750	1	12.875
	JT	**63.000**	**8**	15.000	**60.750**	**9**	12.875
	CB	63.000	8	**1.000**	60.750	9	12.875
E64	NM	8.000	1	15.000	6.125	1	11.500
	JT	**64.000**	**8**	15.000	**61.250**	**10**	11.500
	CB	64.000	8	**1.000**	61.250	10	11.500
E65	NM	8.000	1	15.000	7.125	1	13.375
	JT	**64.000**	**8**	15.000	**57.000**	**8**	13.375
	CB	64.000	8	**1.000**	57.000	8	13.375

range, and N is divisible by K. The number of possible sets of parameters is at most an order of hundreds of thousands. We conducted full search to find promising parameter sets.

In this paper, we set the range of N to $150 \leq N \leq 1000$. The number of samples per count, N, affects the tradeoff between the generation bit rate and the quality of random numbers. We decreased the lower limit of N from 400, which was used in the DCM-based TRNG [10] and the preliminary version of this study [6], to aim for as high throughput as possible.

We found 319 sets of parameters in a range of sampling frequency between 50 and 100 MHz. We grouped them according to N as shown in Table 5. We gave an ID to each of them using the name of the corresponding group and a serial number in the group. For example, the parameter sets where $150 \leq N < 220$ were assigned numbers of A01, A02, ..., and A77. Table 6 summarizes a part of them. There are two parameter sets which have also enumerated in the DCM-based TRNG [10]: E03 and E10 are identical to J19 and J23 (except for the value of K in the CB method), respectively. Full lists of the parameter sets are available at a GitHub repository `https://github.com/nfproc/MMCM_TRNG`.

5 Evaluation

5.1 Min-entropy

In this section, we evaluate the effect of the proposed parameter selection on the entropy, the bit rate of generation, and the amount of hardware of an MMCM-based TRNG. We first evaluate the entropy based on the parameter sets from the DCM-based TRNG (i.e. J01–J23). We measured

Table 7: Min-entropy of generated random bitstrings.

ID	NM	JT	CB	ID	NM	JT	CB
J01	**0.0294**	0.9964	0.7988	J13	0.7646	0.8534	-
J02	0.9254	0.6097	**0.6157**	J14	0.9749	0.6827	0.6623
J03	0.2614	0.6742	-	J15	0.9762	0.5669	0.7551
J04	0.4915	**0.5647**	-	J16	0.4921	0.6435	0.8879
J05	0.9117	0.9191	0.7953	J17	0.8709	0.9839	0.9935
J06	0.4927	0.7687	-	J18	0.9895	0.7425	-
J07	0.9800	0.6926	-	J19	0.7684	0.9936	1.0000
J08	0.4113	0.7566	-	J20	0.8910	0.9917	-
J09	0.6232	0.6316	0.6782	J21	0.8384	0.8969	0.8906
J10	0.7167	0.8034	-	J22	0.6670	0.7579	-
J11	0.9621	0.6798	0.6442	J23	0.9863	0.9963	0.9959
J12	0.3369	0.8610	0.8002	Avg.	0.7114	0.7855	0.8090
					(0.7458)	(0.8047)	

the occurrence frequency of the LSB of 10^7 counter values. We calculated the min-entropy of the sequence of LSBs as $H_\infty = -\log_2\{\max(p_0, p_1))\}$, where the occurrence frequencies of the LSB of '0' and '1' are denoted as p_0 and p_1, respectively.

Table 7 enumerates the calculated min-entropy. The row Avg. corresponds to the arithmetic mean of the evaluated sets. The arithmetic mean of the 13 sets to which the CB method is applicable is noted in parentheses. The minimum value for each method is shown in boldface type. The results indicates that the min-entropy is basically increased by applying the JT method. In particular, any case that only one counter value frequently appeared, as shown in J01 of Figure 4, were not observed in the JT and CB methods. No significant differences were observed between JT and CB.

5.2 AIS-31 Statistical Tests

We then evaluate the quality of random bitstrings, generated by concatenating the LSBs of the counters, using AIS-31 [11] Procedure B statistical test suite. We used all of the 319 parameter sets found in Section 4.3. This test suite assumes that the input bitstring is no less than 7 Mbit long and not post-processed. If the decision is not reliable (i.e. failing in only one of the tests), the tests are conducted again with another bitstring. We thus generated 16 Mbits of bitstring for each parameter set. We also calculated the generation bit rate by measuring the time to obtain the bitstring.

Figure 7 summarizes the result of the statistical tests, while Figure 8 plots the average of generation bit rate. The X-axis represents a kind of method and a group of parameter sets, while Y-axis is the proportion of sets passed or failed (Figure 7) or the generation bit rate (Figure 8). Failed parameter sets are excluded from the calculation of the average bit rate. Since none of the sets passed the test in the group B of the NM method, its average bit rate was not available (N/A). If the proporion of passed sets is smaller than 10%, corresponding bars in Figure 8 are marked in gray.

While only 14 sets (out of 319) passed when the existing method is simply ported (NM), 197 sets and 134 sets passed in JT and CB, respectively. Interestingly, though the proportion of passed sets was almost the same among the groups in the JT method, it decreased by the decrease of N in the CB method. A possible reason is that the number of samples N/K became too small relative to the deviation of counter value. An expected usage in actual applications is to find an appropriate parameter set through testing some possible sets using dynamic reconfiguration (see Section 6 for our initial prototype). Increase of the number of passing parameter sets means the reduction of the time spent for such an advance preparation. For example, in the groups B and C, 38.2% (42 out of 110) of the sets passed the test with the CB method. This means that, if a parameter set is randomly selected, we can find an appropriate parameter set by ten or less trials on these sets with more than 99% probability. However, the proportion of passed sets dropped to less than 5% in the group A, which means searching for appropriate parameter set from them is impractcal.

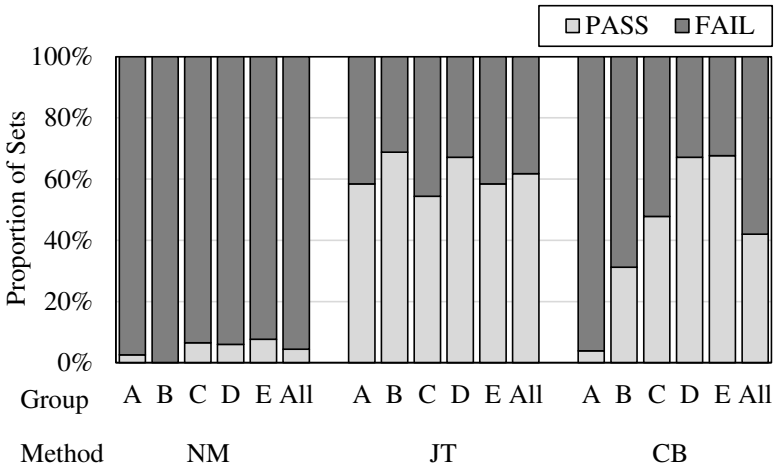

Figure 7: Proportion of parameter sets that passed AIS-31 Procedure B.

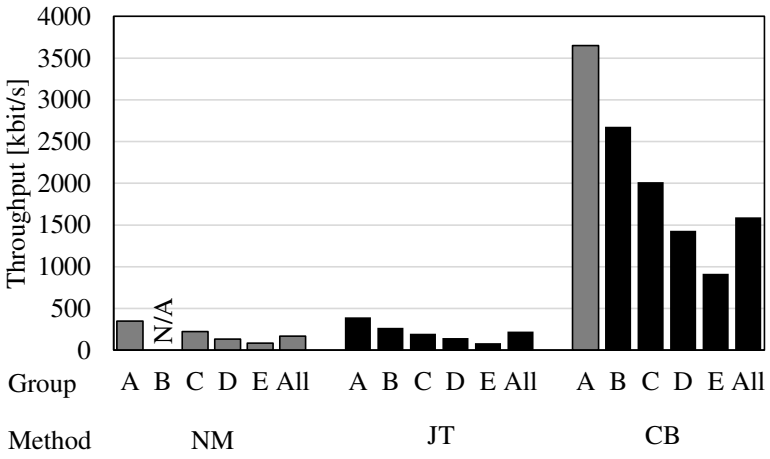

Figure 8: Generation rate of random bits.

Regarding the generation bit rate, CB achieved 1.59 Mbit/s on average while NM and JT were 0.167 Mbit/s and 0.225 Mbit/s, respectively. In other words, the proposed TRNG was about ten times faster than the simple porting of the existing method by Johnson et al. [10]. It was even 7.6 times faster than their implementation on a Virtex-5 FPGA [7]. In particular, when we extract the groups B and C as the most promising parameter sets, the average bit rate reached 2.44 Mbit/s. It was comparable to other types of recently proposed FPGA-based TRNGs [16].

5.3 NIST SP 800-22 Test Suite

We conduct a more detailed statistical test, NIST SP 800-22 test suite [17], to random numbers generated by the proposed method with a simple post-processing. We chose E10 (basically equivalent to J23) and B01 (the first set in the group B that passed AIS-31) parameter sets with the CB method. As a post-processing method, the random bitstring was XOR-ed with an output sequence of 4-bit (E10) or 8-bit (B01) linear feedback shift register (LFSR) by software. This corresponds to a quite simple debiasing. As recommended in AIS-31 [11], we obtained 1,073 1-Mbit bitstrings and conducted the tests to each of them. The generation bit rate of the bitstrings was 0.808 Mbit/s and 2.667 Mbit/s, respectively.

Table 8: Result of the NIST SP 800-22 test suite. A simple post-processing with a 4-bit (E10) or 8-bit (C01) LFSR was applied.

name	E10		C01	
	p-value	proportion	p-value	proportion
Frequency	0.37287	99.35%	0.29542	98.79%
BlockFrequency	0.72770	98.97%	0.61767	99.07%
CumulativeSumsUp	0.85956	99.35%	0.07620	98.97%
CumulativeSumsDown	0.38881	99.44%	0.60990	99.16%
Runs	0.05050	98.97%	0.77034	98.97%
LongestRun	0.21278	99.25%	0.28874	98.88%
Rank	0.24976	98.97%	0.12067	99.16%
FFT	0.82609	99.16%	0.60407	98.88%
NonOverlappingTemplate	0.16313	99.00%	0.10119	99.01%
OverlappingTemplate	0.06071	98.60%	0.43040	99.07%
Universal	0.02751	98.79%	0.40681	99.07%
ApproximateEntropy	0.60990	98.97%	0.44073	99.07%
RandomExcursions	0.97985	98.98%	0.74299	99.06%
RandomExcursionsVariant	0.01551	99.36%	0.13748	98.88%
Serial1	0.93685	98.70%	0.26065	99.35%
Serial2	0.57125	98.60%	0.40021	98.88%
LinearComplexity	0.01405	99.07%	0.15433	98.42%

Table 9: Number of logic elements for TRNG.

Method	LUT	D Flip-flop
NM	19	18
JT	19	18
CB	17	18

Table 8 summarizes the result of the NIST test suite. For each test, a p-value and the proportion of passed bitstrings are shown. The test is considered as pass if the p-value is no less than 10^{-4} and the proportion is within 3σ range from 99%. The post-processed bitstrings passed the NIST test suite, for all of the tests meets these conditions. The raw bitstrings did not pass the tests but it was expected result. Procedure B of AIS-31 targets raw bitstrings and they have small bias acceptable there. They are expected to be used with a post-processing.

5.4 Amount of Hardware

Finally, we evaluate the amount of hardware after synthesized, placed and routed. Evaluated circuits were the coherent sampling and the data packer in Figure 3. The UART transmitter and a post-processing circuit (assumed in Section 5.3) is not included. To avoid packing with other circuits, we add a `-flatten_hierarchy none` synthesis option. We used the E10 parameter set in the same way as Section 5.3.

Table 9 shows the summary of the implementation results. The number of required LUTs (look-up tables) or flip-flops was comparable to the DCM-based TRNG [10] and much smaller than other circuits.

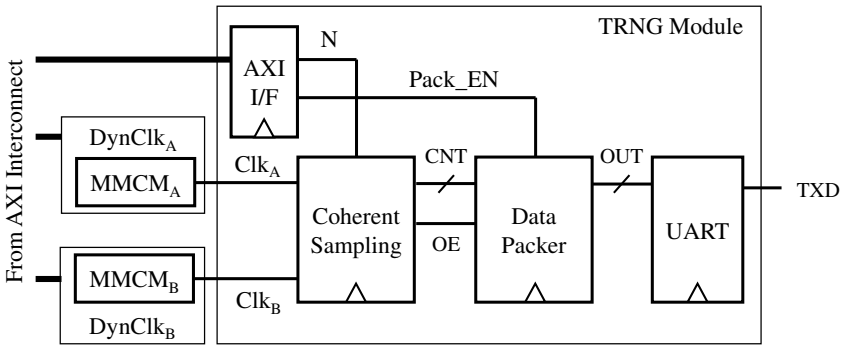

Figure 9: Block diagram of the evaluation system modified for dynamic reconfiguration.

Table 10: Parameters of MMCMs that consists only of integers.

ID	M_A	D_A	Q_A	M_B	D_B	Q_B	N
D17	57	8	1	64	8	9	64
E25	60	10	1	61	8	14	61
E31	63	9	1	61	8	12	61
E57	64	8	1	59	8	12	59
E58	57	8	1	61	8	15	61
D63	58	8	1	62	8	15	31

6 Prototyping of Dynamic Reconfiguration

6.1 System Design

In this section, we describe and evaluate a prototype system for dynamic reconfiguration of parameters. As explained in Section 3, the evaluation system in the previous sections has constants of MMCM parameters, the number of samples per count, and whether the data packer is enabled. Changing parameters of MMCMs in operation requires dynamic reconfiguration of MMCM [18]. Furthermore, in an FPGA-based system, circuits are usually packaged as IP cores and controlled by a processor via an AXI interface.

Based on above considerations, we built the system using IP cores and a MicroBlaze soft processor. Figure 9 describes basic components of the system controlled by MicroBlaze. To achieve dynamic reconfiguration of an MMCM, we use a DynClk (dynamic clocking) IP developed by Digilent [3]. Components except MMCMs in Figure 3 are packed into another IP, depicted as TRNG module in Figure 9. To enable modification of the number of samples (N) and whether the data packer is enabled (Pack_EN), an AXI interface (AXI I/F) circuit is added to the TRNG module.

On the development of the software, we slightly modified the driver of the DynClk core. It originally takes a target output frequency as an argument and the parameters of the MMCM (M, D, and Q) are calculated from it. We modified it to bypass this calculation and to provide the parameters directly. Also, since the driver did not support fractional values for M and D and it was expected that modification will take a lot of time and effort, we chose parameter sets that consisted only of integers. From the parameter sets in the groups D and E, we found six sets shown in Table 10.

6.2 Verification of Operation

We ran a test program on the prototype system, where the counter values were generated for ten seconds for each set of parameters. Figure 10 plots the time variation of the distribution of counter values throughout the runtime of the program. The x-axis represents the elapsed time and the y-axis stands for counter value. The more frequently the counter value appears, the darker the color of the

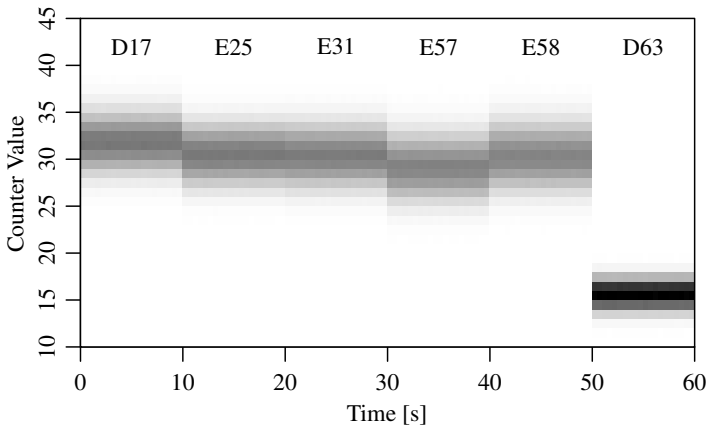

Figure 10: Time variation of the distribution of counter values.

Table 11: Number of logic elements for the prototype system.

Component	LUT	D Flip-flop
Whole System	2,366	2,382
MicroBlaze	1,061	934
TRNG Module	487	359
UART	428	277
Others	59	82
DynClk	239	352
AXI Interface	187	269
Others	52	83

corresponding cell. The distribution was measured every 0.5 seconds.

From Figure 10, a change of the distribution was observed every ten seconds. For each section, the mode value appeared near the expected value of the counter, or $N/2$. Therefore, we can conclude that the prototype system is operating as we expect.

6.3 Amount of Hardware

We evaluate the amount of hardware of the system after synthesized, placed and routed. Experimental conditions were almost the same as Section 5.4 with the exception of the version of Vivado, which was 2020.1 rather than 2019.2.

Table 11 summarizes the number of logic elements required for the prototype system. Note that the row DynClk corresponds to a single DynClk core, while the system includes two DynClk cores. The system used about 11% of LUTs and 6% of flip-flops of an XC7A35T FPGA, almost half of which were required for a MicroBlaze processor. The TRNG module used 487 LUTs and 359 flip-flops but they are mostly used by the UART transmitter, which was slightly complicated because of additional functionality for debug. The AXI interface circuit required only tens of logic elements because it only has to receive eleven bits of parameters (ten bits for N and one bit for Pack_EN) from software.

Although we leave it future work to develop a more advanced prototype, we discuss a future outlook for it here. It will integrate MMCMs with a function of DynClk into the TRNG module. Most of the logic elements in the DynClk core are used by its AXI interface circuit, which receives about 200 bits of the values to be written to the MMCM from software. However, in the MMCM-based TRNG, these values can be precalculated and stored in a ROM. The required numbers of LUTs and flip-flops will be greatly reduced by this optimization, in exchange for an increase of RAM blocks.

Table 12: Comparison of recent TRNGs on Xilinx FPGAs.

Type	Target	Area (LUT/FF)	Bit Rate (Mbit/s)
MMCM (This work)	Artix-7	17/18	2.44
DCM [7]	Virtex-5	19/26	0.21
Configurable COSO [15]	Spartan-6	108/39*	3.30
TC-TERO [5]	Artix-7	40/29	1.91
RS Latch [8]	Artix-7	716/974	20.0
Self-timed Ring [16]	Spartan-6	346/256	154

* including self-calibration circuit

7 Related Work

A summary of comparison of the proposed MMCM-based TRNG with other types of recent TRNGs on Xilinx FPGAs is shown in Table 12. There have been some types of TRNGs that achieved small number of logic elements and high throughput at the same time. Our MMCM-based TRNG now became one of them. Considering a risk that some types of TRNGs will come out to be compromised in the future, it is important that there are multiple types of TRNGs that have different operating principles.

As we have explained in Section 2.2, it is possible to obtain clock signals for coherent sampling using configurable (route-selectable) ring oscillators [15]. They have copies of a set of a NAND gate and buffers that compose a ring and replace the buffers with multiplexers. Selection inputs of the multiplexers are used as parameters. This enables to select the route of the ring and, consequently, the frequency of the ring by the parameter. It already has a mechanism of self-calibration to find an appropriate parameter set automatically. However, the probability of getting one from random parameters is relatively low (few percent), which means that it might take long time to complete the self-calibration.

The idea of configurable ring oscillator can be extended to another type of TRNGs called a transition effect ring oscillator (TERO) [5, 20]. It uses an oscillatory behavior of an RS latch that transits from the metastable (forbidden) state to one of the stable states. The number of oscillation will be randomly distributed if the ring is well balanced. A TC-TERO [5] has been proposed as a configurable TERO with some additional optimizations. Although the operating principle of the TERO is totally different from coherent sampling, they have the same weakness of low probability to get an appropriate parameter.

An RS latch-based TRNG [8, 9] also utilizes the metastable state of an RS latch, but it focuses on the final output after the oscillation. Since the output of the RS latch is more biased than the number of oscillation, this type of TRNG places a number of RS latches and XORs their output to ensure enough entropy.

A Self-timed ring (STR) [2, 16] is a ring of inverters and circuit elements called Muller C-gates, which are used in a handshake protocol of an asynchronous circuit. In a STR-based TRNG, outputs of the C-gates are sampled and XORed to generate a random bit.

RS latch-based and STR-based TRNGs have similar advantage and disadvantage: quite high throughput but large logic area. They generate tens or hundreds of Mbit of random numbers per second, but they require hundreds of LUTs and flip-flops. The STR-based TRNG has another disadvantage of low degree of freedom in placement of elements. Hundreds of elements must be carefully placed for its proper operation.

8 Conclusion

In this paper, we proposed an improved true random number generator for Xilinx FPGAs, using MMCMs. We showed that careful selection of MMCM parameters improved both the randomness and the throughput of the TRNG. The bit rate of generation became an order of magnitude larger

than the previous DCM-based TRNG. We also demonstrated the feasibility of dynamic reconfiguration of the parameters by our prototype system.

We are developing a more advanced prototype. We will finally package the outcome of the research into a TRNG IP core handy for FPGA system developers, where an appropriate parameter will be set automatically.

Acknowledgement

We would like to thank Mr. Yuto Hirano, who partially contributed to the prototyping of dynamic reconfiguration.

References

[1] F. Bernard, V. Fischer, and B. Valtchanov. Mathematical Model of Physical RNGs Based on Coherent Sampling. *Tarta Mountains Mathematical Publications*, 45(1):1–14, 2010.

[2] A. Cherkaoui, V. Fischer, A. Aubert, and L. Fesquet. A Self-timed Ring Based True Random Number Generator. In *19th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 99–106, 2013.

[3] Digilent Inc. *axi_dynclk*. `https://github.com/Digilent/vivado-library/tree/master/ip/axi_dynclk`, cited 30 January, 2021.

[4] V. Fischer and M. Drutarovsky. True Random Number Generator Embedded in Reconfigurable Hardware. In *3rd Workshop on Cryptographic Hardware and Embedded Systems*, pages 415–430, 2002.

[5] N. Fujieda. On the feasibility of TERO-based true random number generator on Xilinx FPGAs. In *30th International Conference on Field-Programmable Logic and Applications*, pages 103–108, 2020.

[6] N. Fujieda and S. Takashima. Enhanced use of mixed-mode clock manager for coherent sampling-based true random number generator. In *8th International Symposium on Computing and Networking Workshops*, pages 197–203, 2020.

[7] N. Fujieda, M. Takeda, and S. Ichikawa. An Analysis of DCM-based True Random Number Generator. *IEEE Transaction on Circuits and Systems II: Express Briefs*, 67(6):1109–1113, 2020.

[8] Naoki Fujieda and Shuichi Ichikawa. A latch-latch composition of metastability-based true random number generator for Xilinx FPGAs. *IEICE Electronics Express*, 15(10):20180386:1–20180386:12, 2018.

[9] H. Hata and S. Ichikawa. FPGA implementation of metastability-based true random number generator. *IEICE Transactions on Information & Systems*, E95-D(2):426–436, 2012.

[10] A. P. Johnson, R. S. Chakraborty, and D. Mukhopadyay. An Improved DCM-Based Tunable True Random Number Generator for Xilinx FPGA. *IEEE Transaction on Circuits and Systems II: Express Briefs*, 64(4):452–456, 2017.

[11] W. Killmann and W. Schindler. *A proposal for: Functionality classes for random number generators, version 2.0*. Federal Office for Information Security, 2011.

[12] P. Kohlbrenner and K. Gaj. An embedded true random number generator for FPGAs. In *12th International Symposium on Field Programmable Gate Arrays*, pages 71–78, 2004.

[13] Y. Lao, Q. Tang, C. H. Kim, and K. K. Parhi. Beat Frequency Detector–Based High-Speed True Random Number Generators: Statistical Modeling and Analysis. *ACM Journal on Emerging Technologies in Computing Systems*, 13(1):1–25, 2016.

[14] N. C. Laurenciu and S. D. Cotofana. Low cost and energy, thermal noise driven, probability modulated random number generator. In *2015 IEEE International Symposium on Circuits and Systems*, pages 2724–2727, 2015.

[15] A. Peetermans, V. Rožić, and I. Verbauwhede. A Highly-Portable True Random Number Generator based on Coherent Sampling. In *29th International Conference on Field Programmable Logic and Applications*, pages 218–224, 2019.

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

[17] 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.

[18] J. Tatsukawa. *MMCM and PLL Dynamic Reconfiguration.* Application Note XAPP888 v1.8, Xilinx Inc., 2019.

[19] B. Valtchanov, V. Fischer, and A. Aubert. Enhanced TRNG based on the coherent sampling. In *3rd International Conference on Signals, Circuits and Systems*, pages 1–6, 2009.

[20] M. Varchola and M. Drutarovsky. New high entropy element for FPGA based true random number generators. In *Proc. 12th Workshop on Cryptographic Hardware and Embedded Systems*, pages 351–365, 2010.

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

[22] Xilinx Inc. *7 Series FPGAs Clocking Resources.* User Guide UG472 v1.14, 2018.

[23] Xilinx Inc. *Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics.* Data Sheet DS181 v1.25, 2018.

[24] H. Zhun and C. Hongyi. A truly random number generator based on thermal noise. In *4th International Conference of ASIC*, pages 862–864, 2001.