Optimization of the PLL configuration in a PLL-based TRNG design

Elie Noumon Allini, Oto Petura, Viktor Fischer, Florent Bernard

Optimization of the PLL Configuration in a PLL-based TRNG Design

Elie Noumon Allini, Oto Petura, Viktor Fischer, Florent Bernard
Univ Lyon, UJM Saint-Etienne
Laboratoire Hubert Curien, UMR CNRS 5516
42000 Saint-Etienne
(elie.noumon.allini, oto.petura, fischer, florent.bernard)@univ-st-etienne.fr

***Abstract*—Several recent designs show that the phase locked-loops (PLLs) are well suited for building true random number generators (TRNG) in logic devices and especially in FPGAs, in which PLLs are physically isolated from the rest of the device. However, the setup of the PLL configuration for the PLL-based TRNG is a challenging task. Indeed, the designer has to take into account physical constraints of the hardwired block, when trying to achieve required performance (bit rate) and security (entropy rate per bit). In this paper, we introduce a method aimed at choosing PLL parameters (e.g. input frequency, multiplication and division factors of the PLL) that satisfy hardware constraints, while achieving the highest possible bit rate or entropy rate according to application requirements. The proposed method is fast enough to produce all possible configurations in a short time. Comparing to the previous method based on a genetic algorithm, which was able to find only a locally optimized solution and only for one PLL in tens of seconds, the new method finds exhaustive set of feasible configurations of one- or two-PLL TRNG in few seconds, while the found configurations can be ordered depending on their performance or sensitivity to jitter.**

I. Introduction

Random number generators represent cryptographic primitives that are crucial in most cryptographic applications. They are used to generate confidential cryptographic keys, initialization vectors, nonces or even random masks in side channel attack countermeasures. Knowing that the security of cryptographic constructions relies on the secrecy of the key [1], random numbers used in cryptographic schemes are required to be of excellent statistical quality while being completely unpredictable. This can be achieved in true random number generators (TRNGs), in which the unpredictability is guaranteed by some physical random phenomena.

Modern cryptographic systems are usually implemented in logic devices and contain mostly some oscillator-based TRNG using random jitter of generated clock signals as a source of randomness. In FPGAs, free-running oscillators are implemented in logic area, while PLLs are hardwired in a physically isolated zone, in which the impact of processes running in the logic area is significantly reduced. This is considered to be the main advantage of implementation of PLL-TRNG in FPGAs. Its simple and comprehensive design based on the use of one [2], [3], [4] or two PLLs [5] and availability of the stochastic model [6] represent other important advantages. Last but not least, the use of PLL ensures good stability and repeatability of results in different devices in time, and in large range of temperatures and power supply voltages.

The main difficulty in the PLL-TRNG design is related to the choice of appropriate PLL configurations from a large design space. The chosen configuration should ensure required entropy rate and sufficient output bit rate. Among restricted amount of possible configurations that meet frequency ranges of individual PLL blocks (we call them feasible configurations), only few ensure required entropy and output bit rate to achieve security and performance requirements (we call them suitable configurations).

Authors of a recent survey [7] achieved relatively weak results in the PLL-TRNG design: the entropy rate was mostly insufficient and the output bit rate was quite low. The same team then optimized the design of the generator using a genetic algorithm (GA) [8]. The authors claimed that they significantly reduced the design time and enhanced both entropy rate and bit rate at generator output.

However, since the genetic algorithm finds only the locally optimal solution, which depends on starting values of parameters of the GA selected randomly, it was never sure that the found solution is the best one. Another disadvantage of the use of the GA was that it had limited number of possible input parameters and it was therefore suitable only for the PLL-TRNG using just one PLL.

In this paper, we propose another method which produces all usable configurations of a PLL-based TRNG. We opted for the generation of an exhaustive list of configurations instead of trying to find the best one because the notion of the best configuration is application dependent.

Our contribution:

- We propose and verify efficiency of the new algorithm finding all suitable configurations of a PLL-based TRNG.
- We show that the new method can find configurations for TRNGs based either on one or two PLLs.

The paper is organized as follows. In Section II, we introduce the principle of the PLL-based TRNG, and explain the problem to be solved. In Section III, we describe the proposed

[1]The article was published in the proceedings of the DATE 2018 conference. The published version is:
E. Noumon Allini, O. Petura, V. Fischer and F. Bernard, "Optimization of the PLL configuration in a PLL-based TRNG design," 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 2018, pp. 1265-1270. doi: 10.23919/DATE.2018.8342209

method aimed at finding suitable configurations of the PLL-based TRNG. In Section IV, we present obtained results and compare them with those provided by the GA. We conclude the paper in Section V.

II. General Principle of the PLL-based TRNG

The PLL-based TRNG is based on a coherent sampling principle: a clock signal clk_1 is sampled using another clock signal clk_0 in a D flip-flop (D-FF). Clock signals clk_1 and clk_0 can be generated in one [2] or two [5] PLLs as presented in Fig. 1 a) and b), respectively.

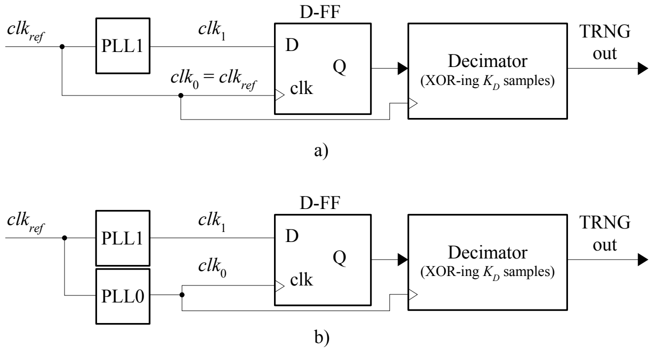

Fig. 1. PLL-based TRNGs using one (a) or two (b) PLLs

Thanks to the use of PLLs, frequencies of clk_1 and clk_0 are mutually related:

$$\frac{f_1}{f_0} = \frac{K_M}{K_D}, \tag{1}$$

where K_M and K_D are integer values representing frequency multiplication and division factors, which depend on PLL configuration(s).

Note that we do not consider the fractional mode of PLL operation, in which multiplication and division factors can obtain real values, since signals generated in this mode can feature a deterministic jitter caused by the operation of the Delta sigma modulator [9], [10].

Consequently, for the one-PLL-TRNG from Fig. 1 a), $K_M = K_{M_1}$ and $K_D = K_{D_1}$. In the case of two-PLL TRNG from Fig. 1 b), TRNG multiplication and division factors are defined as follows:

$$K_M = K_{M_1} \cdot K_{D_0}, \tag{2}$$

$$K_D = K_{M_0} \cdot K_{D_1}. \tag{3}$$

The use of two PLLs has two advantages: 1) it makes the PLL-TRNG design more flexible by making the practical ranges of K_M and K_D much larger and consequently, by increasing ranges of bit and entropy rates that can be attained; 2) it reduces significantly autocorrelation between output bits. The higher price of this solution can be reduced in most of cases by sharing PLL0 with other system blocks.

The output signal of the D-FF from Fig. 1 features pseudo-random pattern with a period $T_Q = K_D/f_0 = K_M/f_1$. This pattern is removed by XOR-ing K_D samples in the decimator, which follows the sampler D-FF [2]. The decimator output then represents the pattern-free random output of the PLL-TRNG.

The bit rate of the PLL-TRNG is defined as follows [2]:

$$R = \frac{f_0}{K_D} = \frac{f_1}{K_M}. \tag{4}$$

The entropy rate per bit at generator output depends on parameters of the jitter and on parameters of the generator, which are characterized by its sensitivity to the jitter [7]:

$$S = f_0 \cdot K_M = f_1 \cdot K_D. \tag{5}$$

The objective of the designer is to set up parameters of the PLL(s) in order to obtain (depending on application requirements) sufficient entropy and bit rate at generator output. When setting up parameters of the PLL, the designer must fulfill hardware requirements of the hardwired PLL circuitry, which will be discussed in the following section.

A. General Structure of the PLL and Its Configuration

The general structure of the PLL is depicted in Fig. 2. The PLL blocks are usually hardwired and the designer can configure the PLL by setting up directly or indirectly (e.g. by selecting the targeted output frequency) programmable dividers N, M, P_{VCO}, C, while strictly respecting relationships between individual blocks, and namely permitted ranges of their input/output frequencies and permitted ranges of programmable dividers, which are specified in the technical documentation of the PLL block.

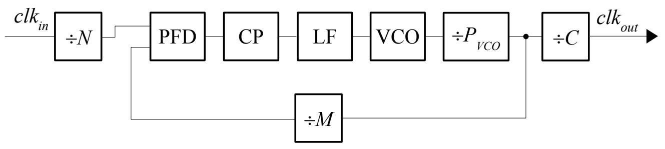

Fig. 2. General internal structure of the PLL (N: input frequency division factor, M: multiplication factor, PFD: phase-frequency detector, CP: charge pump, LF: loop filter, VCO: voltage-controlled oscillator, P_{VCO}: post-VCO frequency divider, C: output frequency division factor).

Namely, the output frequency f_{PFD} of the phase-frequency detector depends on the input frequency as follows:

$$f_{PFD} = \frac{f_{in}}{N}, \tag{6}$$

the VCO frequency is:

$$f_{VCO} = f_{PFD} \cdot M \cdot P_{VCO}, \tag{7}$$

and the output frequency is:

$$f_{out} = \frac{f_{PFD} \cdot M}{C}. \tag{8}$$

The use of the PLL ensures a rational relation between frequencies of the input and output signals. More precisely, we have:

$$K_D \cdot f_{out} = K_M \cdot f_{in}, \tag{9}$$

where:

$$K_M = M \quad \text{and} \quad K_D = N \cdot C. \tag{10}$$

In the case of PLL-based TRNGs from Fig. 1.b), frequencies f_0 and f_1 are related in the following way:

$$K_M \cdot f_0 \; = \; K_D \cdot f_1, \tag{11}$$

where:

$$K_M \; = \; K_{M_1} \cdot K_{D_0} \; = \; M_1 \cdot N_0 \cdot C_0, \tag{12}$$

and

$$K_D \; = \; K_{M_0} \cdot K_{D_1} \; = \; M_0 \cdot N_1 \cdot C_1. \tag{13}$$

Note that if only one PLL is used, $N_0 = M_0 = C_0 = 1$ in Equations (12) and (13). For this reason, we will next consider only the general case in which two PLLs are used.

B. Problem to Solve

The problem we address in this paper can be split into two steps:

1) finding all the feasible configurations of PLLs that fulfill hardware constraints given in their technical documentation,
2) among these feasible configurations, finding suitable configurations that satisfy performance and/or security criteria of the targeted application (e.g. entropy rate specified in AIS-31 recommendations [11]).

The first step is to find, for each $i \in \{0, 1\}$, the values of M_i, N_i, C_i satisfying Equations (6) to (13), that are constrained by the following inequalities:

$$\left\{ \begin{array}{rcl} M_{min} & \leqslant \; M_i \; \leqslant & M_{max} \quad (14) \\ N_{min} & \leqslant \; N_i \; \leqslant & N_{max} \quad (15) \\ C_{min} & \leqslant \; C_i \; \leqslant & C_{max} \quad (16) \\ f_{PFD_{min}} & \leqslant \; f_{PFD_i} \; \leqslant & f_{PFD_{max}} \quad (17) \\ f_{VCO_{min}} & \leqslant \; f_{VCO_i} \; \leqslant & f_{VCO_{max}} \quad (18) \\ f_{out_{min}} & \leqslant \; f_{out_i} \; \leqslant & f_{out_{max}}, \quad (19) \end{array} \right.$$

where the numbers M_{min}, N_{min}, C_{min}, M_{max}, N_{max} and C_{max} are positive integers, and $f_{PFD_{min}}$, $f_{PFD_{max}}$, $f_{VCO_{min}}$, $f_{VCO_{max}}$, $f_{out_{min}}$ and $f_{out_{max}}$ are positive real numbers, which represent hardware limitations of the PLL given by manufacturers.

In our experiments, we considered PLL-based TRNGs implemented in three different FPGA families: Intel Cyclone V [9], Xilinx Spartan-6 [10], and Microsemi SmartFusion®2 FPGAs [12]. Table I gives hardware restrictions for selected FPGA families.

Once the configurations that are feasible in selected hardware are found, the designer needs to filter out those, which do not attain required entropy rate and/or sufficient output bit rate depending on application objectives and constraints, according to Equations (4) and (5).

The stochastic model of the PLL-based TRNGs [6] can be used to provide thresholds for these parameters, which are necessary to obtain suitable configurations out of all feasible configurations.

TABLE I
PLL SPECIFICATIONS OF SELECTED FPGA FAMILIES

Parameter	**Cyclone V**		**Spartan-6**		**SmartFusion®2**	
	Min	**Max**	**Min**	**Max**	**Min**	**Max**
f_{ref}(MHz)	5	500	19	540	1	200
P_{VCO_i}	1	2	1	1	1	32
N_i	1	512	1	52	1	16384
M_i	1	512	1	64	1	4194304
C_i	1	512	1	128	1	255
f_{PFD_i}(MHz)	5	325	19	500	1	200
f_{VCO_i}(MHz)	600	1300	400	1080	500	1000
f_{out_i}(MHz)	0	460	3.125	400	20	400

III. Research of PLL-TRNG Configurations

As explained in the previous section, we proceed in two steps: the search for all feasible PLL configurations is followed by the search for configurations giving acceptable results.

To find all feasible configurations, the first naive idea could be to perform an exhaustive search on the PLL design space and to save configurations that satisfy hardware constraints (Inequalities (14) to (19)). Due to the number of possible configurations, it is not reasonable to process this way, since the algorithm running time could reach several years.

The proposed improvement of the optimization algorithm consists in a sequential search of all values of M_i, N_i and C_i that lead to feasible configurations of the two PLLs.

Indeed, from Equation (6), we can see that for the given value f_{ref}, N_i is the only variable for which the value has to be found. Also, from Equations (6) and (7), we can express f_{VCO_i} as a function of f_{ref}, N_i and M_i. So for given values of f_{ref} and N_i, we look for all possible values of M_i. In the same way, for given values of f_{ref}, N_i and M_i, Equation (8) helps to find the values of C_i. This process is repeated for each value of P_{VCO_i}.

A. Research of All Feasible PLL Configurations

We first set a value for f_{ref} between $f_{ref_{min}}$ and $f_{ref_{max}}$ defined in Table I. Then, for each value of P_{VCO_i}, we proceed as follows:

1) From Equation (6), $N_i = \dfrac{f_{ref}}{f_{PFD_i}}$. When combined with Inequalities (15) and (17), we can write:

$$N_{min_i} \; \leqslant \; N_i \; \leqslant \; N_{max_i}, \tag{20}$$

where

$$N_{min_i} \; = \; \max\left(N_{min}, \left\lceil \frac{f_{ref}}{f_{PFD_{max}}} \right\rceil\right) \tag{21}$$

and

$$N_{max_i} \; = \; \min\left(N_{max}, \left\lfloor \frac{f_{ref}}{f_{PFD_{min}}} \right\rfloor\right). \tag{22}$$

Note that $N_{max_i} - N_{min_i} \leqslant N_{max} - N_{min}$, showing that the range for searching N_i is reduced.

2) Inequality (20) provides a new range of possible values of N_i. For a value of N_i chosen in this range, we can express a new range of possible values for M_i. Indeed, from Equations (6) and (7), we can write:

$$M_i \ = \ \frac{N_i \cdot f_{VCO_i}}{f_{ref} \cdot P_{VCO_i}}. \tag{23}$$

With the assumption that N_i, f_{ref} and P_{VCO_i} are given, f_{VCO_i} is the only variable value. From Inequalities (14) and (18), it follows:

$$M_{min_i} \ \leqslant \ M_i \ \leqslant \ M_{max_i}, \tag{24}$$

where

$$M_{min_i} \ = \ \max\left(M_{min}, \left\lceil \frac{N_i \cdot f_{VCO_{min}}}{f_{ref} \cdot P_{VCO_i}} \right\rceil\right) \tag{25}$$

and

$$M_{max_i} \ = \ \min\left(M_{max}, \left\lfloor \frac{N_i \cdot f_{VCO_{max}}}{f_{ref} \cdot P_{VCO_i}} \right\rfloor\right). \tag{26}$$

3) Inequality (24) provides a smaller set of possible values of M_i, for some a priori chosen value of N_i. For chosen values of N_i and M_i, Equations (6) and (8) give $C_i = \dfrac{f_{ref} \cdot M_i}{N_i \cdot f_i}$. Knowing that all values are given except for f_{out_i} which ranges from $f_{out_{min}}$ to $f_{out_{max}}$, it follows:

$$C_{min_i} \ \leqslant \ C_i \ \leqslant \ C_{max_i}, \tag{27}$$

where

$$C_{min_i} \ = \ \max\left(C_{min}, \left\lceil \frac{f_{ref} \cdot M_i}{N_i \cdot f_{out_{max}}} \right\rceil\right) \tag{28}$$

and

$$C_{max_i} \ = \ \min\left(C_{max}, \left\lfloor \frac{f_{ref} \cdot M_i}{N_i \cdot f_{out_{min}}} \right\rfloor\right). \tag{29}$$

Thanks to Inequalities (20), (24) and (27) we are guaranteed to obtain only feasible configurations and all of them in a reasonable amount of time (several hours instead of years).

This search process can be improved in the context of PLL-based TRNG.

B. Research of Suitable PLL Configurations

A feasible configuration for a PLL-based TRNG is any one found according to the process described in section III-A. Moreover, in the PLL-based TRNG [2], K_D must be odd and co-prime with K_M. Even though the coprimality of K_M and K_D has to be checked by the Euclidean algorithm ($\gcd(K_M, K_D) = 1$), it is possible to skip the parity test of K_D. Indeed, K_D ought to be odd, and from Equation (13), this implies that M_0, N_1 and C_1 should all be odd. Values of M_0, N_1 and C_1 can then be looked for by a step of 2, starting with the smallest odd number in each range. This consideration reduces by a factor of 2, for each of these parameters, the number of values that have to be checked, and it thus speeds up the algorithm.

Furthermore, for security reasons, the sensitivity to the jitter must be sufficiently high. For performance reasons, we want the output bit rate to be as high as possible. Then, using application requirements and Equations (4) and (5), values of K_M and K_D must be bounded. In order to give more flexibility to the designer, we introduce s_M and s_D as their respective upper bounds. Thus:

$$0 \ \leqslant \ K_M \ \leqslant s_M \quad \text{and} \quad 0 \ \leqslant \ K_D \ \leqslant s_D. \tag{30}$$

Using Equations (12) and (13), it follows:

$$0 \ \leqslant \ C_0 \ \leqslant \frac{s_M}{M_1 \cdot N_0} \quad \text{and} \quad 0 \ \leqslant \ C_1 \ \leqslant \frac{s_D}{M_0 \cdot N_1}. \tag{31}$$

We can thus define smaller upper bounds C'_{max_0}, C'_{max_1} to C_0 and C_1, respectively, by:

$$C'_{max_0} \ = \ \left\lfloor \min\left(C_{max}, \frac{f_{ref} \cdot M_0}{N_0 \cdot f_{out_{min}}}, \frac{s_M}{M_1 \cdot N_0}\right) \right\rfloor \tag{32}$$

and

$$C'_{max_1} \ = \ \left\lfloor \min\left(C_{max}, \frac{f_{ref} \cdot M_1}{N_1 \cdot f_{out_{min}}}, \frac{s_D}{M_0 \cdot N_1}\right) \right\rfloor. \tag{33}$$

These new considerations reduce significantly the number of possible configurations of both PLLs. We thus have a smaller subset among all the feasible configurations suitable for the PLL-based TRNGs implemented in the selected FPGA family.

The whole search process is resumed in Algorithm 1.

IV. Results and Discussion

A. Implementation Results

To evaluate the speed and efficiency of the search process, according to hardware limitations specified by manufacturers, we implemented our algorithm in C language. The algorithm takes only two inputs: the reference frequency f_{ref} and the FPGA family, for which we want to generate configurations.

We ran the algorithm on an HP Compaq 6005 Pro MT PC AMD Athlon^{TM} II X2 B24 Processor. When fed with a reference frequency of $125\,\text{MHz}$ (we selected the same reference frequency for all families in order to get comparable results), our algorithm found all the usable configurations in less than 10 seconds for each of the above mentioned FPGA families.

We decided to generate all the usable configurations, instead of trying to find the best configuration, because we wanted to provide a general tool that designers can use to generate a suitable PLL-TRNG configuration. Indeed, the notion of best configuration is closely dependent on the target application.

According to the application the TRNG is designed for, parameters that have to be optimized may change. For example, when the generator is expected to achieve a fair level of security with the highest output bit rate, the optimization process will search for the highest output bit rate which guarantees a suitable security level that reaches the requirements. However, for a high level security application, the optimization process would be finding configurations that ensures the highest entropy, regardless or not the output bit rate.

```
compute $N_{min_0}$ from Equation (21)
compute $N_{max_0}$ from Equation (22)
$N_{min_1} \leftarrow$ round_up_to_odd($N_{min_0}$)
$N_{max_1} \leftarrow N_{max_0}$
configs $\leftarrow$ MAKEEMPTYLIST()
for all $P_{VCO_0}$ in Pvco_vals do
  for all $P_{VCO_1}$ in Pvco_vals do
    for $N_1 = N_{min_1}$ to $N_{max_1}$ by 2 do
      compute $M_{min_1}$ from Eq. (25)
      compute $M_{max_1}$ from Eq. (26)
      for $N_0 = N_{min_0}$ to $N_{max_0}$ do
        compute $M_{min_0}$ from Eq. (25)
        $M_{min_0} \leftarrow$ round_up_to_odd ($M_{min_0}$)
        compute $M_{max_0}$ from Eq. (26)
        for $M_0 = M_{min_0}$ to $M_{max_0}$ by 2 do
          compute $C_{min_0}$ from Eq. (28)
          for $M_1 = M_{min_1}$ to $M_{max_1}$ do
            compute $C_{min_1}$ from Eq. (28)
            $C_{min_1} \leftarrow$ round_up_to_odd ($C_{min_1}$)
            compute $C'_{max_0}$ from Eq. (32)
            compute $C'_{max_1}$ from Eq. (33)
            for $C_1 = C_{min_1}$ to $C'_{max_1}$ by 2 do
              compute $K_D$ from Eq. (13)
              for $C_0 = C_{min_0}$ to $C'_{max_0}$ do
                compute $K_M$ from Eq. (12)
                if $\gcd(K_M, K_D) = 1$ then
                  compute $f_0$ and $f_1$ from Eq. (8)
                  compute $R$ from Eq. (4)
                  compute $S$ from Eq. (5)
                  save into configs, values of $f_{ref}$,
                    $P_{VCO_0}$, $P_{VCO_1}$, $M_0$, $N_0$, $C_0$, $M_1$,
                    $N_1$, $C_1$, $f_0$, $f_1$, $K_M$, $K_D$, $R$, $S$
                end if
              end for
            end for
          end for
        end for
      end for
    end for
  end for
end for
```

Algorithm 1. PLL configuration search algorithm for the PLL-TRNG

The timing analysis of our PLL-TRNG design showed the maximum supported clock frequency of around 250 MHz on all three tested FPGA families. This frequency is the limiting frequency of the logic resources in our design. To comply with these limits, we decreased the PLL maximum output frequency from manufacturers' limit to 250 MHz in order to exclude the PLL configurations to fast for our circuitry.

The minimum value of sensitivity to jitter can be determined from the required entropy rate using stochastic model presented in [6]. For the Shannon entropy rate required by AIS31 [11] (H_1=0.997), the minimum sensitivity to jitter obtained from the model must be higher than $0.09\,\mathrm{ps}^{-1}$. So we limited our search to configurations satisfying this security condition.

The search returned 188 suitable configurations out of 389 853 (0.048%) feasible ones for Intel Cyclone V, 8 out of 89 025 (0.0089%) for Xilinx Spartan-6, and 9 976 out of 2 339 412 (0.426%) for SmartFusion®2.

The number of configurations satisfying security conditions is smaller than 1 % of the total number of feasible configurations (even with the additional output frequency constraint mentioned above) for every FPGA family tested, so the manual search is nearly impossible.

Table II presents three representative configurations for each FPGA family: the one with the best bit rate R, best sensitivity S, and best product $R \cdot S$. While for Cyclone V and Spartan-6 families we could select three different configurations depending on optimization criteria, one configuration (out of seven) in SmartFusion®2 was the best regarding all of them.

To validate the quality of the chosen configurations, we ran the AIS31 statistical test suite on output bit sequences of the TRNG with given configurations. Since the sensitivity limit was chosen according to the stochastic model, it is not surprising that outputs of all the configurations passed the statistical tests.

B. Comparison of Found One-PLL Configurations with Results of the GA

In order to compare our method with the one based on the genetic algorithm, we implemented a version of our algorithm which uses only one PLL. For each of the three FPGA families, we ran this algorithm with the reference frequency recorded in [8, Table II]. Table III presents the obtained results.

As could be expected, the proposed algorithm found all the configurations provided by the GA. However, as can be seen in Table III, it also found better configurations ensuring higher output bit rate or jitter sensitivity. It thus gives the designer a very efficient tool to find the best configuration, which fulfills hardware constraints and satisfies high security requirements in the given true random number generation context.

V. Conclusion

In this paper, we have introduced a method for generating all suitable PLL-TRNG configurations out of all feasible PLL configurations in a TRNG design based on one or two PLLs. This solves the problem of finding configurations of a PLL-TRNG in a most efficient way and also in the case, in which the published genetic algorithm could not be applied because of high number of variables. We also showed that in the case of a one-PLL-TRNG, the algorithm finds all configurations found by the genetic algorithm, but also configurations providing higher output bit rate or jitter sensitivity, than the previous ones.

Furthermore, contrary to the genetic algorithm approach, our method is fast enough to generate all suitable configurations within seconds for both one-PLL and two-PLL designs. This approach guarantees the global optimality of found results as all feasible configurations are obtained and all suitable configurations are selected from them according to specific constraints of the application.

TABLE II
TWO-PLL-TRNG CONFIGURATIONS FOR SENSITIVITY $S > 0.09$ ps^{-1}

Config.	f_{ref} (MHz)	P_{VCO_0}	P_{VCO_1}	M_0	N_0	C_0	M_1	N_1	C_1	f_0 (MHz)	f_1 (MHz)	K_M	K_D	R (Mbit$/s$)	S (ps^{-1})	$R \cdot S$
Intel Cyclone V																
Highest S	125	1	1	7	1	4	113	19	3	218.75	247.807	452	399	0.548	0.0988	0.0542
Highest R	125	2	1	43	11	2	17	3	3	244.318	236.111	374	387	0.631	0.0913	0.0577
Highest $R \cdot S$	125	1	1	19	2	5	41	7	3	237.5	244.047	410	399	0.595	0.0973	0.0579
Xilinx Spartan-6																
Highest S, R, and $R \cdot S$	125	1	1	43	5	5	17	3	3	215	236.11	425	387	0.555	0.0913	0.0507
Microsemi SmartFusion®2																
Highest S	125	1	1	7	1	4	113	19	3	218.75	247.807	452	399	0.548	0.098	0.054
Highest R	125	1	4	29	5	3	25	13	1	241.66	240.384	375	377	0.641	0.090	0.058
Highest $R \cdot S$	125	1	4	23	3	4	33	17	1	239.58	242.647	396	391	0.612	0.094	0.058

TABLE III
COMPARISON OF ONE-PLL-TRNG CONFIGURATIONS FOUND BY THE PROPOSED ALGORITHM WITH THOSE FOUND USING THE GA

	f_{ref} (MHz)	P_{VCO}	M	N	C	f_1 (MHz)	K_M	K_D	R (Mbit$/s$)	S (ps^{-1})
	Intel Cyclone V									
Best configuration from [8]	350	1	131	37	3	413	131	111	3.15	0.045
Configuration found in this paper	350	1	136	37	3	428.82	136	111	3.15	**0.047**
	Xilinx Spartan-6									
Best configuration from [8]	430	1	47	21	5	192	47	105	4.095	0.020
Configuration found in this paper	430	1	47	21	3	320.79	47	63	**6.82**	0.020
	Microsemi SmartFusion®2									
Best configuration from [8]	200	2	216	127	1	340	216	127	1.574	0.043
Configuration found in this paper	200	2	253	127	1	398.425	253	127	1.574	**0.05**

Last but not least, a stochastic model can be used to set a minimum threshold for the sensitivity in order to guarantee a sufficient entropy rate at the TRNG output. This threshold has been taken as a constraint and has been used to get suitable PLL configurations (less than 1 % of all feasible configurations) that fulfill this important security requirement. The designer can then sort the configurations in order of their preference while still satisfying the given security level, knowing that for some security levels sufficient bit rate may not exist.

ACKNOWLEDGMENT

This work has received funding from the French DGA under grant agreement No 16810066 and the European Union's Horizon 2020 in the framework of the project HECTOR under grant agreement No 644052.

REFERENCES

[1] A. Kerckhoffs, "La cryptographie militaire," *Journal des Sciences Militaires*, vol. IX, pp. 5–38, January 1883.

[2] V. Fischer and M. Drutarovsky, "True random number generator embedded in reconfigurable hardware," in *Cryptographic Hardware and Embedded Systems - CHES 2002*, ser. LNCS, vol. 2523, Redwood Shores, CA, USA. Springer Verlag, 2002, pp. 415–430. [Online]. Available: http://www.springerlink.com/content/00veem7fjd2ejaqj/fulltext.pdf

[3] S. C. Bagal, V. V. Deotare, D. V. Padole, and S. C. Bagal, "Generation of true random number using analog phase locked loop," in *International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE)*, 2015, pp. 1913–1916.

[4] C. Liu and J. McNeill, "A digital-PLL-based true random number generator," in *2005 PhD Research in Microelectronics and Electronics*, vol. 1, 2005, pp. 113–116.

[5] V. Fischer, M. Drutarovsky, M. Simka, and N. Bochard, "High performance true random number generator in Altera stratix FPLDs," in *Field Programmable Logic and Application: 14th International Conference, FPL 2004, Leuven, Belgium, August 30-September 1, 2004. Proceedings*. Springer, 2004, pp. 555–564. [Online]. Available: http://www.springerlink.com/content/mx45y5j24ng3kx6q/fulltext.pdf

[6] F. Bernard, V. Fischer, and B. Valtchanov, "Mathematical model of physical rngs based on coherent sampling," *Tatra Mountains Mathematical Publications*, vol. 45, no. 1, pp. 1–14, 2010. [Online]. Available: http://tatra.mat.savba.sk/Full/45/01be-f-v.pdf

[7] 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, FPL '16, Lausanne, Switzerland*, Aug. 2016.

[8] O. Petura, U. Mureddu, N. Bochard, and V. Fischer, "Optimization of the PLL Based TRNG Design Using the Genetic Algorithm," in *IEEE International Symposium on Circuits and Systems, ISCAS*, 2017, pp. 2202–2205.

[9] Altera, *Cyclone V Device Datasheet (CV51002)*, 2015. [Online]. Available: www.altera.com

[10] Xilinx, *Spartan-6 FPGA Clocking Resources (UG382)*, 2015. [Online]. Available: www.xilinx.com/support/documentation/user_guides/ug382.pdf

[11] W. Killmann and W. Schindler, "A proposal for: Functionality classes for random number generators," 2011. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_Functionality_classes_for_random_number_generators_e.pdf?__blob=publicationFile

[12] Microsemi, *SmartFusion2 and IGLOO2 Clocking Resources (UG0449)*, 2015. [Online]. Available: www.microsemi.com/document-portal/doc_view/132012-ug0449-smartfusion2-and-igloo2-clocking-resources-user-guide