A Highly-Portable True Random Number Generator based on Coherent Sampling

ADRIAAN PEETERMANS, VLADIMIR ROŽIĆ, and INGRID VERBAUWHEDE, imec-COSIC, KU Leuven, Belgium

True Random Number Generators (TRNGs) are indispensable in modern cryptosystems. Unfortunately, in order to guarantee high entropy of the generated numbers, many TRNG designs require a complex implementation procedure, often involving manual placement and routing. In this work, we introduce, analyse and compare three dynamic calibration mechanisms for the Coherent Sampling Ring Oscillator based TRNG (COSO-TRNG): *GateVar*, *WireVar* and *LUTVar*, enabling easy integration of the entropy source into complex systems. The TRNG setup procedure automatically selects a configuration that guarantees the security requirements. In the experiments, we show that two out of the three proposed mechanism are capable of assuring correct TRNG operation even when an automatic placement is carried out and when the design is ported to another FPGA family. We generated random bits on both a *Xilinx Spartan 7* and a *Microsemi SmartFusion2* implementation that, without post processing, passed the AIS-31 statistical tests at a throughput of 4.65 Mbit/s and 1.47 Mbit/s respectively.

CCS Concepts: • **Hardware → Reconfigurable logic applications**; • **Security and privacy → Embedded systems security**.

Additional Key Words and Phrases: Security, Entropy, TRNG, AIS-31, NIST SP 800-90B

ACM Reference Format:
Adriaan Peetermans, Vladimir Rožić, and Ingrid Verbauwhede. 2020. A Highly-Portable True Random Number Generator based on Coherent Sampling. 1, 1 (July 2020), 19 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

True Random Number Generators (TRNGs) are used in cryptographic applications to create random keys, parameters in challenge-response protocols, padding values and masks. Implementing TRNGs on Field-Programmable Gate Arrays (FPGAs) is challenging due to limited TRNG-specific resources and techniques available to the designer. Due to the availability of only digital resources in FPGAs, these TRNGs are usually based either on the unpredictability of metastable memory elements [5, 24] or on the timing jitter present in free-running oscillators [2, 22].

The output of a TRNG should pass statistical tests, such as NIST SP 800-22 [6], NIST SP 800-90B [20] or FIPS 140-2 [14]. However, simply passing statistical tests is not sufficient to guarantee the security of a TRNG. According to the AIS-31 [7] and NIST SP 800-90B [20] recommendations, the security of a TRNG should be justified by a stochastic model of the entropy source.

From an implementation aspect, TRNG designers need to consider the feasibility of the entropy source on an FPGA, portability across different FPGA families and vendors, design constraints, and

Authors' address: Adriaan Peetermans, adriaan.peetermans@esat.kuleuven.be; Vladimir Rožić, vladimir.rozic@esat.kuleuven.be; Ingrid Verbauwhede, ingrid.verbauwhede@esat.kuleuven.be, imec-COSIC, KU Leuven, Kasteelpark Arenberg 10, Leuven-Heverlee, Belgium, B-3001.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2020 Association for Computing Machinery.
XXXX-XXXX/2020/7-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

design effort. Unlike most digital designs, TRNGs usually require some manual setup or placement and routing constraints. For example, designs based on Self-Timed Ring oscillators (STRs) [4], and delay chains [18, 26] require placement constraints that have to be set up for each FPGA family, thus limiting the portability of these designs. In addition, some entropy sources [22] don't work correctly on all locations on an FPGA. Therefore, a search procedure is required for each individual device until a suitable placement is found.

This work focuses on the COherent Sampling ring Oscillator based TRNG (COSO-TRNG) [8]. A stochastic model of this entropy source was first developed by [3] for Phase-Locked Loop (PLL) based implementations. A more general stochastic model, proposed in [27], doesn't specify the source of the random jitter, as this model is based on the period difference between two oscillators of any type.

The existing COSO-TRNG implementations can achieve a throughput of the order of 1 Mbit/s, requiring only minimal chip area. The entropy source consists of a structure that generates two oscillating signals with similar periods. In practice, a generic way of creating these two signals is by using two identically designed Ring Oscillators (ROs). Process and interconnect delay variations, however, make it very challenging to match the periods of the two ROs in an FPGA [16]. For this reason, a search procedure has to be applied until two well matched ROs are found. This high effort renders the COSO-TRNG implementation unpractical as this procedure has to be repeated for every device, even from the same FPGA family.

In this paper, we propose a set of highly-portable, easy-to-use architectures of a COSO entropy source without requiring any device-specific blocks such as PLLs, carry-chains or DSPs, without any placement and routing constraints, and without the need for a manual search procedure. The main contributions are the following:

- We compare three entropy source constructions with reconfigurable ROs, who enable matching two oscillating periods with a precision of a few picoseconds. All three of the proposed sources use only LookUp Tables (LUTs) without requiring device-specific resources, making the design portable across different families and vendors.
- We provide experimental confirmation that this matching is possible even without placement and routing constraints. The Verilog source code of this TRNG is open source and publicly available[1].
- We design a control circuit that monitors the TRNG health, dynamically adjusts the matching conditions, and notifies the user if suitable matching could not be obtained.

This paper extends the work presented in [15] by analysing two additional configurable RO architectures and presenting experimental results from a more modern FPGA platform.

2 BACKGROUND

The Coherent Sampling (CS) designs use two oscillators in the entropy source. Their periods should satisfy one of the two conditions: either the ratio of the periods is equal to a known rational fraction, which is the case in PLL-based designs, or this ratio is tuned to a value very close to one, which is the case when they are generated by ROs. Here, we focus on the latter case because we want to avoid using any device-specific components such as PLLs. The notation used in this text is summed up in Table 1.

2.1 Stochastic model

Figure 1 shows the architecture of the COSO-TRNG. A Data Flip-Flop (DFF) performs the sampling and outputs a low frequency beat signal (S_{beat}). The period length of this signal is measured using

[1]https://github.com/KULeuven-COSIC/COSO-TRNG

Table 1. Notation.

Symbol	Definition
$\mathrm{E}[\cdot]$	Expected value, defined as: $\mathrm{E}[X] = \int_{-\infty}^{\infty} x f_X(x) dx$ for a continuous random variable X with probability density function: $f_X(x)$ or $\mathrm{E}[X] = \sum_{i=1}^{\infty} x_i p_i$ for a discrete random variable X with probability masses: $Pr(X = x_i) = p_i$.
$\mathrm{Var}[\cdot]$	Variance, defined as: $\mathrm{Var}[X] = \mathrm{E}[(X - \mathrm{E}[X])^2]$.
$\mathcal{N}(\mu, \sigma^2)$	The normal probability distribution, describing a continuous random variable X with probability density function: $f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$. It can be shown that: $\mathrm{E}[X] = \mu$ and $\mathrm{Var}[X] = \sigma^2$.

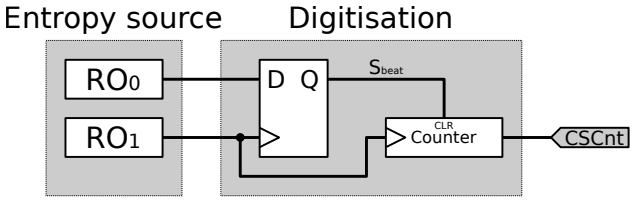

Fig. 1. Architecture of the COSO-TRNG.

a counter clocked by the sampling signal, reset every period of S_{beat}. The output of this counter ($CSCnt$) is a discrete random variable due to the independent random jitter that is present in both oscillators. According to the model in [27], the mean and the variance of this random variable are:

$$\mathrm{E}[CSCnt] = \frac{\mathrm{E}[T_{RO0}]}{\mathrm{E}[\Delta]}, \tag{1}$$

$$\mathrm{Var}[CSCnt] = \mathrm{E}[CSCnt] \frac{\mathrm{Var}[\Delta]}{\mathrm{E}[\Delta]^2}. \tag{2}$$

Where T_{RO0} and T_{RO1} denote the oscillator period lengths. Mean and variance of the period difference, Δ, are given by:

$$\mathrm{E}[\Delta] = |\mathrm{E}[T_{RO1}] - \mathrm{E}[T_{RO0}]|, \tag{3}$$

$$\mathrm{Var}[\Delta] = \mathrm{Var}[T_{RO0}] + \mathrm{Var}[T_{RO1}]. \tag{4}$$

The Least Significant Bit (LSB) can now be used from the output of the counter to generate random bits.

2.2 Entropy rate optimisation

To calculate the entropy per generated bit, knowledge of the exact distribution of $CSCnt$ is required. Derivation of an analytical expression for this distribution starting from the stochastic model is a difficult problem. Yang et. al. in [27] mitigate this problem by assuming that the resulting distribution is Gaussian. This assumption reduces the stochastic model to the elementary RO-TRNG model presented by Ma et. al. in [10]. However, our results show that the approximation used by Yang et. al. [27] doesn't hold in case when the oscillators are very well matched, i.e. when the period difference is the same order of magnitude as the jitter. For this reason, we estimate the distribution of $CSCnt$ by running an event-driven simulation of the COSO-TRNG.

Variables for this model are:

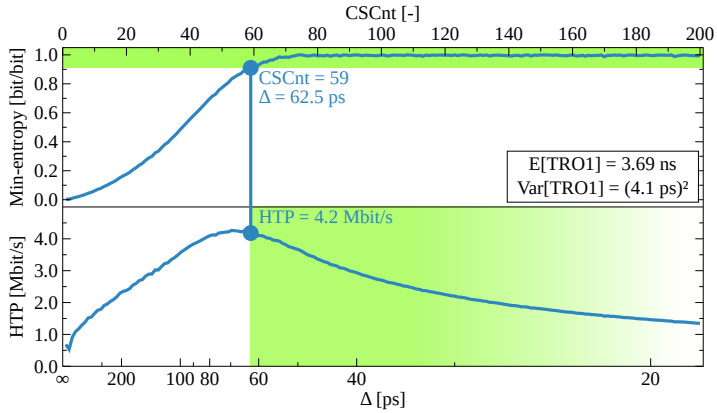

Fig. 2. Estimated min-entropy and HTP versus E[Δ] and E[*CSCnt*] for a *Spartan 7* implementation.

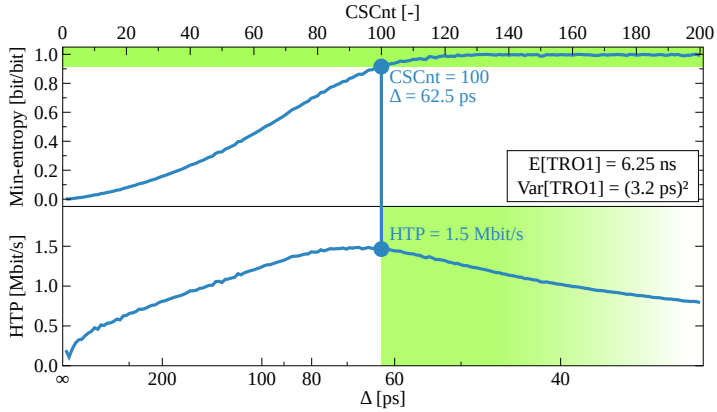

Fig. 3. Estimated min-entropy and HTP versus E[Δ] and E[*CSCnt*] for a *SmartFusion2* implementation.

- The sampled signal's average period length: $\mathrm{E}[T_{RO0}]$
- The average period length difference: $\mathrm{E}[\Delta]$
- The jitter strength: $\mathrm{Var}[T_{RO0}]/\mathrm{E}[T_{RO0}]$

In the model, we assumed both random variables T_{RO0} and T_{RO1} to be independent and Gaussian distributed:

$$T_{RO0} \sim \mathcal{N}(\mathrm{E}[T_{RO0}], \mathrm{Var}[T_{RO0}]), \tag{5}$$

$$T_{RO1} \sim \mathcal{N}(\mathrm{E}[T_{RO1}], \mathrm{Var}[T_{RO1}]). \tag{6}$$

Another assumption is made by equating the variance of both oscillating signals:

$$\mathrm{Var}[T_{RO0}] = \mathrm{Var}[T_{RO1}]. \tag{7}$$

It can be justified by the fact that both ROs are implemented in the same technology (have the same jitter strength) and the period difference ($\mathrm{E}[\Delta]$) is small.

From the estimated *CSCnt*-distribution, we estimate the probability of a zero/one (p_0/p_1) as:

$$p_i = \sum_{j=0}^{\infty} \Pr(CSCnt = 2j + i), \text{ with } i \in \{0, 1\}. \tag{8}$$

The min-entropy is then calculated as:

$$H_\infty = -\log_2(\max_{i \in \{0,1\}} p_i). \tag{9}$$

The throughput is equal to:

$$T = \frac{1}{\mathrm{E}[CSCnt] \cdot \mathrm{E}[T_{RO1}]}. \tag{10}$$

These equations enable us to estimate the min-entropy-throughput product (HTP). Figures 2 and 3 show the simulation results for variables obtained for a *Spartan 7* and *SmartFusion2* implementation respectively. The upper part depicts the min-entropy, which should be higher than 0.91 according to AIS-31 [7] as indicated by the shaded region. The point with minimal *CSCnt* that meets this requirement is indicated by the vertical line and all values of *CSCnt* or Δ to the right of it give configurations that comply with the AIS-31 standard. This region is indicated by a shade with gradient, because configurations further to the right have reduced throughput and are therefore less desired. To maximise throughput, a configuration as close as possible to the vertical line is wanted. The lower part shows the HTP. It shows a maximum for certain values of $\mathrm{E}[\Delta]$. This maximum is a trade-off between long accumulation time to provide sufficient entropy and keeping the throughput high.

3 ARCHITECTURE

3.1 Entropy source

As shown by the stochastic model in Sect. 2, a small oscillation period difference (tens of ps) is necessary to obtain high enough entropy at the TRNG output. To achieve this small period difference in an FPGA platform, there is a need for precise frequency tuning of clock signals. Various ways are available to achieve this tuning. The most straightforward way is by using a Phase-Locked Loop (PLL). PLLs are present as a primitive circuit in most FPGA devices and are intended for the use in clock generation. A PLL enables to create multiple clock signals with a frequency relation chosen by the designer, based on a reference input clock signal. This input can either originate external to the FPGA fabric e.g. a crystal oscillator, or can be created by another clock resource on the FPGA.

Another approach would be to generate the required clock signals by only utilising the primitive elements that make up the combinatorial logic in the FPGA fabric: LUTs. This removes the dependency of the design on often desired circuit blocks as PLLs and eases off the implementation effort, because LUTs are one of the most abundant resources in most common FPGA families.

Once this RO is implemented, its frequency is fixed at an often unwanted value. In addition, it is also hard to precisely predict this value before programming the FPGA, as due to process variations every device has unique characteristics. Even inside a single device, the location of the RO can alter its frequency by as much as 10 % [13]. This RO frequency variability is used by RO Physical Unclonable Functions (PUFs) to provide a device-unique fingerprint [11].

For obtaining a sufficient entropy density at the output of the TRNG, the designer wants to precisely control the RO frequency. To provide this control, multiple techniques are available to alter the RO frequency after the circuit has been manufactured or programmed in case of an FPGA. These techniques range from dynamically adjusting the load capacitance at the output of every stage, to altering the individual stage drive current strength. Although very effective and commonly used in Application-Specific Integrated Circuit (ASIC) designs, these remain infeasible for use in FPGA systems due to the limited design freedom the fixed fabric provides. The FPGA designer has

to fall back on other techniques that make use of the inherent variations in both wiring as logic delay. To provide the desired RO frequency tuning control, this work will elaborate the following three concepts of delay variability in FPGAs: *Gate delay variability*, *Wiring delay variability*, and *Intra-LUT delay variability*. Each of these three concepts will be introduced in the following three sections.

3.1.1 Gate delay variability. Process variations in nanoscale CMOS technologies make sure that every transistor has unique characteristics. These variations manifest themselves in what is known as device mismatch in the analog design community [19]. Mismatch is the transistor parameter variation between identically designed transistors. It can be either systematic, due to e.g. a poor design layout, or caused by process variations.

These device variations express themselves as variations in the LUT propagation delay. Because a rising edge at the LUT output will be driven by a disjoint set of transistors compared to a falling edge, a LUT programmed to implement a buffer can be characterised by two variables representing the propagation delay of a rising and falling edge.

In this work, we propose to use device variations to our advantage. Fig. 4 shows a proposed *GateVar* configurable RO architecture, that use the LUT propagation delay variability to generate a tuneable oscillating signal. Each column of four multiplexers (MUXs) represents one RO stage. A select signal enables to choose the output of one of the four MUXs that make up the previous stage. In this way, one MUX can be chosen from every column and a custom chain of MUXs can be selected through this network. Due to process variations, every chain will have a unique propagation delay. An additional column containing only NAND gates is added to provide the necessary inversion and enable functionality. Hereby, an RO with $n + 1$ stages is obtained, where n represents the number of MUX columns. Every MUX stage enables for four different configurations (select one out of the four previous stage MUXs), which produces a total of 4^n number of delay configurations for an $n + 1$-stage RO. Both the ROs that make up the entropy source, shown in Fig. 1, are implemented using this architecture to increase matching symmetry resulting in $(4^n)^2$ combinations in total.

3.1.2 Wire delay variability. Process variations do not only translate into variations in transistor properties, but do also affect the interconnection circuitry. On an FPGA, interconnection circuitry includes both the wiring and the switching matrices. The wires interconnecting the logic elements on a chip can be seen as a distributed network of resistors, capacitors, and inductors. Each of these circuit elements will vary in size due to e.g. variations in line edge roughness [9] or the materials used [23].

Figure 5 presents the proposed *WireVar* RO. Each MUX represents one RO stage and selects one out of four wires originating from the output of the previous stage. An additional NAND gate is added to enable/disable the RO. By changing the select input signals of the individual MUXes, a different wire combination is used to propagate the running edge, and a different RO frequency will be obtained. Each RO stages provides four different wires to choose from, which gives 4^n possible configurations for an $n + 1$-stage RO. Again, both ROs making up the entropy source are constructed using this topology, resulting in $(4^n)^2$ combinations in total.

3.1.3 Intra-LUT delay variability. Variations are also present in the internal structure of a LUT. Figure 6 depicts the possible internal structure of a three input LUT. The exact layout of a LUT in the FPGA fabric is often not publicly known. The structure shown, using CMOS transmission gates, here is often assumed [12] and a version of it was patented by *Xilinx* [17]. Adding more inputs to the LUT will increase the depth of the transmission gate tree and additional buffers will have to be inserted to preserve signal integrity. The shaded boxes represent Static Random-Access Memory (SRAM) cells that store the FPGA configuration.

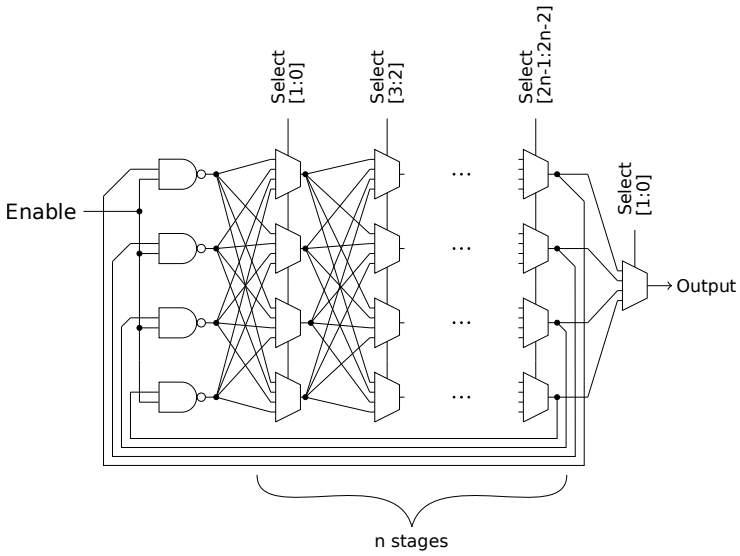

Fig. 4. Architecture of a configurable RO (indicated as *GateVar* in the remainder of this work), using stage delay variability.

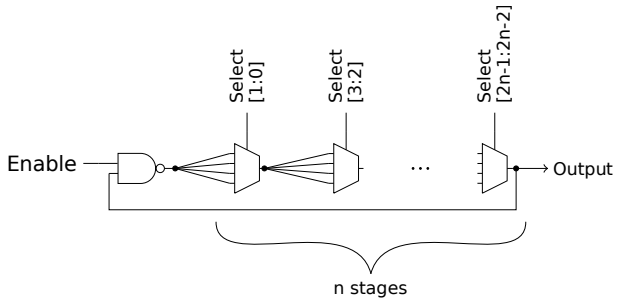

Fig. 5. Architecture of a configurable RO (indicated as *WireVar* in the remainder of this work), using wire delay variability.

The LUT shown in Fig. 6 is configured to act as a buffer to input $in0$. The inputs $in1$ and $in2$ do not have an effect on the logical value of the output, but do have an effect on the internal path selected to propagate the content of an SRAM cell to the output. As proposed by [12], these unused inputs can be used as select signals to fine-tune the LUT propagation delay.

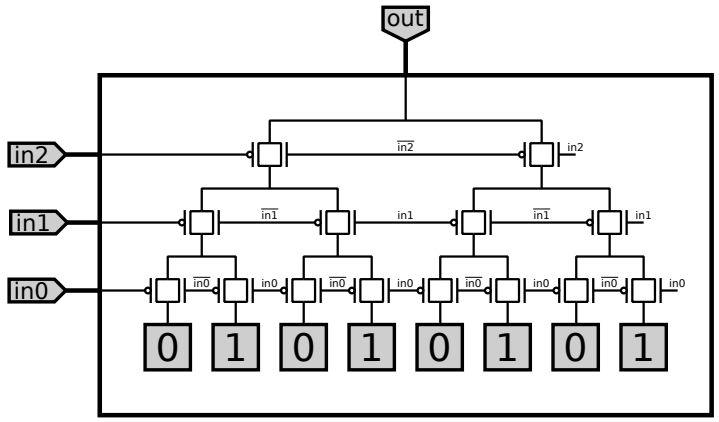

Fig. 6. Possible internal structure of a three input LUT.

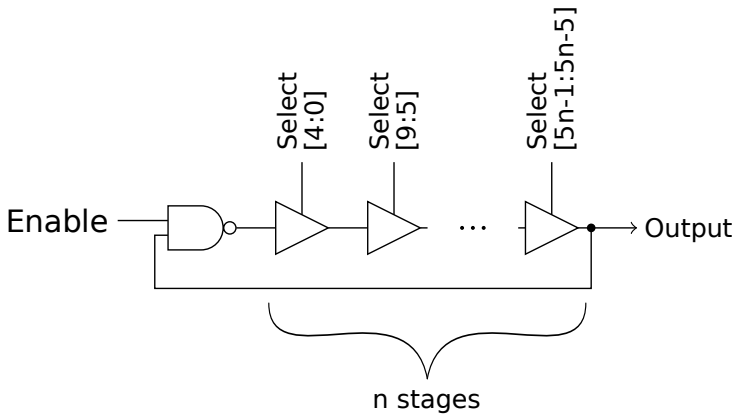

Fig. 7. Architecture of a configurable RO (indicated as *LUTVar* in the remainder of this work), using internal LUT delay variability.

Architecture. The *LUTVar* RO, constructed from LUT buffers, is depicted in Fig. 7. In case of six-input LUTs, Each LUT configured as a buffer has five select inputs and one data input. These five inputs result in 32 configurations per stage and a total of 32^n configurations for an $n+1$-stage RO. A NAND gate is added to provide an inversion and enable input to the RO. An additional degree of freedom is provided by the selection of which physical LUT input port corresponds with the data input. For a six-input LUT, there are six possibilities to assign the data input. The other five physical input ports will be assigned to a select input. When both ROs in the TRNG entropy source are constructed as *LUTVar* ROs, $(32^n)^2$ combinations are obtained in total.

Comparison of LUT physical input ports. To compare the different physical input ports, we introduce two metrics characterising configurable ROs: *range* and *resolution*. *Range* is defined as the InterQuartile Range (IQR) of the obtained oscillation period distribution, when iterating over all select input combinations:

$$range = Q(0.75) - Q(0.25), \tag{11}$$

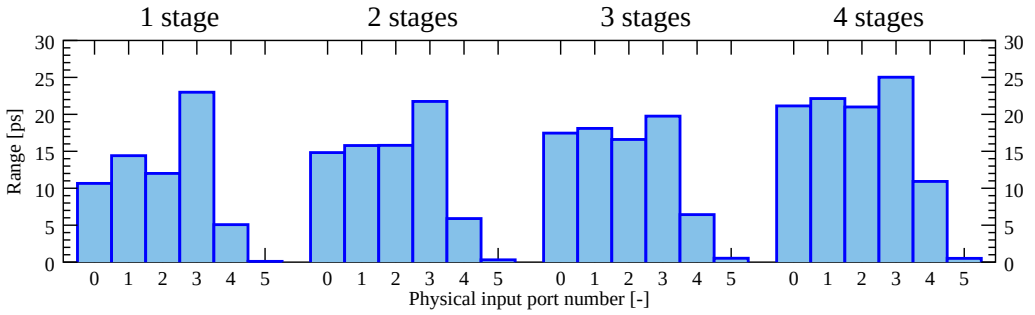

Fig. 8. *Range* comparison of *LUTVar* ROs with data input port ranging from 0 to 5, and number of stages ranging from 1 to 4.

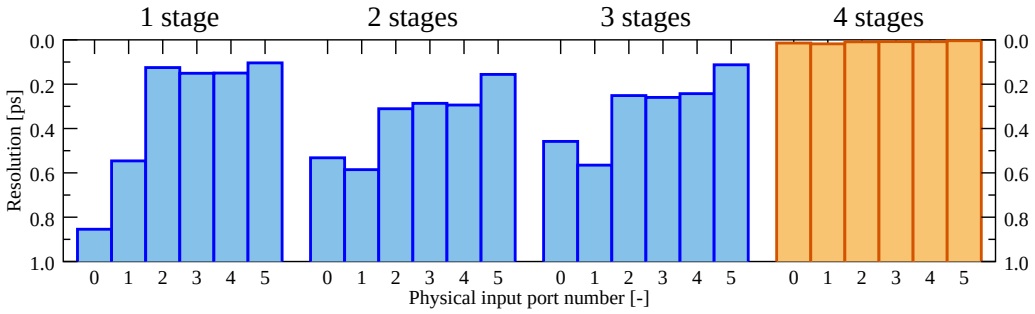

Fig. 9. *Resolution* comparison of *LUTVar* ROs with data input port ranging from 0 to 5, and number of stages ranging from 1 to 4.

with $Q(\cdot)$, representing the inverse cumulative distribution function or quantile function of the obtained RO oscillation period distribution. *Resolution* is obtained by first sorting the obtained oscillation periods and then calculating the median distance between these sorted values. For use as an entropy source in TRNGs, a wide range and a small resolution value are desirable.

Figures 8 and 9 show respectively *range* and *resolution* for *LUTVar* ROs constructed with 1 to 4 stages and signal physical input port numbers ranging from 0 to 5 for a *Spartan 7* implementation (six-input LUTs are used). As can be seen in the figures, *range* increases with increasing number of stages and decreases with increasing signal physical input port number. *Resolution* also improves with increasing number of stages and improves with increasing signal physical input port number. Note that the measured resolution for a *LUTVar* RO with 4 stages was very small (smaller than 0.1 ps) and could not be measured accurately. The values shown in Fig. 9 for 4 stages (indicated with a lighter colour) represent the measurement accuracy and not the actual resolution.

As was shown by these first experiments, signal input ports 0 and 5 mark the two extremes in terms of *range* and *resolution*. To reduce measurement time in the experiments presented in this work, only signal input ports 0 and 5 are tested. The other signal input ports are assumed to have characteristics bounded by these two extremes.

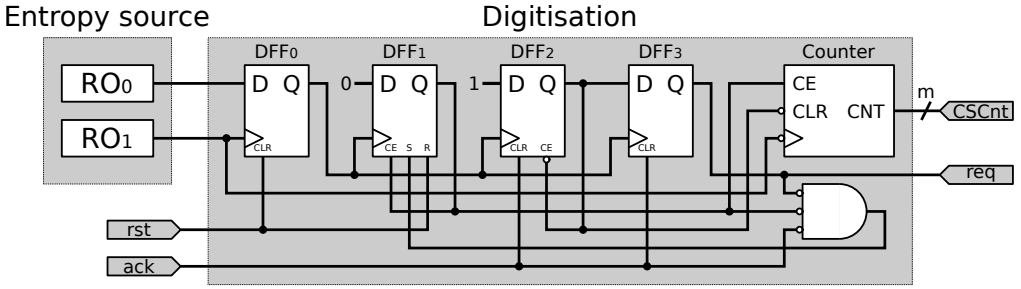

Fig. 10. Architecture of the COSO-TRNG.

3.2 Digitisation

A detailed hardware diagram of the digitisation module is provided in Fig. 10. RO_0 is sampled by a clock signal generated by RO_1 using DFF_0. The output of DFF_0 (S_{beat}) is used to clock DFF_1, DFF_2 and DFF_3. These DFFs regulate the enabling and clearing of the asynchronous counter that counts the period length of S_{beat} relative to the period length of the oscillating signal produced by RO_1. Synchronisation with other logic happens via a two-way handshake, using the signals *ack* and *req*, generated by DFF_3. The asynchronous counter produces an m-bit value (*CSCnt*) used by the controller to configure the ROs and the LSB of this value is used to generate random bits.

3.3 Controller

Simplified pseudocode describing the controller is depicted in Alg. 1. Its function is monitoring if the *CSCnt*-value is still within predefined boundaries (L and H in the algorithm, representing the lower and upper bound respectively) and selecting a new RO configuration (*ROSel*) if necessary. This E[*CSCnt*] is directly related to the matching of the two ROs (E[Δ]) via Eq. (1). The controller sequentially goes through the possible configurations until a suitable one is found. Optimal boundaries have to be chosen to maximise the throughput and provide sufficient entropy from Fig. 2 and 3. A smaller range $[L, H)$ enables finer control, but increases the controller latency to find a suitable configuration.

This controller activates at start-up to find a configuration that validates the stochastic model. After start-up, the controller remains actively checking the produced *CSCnt* values and dynamically recalibrates the ROs when necessary.

4 EXPERIMENTAL RESULTS

The experimental results presented in this section should provide an answer to the following questions:

- Can the proposed configurable RO architectures produce a wide range of frequencies?
- Is searching for an optimal placement inside the FPGA chip still necessary?
- Are placement constraints still necessary?
- Can the proposed configurable RO architectures also work on other FPGAs?
- How many stages are necessary?
- What is the influence of the implementation strategy?

Each of these questions is answered in the following subsections. First the experimental set-up is further clarified.

Algorithm 1 Controller

Input: $CSCnt[m\text{-}1{:}0]$, req
Output: $ROSel[2n\text{-}1{:}0]$, $matched$
Global constant: L, H

```
1:  goodSamples[6:0] ← 0
2:  sampleCnt[6:0] ← 0
3:  ROSel[2n-1:0] ← 0
4:  matched ← 0
5:  while true do
6:    if req then
7:      if L ≤ CSCnt[m-1:0] < H then
8:        goodSamples[6:0] ← goodSamples[6:0] + 1
9:        matched ← 1
10:     end if
11:     if sampleCnt[6:0] == 2^7 − 1 then
12:       if goodSamples[6:0] == 0 then
13:         ROSel[2n-1:0] ← ROSel[2n-1:0] + 1
14:         matched ← 0
15:       end if
16:       goodSamples[6:0] ← 0
17:     end if
18:     sampleCnt[6:0] ← sampleCnt[6:0] + 1
19:   end if
20: end while
```

Measured $CSCnt$ and RO period distributions will be visualised using box plots. A box plot visualises the data distribution, where 50 % of the data is contained within the box. The distribution's median is shown as a horizontal line inside the box. A data point is considered as an outlier (marked with a cross in the plot) if it is situated further than 1.5 times the interquartile range from the closest quartile. All the box plot figures depicting a $CSCnt$ distribution will be on a shade with gradient representing the configurations with high enough entropy per bit.

4.1 Experimental set-up

We first implemented the proposed entropy source and digitisation as depicted in Fig. 10 on a *Xilinx Spartan 7* FPGA. The number of configurable stages (n) in any experiment except in Sect. 4.3, is set to four, which gives a total of $4^4 = 256$ combinations for each *GateVar* and *WireVar* RO and $32^4 = 1\,048\,576$ combinations for each *LUTVar* RO. To reduce measurement time, we did not test every *LUTVar* combination, but used a Linear-Feedback Shift Register (LFSR) to generate $2^{15} = 32\,768$ configuration inputs. Every four-input MUX fits inside a six-input LUT, the NAND gates and configurable *LUTVar* buffers are also implemented by one six-input LUT each. As for realistic values of $\mathrm{Var}[T_{RO}]$ and $\mathrm{E}[\Delta]$, $\mathrm{E}[CSCnt]$ is below 256, an 8-bit asynchronous counter can be used in the digitisation, but initially we used a 16-bit counter in the experiments. This counter is reset every period of S_{beat} and read by the controller. The LSB of this counter is used as the generated random bit.

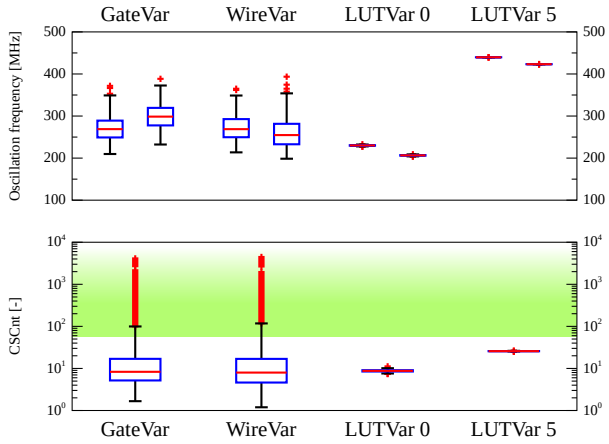

Fig. 11. Measured RO periods and obtainable *CSCnt* values for a fixed placement on *Spartan 7*.

4.2 Feasibility of the architecture

We first implemented the entropy source and digitisation, with manual placement on a *Xilinx Spartan 7* FPGA. The LUTs containing the RO stages are placed in a symmetrical way to facilitate matching the two ROs. The upper part of Fig. 11 shows a box plot of the measured frequencies for both ROs, sweeping all the configurations. This was repeated for four different configurable RO architectures: *GateVar*, *WireVar*, *LUTVar 0* and *LUTVar 5*. In the lower part of the figure, a box plot is given for the obtained average *CSCnt* realisations. For the *GateVar* and *WireVar* architectures, a wide range up to 10^4 is achievable (Δ ranging down to 0.4 ps), which enables the controller to find a configuration close to the optimal HTP. On the other hand, the *LUTVar* ROs do not provide a wide enough range of oscillation frequencies to enable high enough *CSCnt* values. This is due to the limited range of oscillation frequencies a single *LUTVar* RO can produce and makes them unable to overcome the inherent mismatch present between the two ROs.

4.3 Global placement

To test if the RO architectures are capable of matching the ROs independent from Global Placement (GP) constraints, the previous experiment is carried out on 25 different locations, manually chosen to cover the entire chip. The Local Placement (LP) constrains are maintained to improve matching. Figures 12, 13, 14 and 15 show a box plot for each of the tested locations for the *GateVar*, *WireVar*, *LUTVar 0* and *LUTVar 5* architecture respectively. The *CSCnt* values in this experiment are calculated based on the measured RO frequencies, instead of testing every possible RO pair, to reduce measurement time. ROs with three configurable stages were analysed.

This experiment shows that indeed sufficient matching can be obtained at every tested location for the *GateVar* and *WireVar* architectures, which is in strong contrast with previous designs that use coherent sampling with ROs [16, 21, 27], where a large design effort was needed to manually find an FPGA location with sufficient matching between the ROs.

The results from the *LUTVar* architectures are similar as was reported in these previous COSO designs: some locations provide well-matched ROs and high *CSCnt* values. But these locations remain sparsely scattered over the FPGA fabric (7/25 locations for *LUTVar 0* and 12/25 locations for

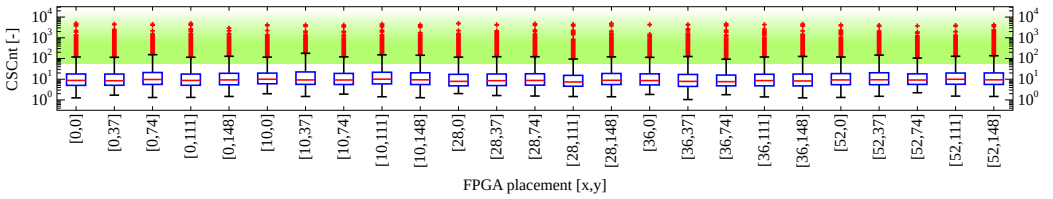

Fig. 12. Calculated *CSCnt* values for *GateVar* ROs at 25 different locations on *Spartan 7*.

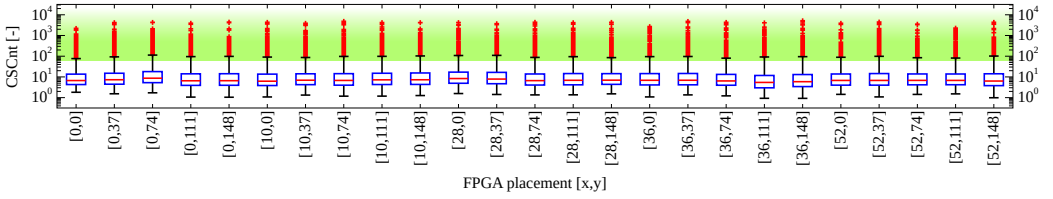

Fig. 13. Calculated *CSCnt* values for *WireVar* ROs at 25 different locations on *Spartan 7*.

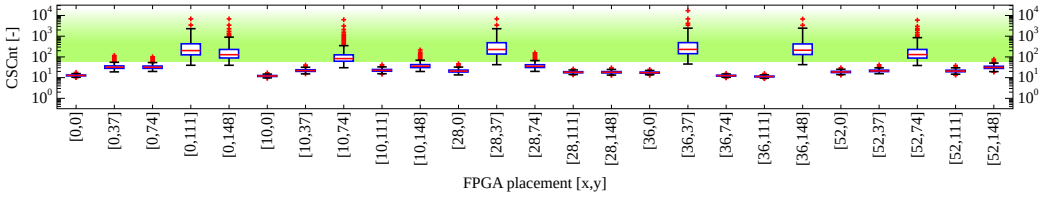

Fig. 14. Calculated *CSCnt* values for *LUTVar* ROs with signal input 0 at 25 different locations on *Spartan 7*.

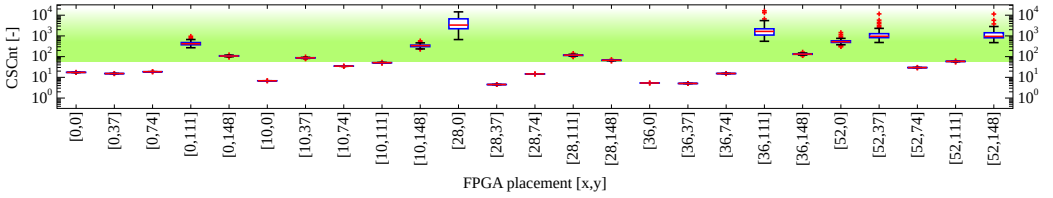

Fig. 15. Calculated *CSCnt* values for *LUTVar* ROs with signal input 5 at 25 different locations on *Spartan 7*.

LUTVar 5). Note that *LUTVar 0* provides a larger *CSCnt* range than *LUTVar 5*, as can be explained by the larger *range* measured in Fig. 8.

4.4 Local placement

The next experiment shows that even the LP constraints are not required for the correct operation of the TRNG. Figure 16 depicts the RO frequencies and corresponding measured average *CSCnt* values for an implementation on the *Xilinx Spartan 7* FPGA, without any specified placement constraints. The figure shows that matching can still be obtained in case of the *GateVar* and *WireVar* RO architectures. In this particular case, both *LUTVar* architectures are also able to obtain high

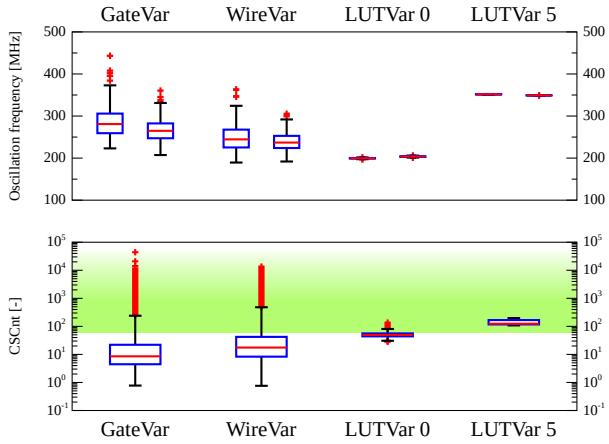

Fig. 16. Measured RO periods and obtainable *CSCnt* values for omitted LP constraints on *Spartan 7*.

enough *CSCnt* values. However, we will show in Sect. 4.7 that these results are highly dependent on the FPGA implementation strategy and therefore very difficult to control by the designer.

4.5 Portability

To prove the portability of the proposed TRNG entropy source architectures, the previous experiment is repeated on a *Microsemi SmartFusion2* FPGA. Figure 17 shows the results, which are similar to those obtained on the *Spartan 7* FPGA. The *SmartFusion2* FPGA only features four-input LUTs, while the *Spartan 7* FPGA has six-input LUTs. The smaller LUTs mean that after porting, one four-input MUX does not fit into one LUT anymore. The design is slightly adapted by redefining the used primitives (only DFFs and LUTs) to account for the change in library naming at different FPGA vendors. The synthesis tool automatically constructs the four-input MUXs using multiple four-input LUTs. The *LUTVar* ROs also use four-input LUTs and signal physical input ports 0 and 3 where tested. The results show that the portability claim is valid for the *GateVar* and *WireVar* architectures.

4.6 Optimal number of stages

In this experiment, we varied the number of stages and monitored the achievable *CSCnt* values. More stages offer a greater number of configurations and increase the chance of finding sufficient matching. Figure 18 shows the results of this experiment on a *Spartan 7* with omitted placement constraints. The *GateVar* and *WireVar* architectures with one or two stages show none or only a few *CSCnt* values that fall into the shaded region. These with three or more stages show a large amount of *CSCnt* values in this region and are therefore preferred. For the *LUTVar* architectures, only ROs with four stages provide high enough *CSCnt* values.

4.7 Implementation strategy

The previous experiment is repeated with the *Area Explore* implementation strategy instead of the default strategy in the *Xilinx Vivado* design tool. This strategy will run multiple optimisation runs to reduce the circuit area [25]. The HDL implementation of the TRNG circuit remained unchanged. Figure 19 shows the results. Both *GateVar* and *WireVar* architectures with four stages still provide

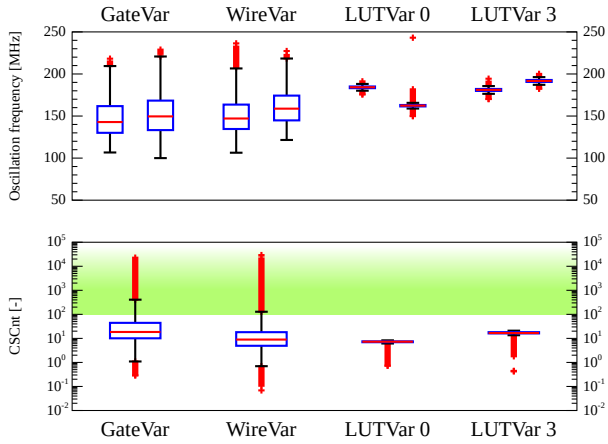

Fig. 17. Measured RO periods and obtainable *CSCnt* values for omitted LP constraints on *SmartFusion2*.

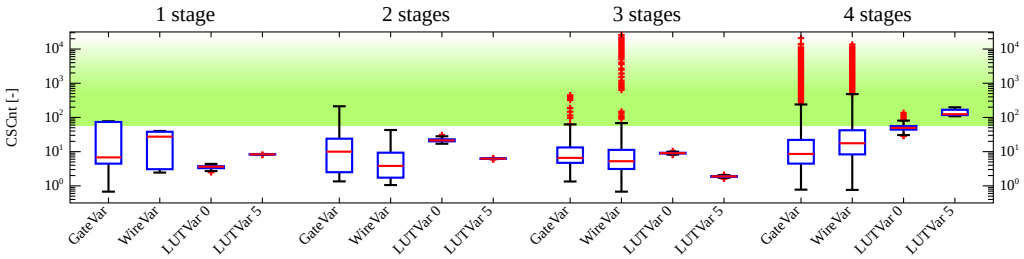

Fig. 18. Measured *CSCnt* values for different number of RO stages and omitted placement constraints on *Spartan 7*.

CSCnt values in the shaded region. *WireVar* with three stages only has few *CSCnt* values high enough. *LUTVar 5* is unable to obtain sufficient RO matching regardless the number of stages we tested. Based on these results, we recommend using *GateVar* or *WireVar* with four stages as this will give high certainty of obtaining well matched ROs, while keeping the RO footprint small.

5 RESULTS AND COMPARISON

To get a theoretical estimate of the entropy per bit, the magnitude of the jitter strength ($\mathrm{Var}[T_{RO0}]/\mathrm{E}[T_{RO0}]$) is needed. For the *SmartFusion2* implementation, we use the values obtained in [16], as similar FPGA devices are used there. To get a conservative entropy estimate, we take the minimal jitter strength over all frequencies tested in [16]. The obtained jitter strength value is 1.6 fs, which gives a period jitter equal to 3.2 ps for an RO oscillation frequency equal to 160 MHz. For the *Spartan 7* implementation, we used the variance of the measured *CSCnt* values to estimate the jitter strength via Eqs. 2 and 4. The obtained jitter strength equals 4.6 fs. At these jitter strengths, the acceptable *CSCnt* range is given by the shaded area in the lower part of Figs. 2 and 3. We assume the jitter strength to be independent of RO architecture, so the calculated *CSCnt* bounds in these figures are valid regardless the architecture of the ROs.

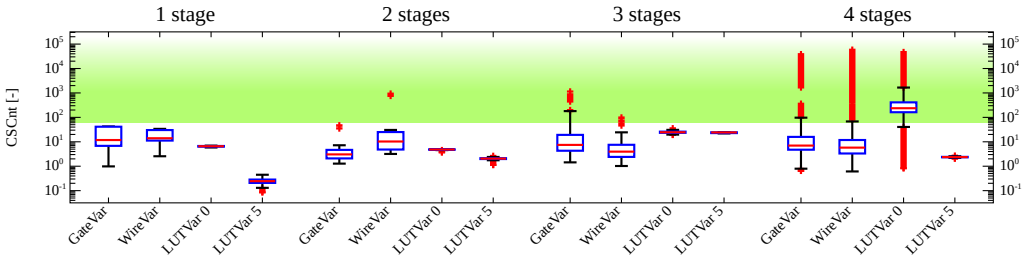

Fig. 19. Measured *CSCnt* values for different number of RO stages and omitted placement constraints on *Spartan 7*, using the *Area Explore* implementation strategy.

Random bits are generated using the *GateVar* and *WireVar* RO architectures in a *Spartan 7* implementation. To demonstrate portability, the *GateVar* architecture is also used to generate random bits on a *SmartFusion2* implementation. We did not generate random bits using the *LUTVar* architecture, as the experiments in Sect. 4 showed that this RO architecture is unable to reliably match the two ROs. All random bits were generated using omitted GP and LP constraints and using ROs with four configurable stages for the *Spartan 7* implementation and three configurable stages for the *SmartFusion2* implementation.

Both the *Spartan 7* (*GateVar* and *WireVar*) and *SmartFusion2* (*GateVar*) implementations obtain a configuration with an average *CSCnt* value in the shaded region, close to the maximal HTP: 60.6, 77.8 and 107.85 respectively, producing a throughput of 4.65 Mbit/s, 2.34 Mbit/s and 1.47 Mbit/s. The Shannon entropy should be larger than 0.997 per bit as required by [7]. Converted to min-entropy, it should be higher than 0.91 per bit. Calculated from the model at a measured oscillation frequency of 271 MHz and 160 MHz, the obtained min-entropy is 0.92 per bit, 0.99 per bit and 0.95 per bit for the *Spartan 7* (*GateVar* and *WireVar*) and *SmartFusion2* (*GateVar*) respectively. The bit streams pass test procedure B (T6 - T8), described in AIS-31 [7] using 120 Mbit of generated data and have an estimated Shannon entropy of 0.998 per bit, 0.999 per bit and 0.997 per bit, using the results from test T8. Both implementations are therefore PTG.3 compliant if cryptographic post processing is added. We utilise CBC-MAC post processing as specified in the NIST SP 800-90B standard [20]. After post processing, the throughput is lowered by a factor of two and all three pass the NIST SP 800-22 tests.

A comparison to previous COSO-TRNGs and other designs that comply with the AIS-31 standard is given in Table 2. This work includes the results reported in [15], where the *GateVar* RO was implemented on a *Xilinx Spartan 6* FPGA. Compared to other designs, this work offers a throughput higher than 1 Mbit/s and a larger area, but still a lot smaller than the design proposed in [27]. However, this design comes with a built-in on-line test which alarms the user in case that no sufficient matching can be found. According to both standards [7, 20], an on-line testing module is a necessary requirement for certification and therefore increases the area usage of the other designs. The design effort is also lowered in this design. Previous work required at least Manual Placement (MP) and often also Manual Routing (MR). This effort had to be repeated every time when the design is implemented on an other device, even from the same FPGA family.

If higher throughput is required, the techniques proposed in [21] can still be utilised with our proposed entropy source. These techniques boost the throughput by a factor of four. However the stochastic model has to be extended to provide a valid entropy estimation when using mutual sampling.

Table 2. Comparison with other work

Architecture	FPGA family	Area [DFFs/LUTs]	Throughput [Mbit/s]	Design effort
This work	***Spartan 6***[a,b]	**39/108**[c]	**3.30**	-
	SmartFusion2[a]	**38/111**[c]	**1.47**	-
	Spartan 7[a]	**62/82**[d]	**4.65**	-
	Spartan 7[e]	**62/58**[d]	**2.34**	-
Original COSO [16]	*Spartan 6*	3/18	0.54	MP
	Cyclone V	3/13	1.44	MP
	SmartFusion2	3/23	0.328	MP
COSO: one bit per half cycle [21]	*Actel Fusion AFS600*	7/24[f]	2	MP & MR
	Spartan 3	7/18[f]	1.6	MP & MR
COSO: mutual sampling [21]	*Actel Fusion AFS600*	14/29[f]	4	MP & MR
	Spartan 3	14/23[f]	3.2	MP & MR
COSO: parameter adjustment [27]	*Virtex-5*	109 slices	4.08	MP & MR
DC-TRNG [1]	*Spartan 6*	128 slices	1.1	MP
	Cyclone V	273 ALMs	1.116	MP & MR
PLL-TRNG [1]	*Spartan 6*	190 slices[g]	1.0416	PLL required
	Cyclone V	273 ALMs	1.04	PLL required
ES-TRNG [26]	*Spartan 6*	5/10	1.15	MP
	Cyclone V	6/10	1.067	MP
TERO [16]	*Spartan 6*	12/39	0.625	MP & MR
	Cyclone V	12/46	1	MP & MR
	SmartFusion2	12/46	1	MP & MR
STR [16]	*Spartan 6*	256/346	154	MP & MR
	Cyclone V	256/352	245	MP & MR
	SmartFusion2	256/350	188	MP & MR

[a] Using the *GateVar* RO architecture.
[b] Results reported in [15].
[c] Values calculated for three-stage ROs, controller hardware included.
[d] Values calculated for four-stage ROs, controller hardware included.
[e] Using the *WireVar* RO architecture.
[f] Values calculated using shown circuit diagram.
[g] Including embedded tests and data interface.

6 CONCLUSION AND FUTURE WORK

In this work, we proposed and analysed three new entropy sources for the COSO-TRNG that significantly reduce the design effort. We experimentally verified the feasibility of *GateVar* and *WireVar* architectures and showed that it is always possible to meet the entropy requirements from the stochastic model even when placement constraints are omitted or the design is ported to another FPGA family. This property renders the TRNG highly suitable for integration into larger cryptosystems. Random bits are generated with a maximal throughput of 4.65 Mbit/s and 1.47 Mbit/s on a *Spartan 7* and a *SmartFusion2* FPGA respectively and are able to pass the statistical

tests. The design only has a modest area requirement, but comes with a cost of increased latency when the controller is searching for an optimal configuration.

Future work will focus on investigating whether this methodology is also applicable to other TRNG designs that require a large design effort, optimising the searching strategy of the controller to improve the induced latency, and observing the behaviour of this entropy source under active manipulation attacks.

ACKNOWLEDGMENTS

This work was supported in part by the Research Council KU Leuven: C16/15/058. In addition, this work is supported in part by the Hercules Foundation AKUL/11/19, and by the European Commission through the Horizon 2020 research and innovation programme Cathedral ERC Advanced Grant 695305. Adriaan Peetermans is funded by an FWO fellowship and Vladimir Rožić is an FWO postdoctoral researcher.

REFERENCES

[1] Josep Balasch, Florent Bernard, Viktor Fischer, Miloš Grujić, Marek Laban, Oto Petura, Vladimir Rožić, Gerard van Battum, Ingrid Verbauwhede, Marnix Wakker, and Bohan Yang. 2018. Design and testing methodologies for true random number generators towards industry certification. In *2018 IEEE 23rd European Test Symposium (ETS)*, Vol. 2018. IEEE, Leuven, Belgium, 1–10.

[2] Mathieu Baudet, David Lubicz, Julien Micolod, and André Tassiaux. 2011. On the security of oscillator-based random number generators. *Journal of cryptology* 24, 2 (2011), 398–425.

[3] Florent Bernard, Viktor Fischer, and Boyan Valtchanov. 2010. Mathematical model of physical RNGs based on coherent sampling. *Tatra Mountains Mathematical Publications* 45, 1 (2010), 1–14.

[4] Abdelkarim Cherkaoui, Viktor Fischer, Laurent Fesquet, and Alain Aubert. 2013. A very high speed true random number generator with entropy assessment. In *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*. Springer, Grenoble, France, 179–196.

[5] Jean Luc Danger, Sylvain Guilley, and Philippe Hoogvorst. 2009. High speed true random number generator based on open loop structures in FPGAs. *Microelectronics journal* 40, 11 (2009), 1650–1656.

[6] Lawrence E Bassham III, Andrew L Rukhin, Juan Soto, James R Nechvatal, Miles E Smid, Elaine B Barker, Stefan D Leigh, Mark Levenson, Mark Vangel, David L Banks, et al. 2010. *A statistical test suite for random and pseudorandom number generators for cryptographic applications.* Technical Report. NIST, Gaithersburg, MD.

[7] Wolfgang Killmann and Werner Schindler. 2011. A proposal for: Functionality classes for random number generators. https://cosec.bit.uni-bonn.de/fileadmin/user_upload/teaching/15ss/15ss-taoc/01_AIS31_Functionality_classes_for_random_number_generators.pdf. [Online; accessed 22-May-2019].

[8] Paul Kohlbrenner and Kris Gaj. 2004. An embedded true random number generator for FPGAs. In *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field-Programmable Gate Arrays*. ACM, ACM, Fairfax, VA, 71–78.

[9] L. H. A. Leunissen, W. Zhang, W. Wu, and S. H. Brongersma. 2006. Impact of line edge roughness on copper interconnects. *Journal of Vacuum Science & Technology B: Microelectronics and Nanometer Structures Processing, Measurement, and Phenomena* 24, 4 (2006), 1859–1862.

[10] Yuan Ma, Jingqiang Lin, Tianyu Chen, Changwei Xu, Zongbin Liu, and Jiwu Jing. 2014. Entropy evaluation for oscillator-based true random number generators. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, Springer, Beijing, China, 544–561.

[11] Abhranil Maiti and Patrick Schaumont. 2009. Improving the quality of a physical unclonable function using configurable ring oscillators. In *2009 International Conference on Field Programmable Logic and Applications*. IEEE, Blacksburg, VA, 703–707.

[12] Mehrdad Majzoobi, Farinaz Koushanfar, and Srinivas Devadas. 2011. FPGA-based true random number generation using circuit metastability with adaptive feedback control. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, Springer, Houston, TX, 17–32.

[13] Anindo Mukherjee and Kevin Skadron. 2006. *Measuring parameter variation on an FPGA using ring oscillators.* Univ. of Virginia Dept. of Computer Science Tech Report CS-2006-16. University of Virginia, Dept. of Computer Science, Charlottesville, VA.

[14] NIST. 2001. *Security requirements for cryptographic modules.* Technical Report. NIST, Washington, DC.

[15] Adriaan Peetermans, Vladimir Rožić, and Ingrid Verbauwhede. 2019. A highly-portable true random number generator based on coherent sampling. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL).*

IEEE, Leuven, Belgium, 218–224.

[16] Oto Petura, Ugo Mureddu, Nathalie Bochard, Viktor Fischer, and Lilian Bossuet. 2016. A survey of AIS-20/31 compliant TRNG cores suitable for FPGA devices. In *2016 26th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, Saint-Etienne, France, 1–10.

[17] Tao Pi and Patrick J Crotty. 2003. FPGA lookup table with transmission gate structure for reliable low-voltage operation. US Patent 6,667,635.

[18] Vladimir Rožić, Bohan Yang, Wim Dehaene, and Ingrid Verbauwhede. 2015. Highly efficient entropy extraction for true random number generators on FPGAs. In *Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015*. IEEE, Leuven, Belgium, 116:1–116:6.

[19] Willy Sansen. 2007. *Analog design essentials*. Vol. 859. Springer Science & Business Media, Leuven, Belgium.

[20] Meltem Sönmez Turan, Elaine Barker, John Kelsey, Kerry A McKay, Mary L Baish, and Mike Boyle. 2018. *Recommendation for the entropy sources used for random bit generation.* Technical Report. NIST, Gaithersburg, MD.

[21] Boyan Valtchanov, Viktor Fischer, and Alain Aubert. 2009. Enhanced TRNG based on the coherent sampling. In *2009 3rd International Conference on Signals, Circuits and Systems (SCS)*. IEEE, Saint-Etienne, France, 1–6.

[22] Michal Varchola and Miloš Drutarovskỳ. 2010. New high entropy element for FPGA based true random number generators. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, Kosice, Slovak Republic, 351–365.

[23] Janet Wang, Praveen Ghanta, and Sarma Vrudhula. 2004. Stochastic analysis of interconnect performance in the presence of process variations. In *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.* IEEE, Tucson, AZ, 880–886.

[24] Piotr Zbigniew Wieczorek and Krzysztof Golofit. 2014. Dual-metastability time-competitive true random number generator. *IEEE Transactions on Circuits and Systems I: Regular Papers* 61, 1 (2014), 134–145.

[25] Xilinx. 2018. Vivado Design Suite User Guide: Implementation (UG904). Retrieved July 2, 2020 from https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug904-vivado-implementation.pdf

[26] Bohan Yang, Vladimir Rožić, Miloš Grujić, Nele Mentens, and Ingrid Verbauwhede. 2018. ES-TRNG: A high-throughput, low-area true random number generator based on edge sampling. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018, 3 (2018), 267–292.

[27] Jing Yang, Yuan Ma, Tianyu Chen, Jingqiang Lin, and Jiwu Jing. 2016. Extracting more entropy for TRNGs based on coherent sampling. In *International Conference on Security and Privacy in Communication Systems*. Springer, Beijing, China, 694–709.