# Towards Inter-Vendor Compatibility of True Random Number Generators for FPGAs

Miloš Grujić, Bohan Yang, Vladimir Rožić and Ingrid Verbauwhede
imec-COSIC, KU Leuven, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
Email:{milos.grujic, bohan.yang, vladimir.rozic, ingrid.verbauwhede}@esat.kuleuven.be

***Abstract*—True random number generators (TRNGs) are fundamental constituents of secure embedded cryptographic systems. In this paper, we introduce a general methodology for porting TRNG across different FPGA vendor families. In order to demonstrate our methodology, we applied it to the delay-chain based TRNG (DC-TRNG) on Intel Cyclone IV and Cyclone V FPGAs. We examine vendor-agnostic generality of the underlying DC-TRNG principle and propose modifications to address differences in structure of FPGAs. Implementation of the DC-TRNG on Cyclone IV uses 149 LEs (<0.1% of available resources) and has a throughput of 5Mbps, while on Cyclone V it occupies 230 ALMs (<1.5% of resources) with an output rate of 12.5 Mbps. The quality of the random bits produced by the DC-TRNG on Intel Cyclone IV and V is further confirmed by using NIST statistical test suite.**

## I. Introduction

True random number generators (TRNGs) are essential cryptographic components which generate unpredictable random bits that can be used as challenges in cryptographic protocols, secret keys, padding values and masks. TRNGs rely on stochastic physical phenomena in order to generate truly unpredictable outputs. The ever-increasing ubiquity of FPGAs in modern cryptographic system has created the demand for having FPGA compatible TRNGs. TRNGs should have a high throughput, small logic consumption and an appropriate accompanying stochastic model, which is needed as an explicit proof of security [1], [2], [3]. Additionally, random bits should pass statistical tests – NIST SP 800-22 [2] and AIS-31 [1].

Harvesting randomness from FPGAs is a challenge since FPGAs are primarily developed to behave in a deterministic digital manner. Most commonly used sources of randomness in digital systems are the variable component of timing delays – jitter and the metastable behavior of flip-flops. Considering that FPGA vendors aim to decrease variations of these nondeterministic physical processes, they need to be carefully characterized and studied before exploiting them as sources of randomness in a TRNG design. Moreover, physical variables that describe these processes have significantly distinctive values for FPGAs of different vendors due to different layouts and process technologies. This often makes porting TRNG designs from their native FPGA platform to the FPGA of a different vendor a challenging task, which requires an approach different from the one used when porting conventional digital designs. If not done correctly and without understanding of the randomness generating processes, it can have disastrous consequences on the security of the TRNG. The design of a TRNG for FPGAs is further hindered by the fact that FPGA design tools are not intended to deal with most primitives used in TRNG designs because of their asynchronous nature.

In this paper we propose a methodology for migrating a TRNG design to an FPGA platform of another vendor, while maintaining the level of security. We then verify correctness of the proposed methodology by applying it on the case of the DC-TRNG, which is originally developed for Xilinx Spartan FPGAs, by porting it to Intel Cyclone FPGAs. Moreover, we propose a novel way of designing the priority encoder component of the DC-TRNG by using the information about the underlying FPGA carry-chain architecture obtained by experiments and the stochastic model of the DC-TRNG. In this way, we reduce the inherent bias introduced by differences in propagation delays of carry-chain stages.

## II. Methodology

As a solution for inter-vendor portability of TRNGs for FPGAs, we propose a methodology with following steps:

- Resource identification
- Measurement of physical parameters
- Initial implementation parameters selection
- Architecture modifications
- TRNG implementation

Next, we describe and explain each step of our methodology.

*1) Resource identification:* Firstly, it is necessary to establish the availability of FPGA blocks and routing resources which are needed for the TRNG implementation on the target FPGA. Some TRNG designs require usage of specific FPGA resources (e.g., PLLs), very fast lookup tables (LUTs) or balanced routing paths [4], [5]. These are not always available on FPGAs of all vendors, either because they do not exist or they are already used by other digital blocks in the system. In those cases, it is necessary to identify possible ways to replace unavailable resources, but without deteriorating the security level of the TRNG.

*2) Measurement of physical parameters:* After establishing which resources will be used on the target FPGA, it is essential to measure values of the physical parameters that are specific for the target FPGA. These parameters are of the utmost importance for the security of the TRNG, since they characterize the randomness generating process and are thus required for a correct entropy estimation of the TRNG on the target FPGA. The values of the physical parameters will differ significantly from one FPGA to another, due to process,

978-3-9819263-0-9/DATE18/©2018 EDAA

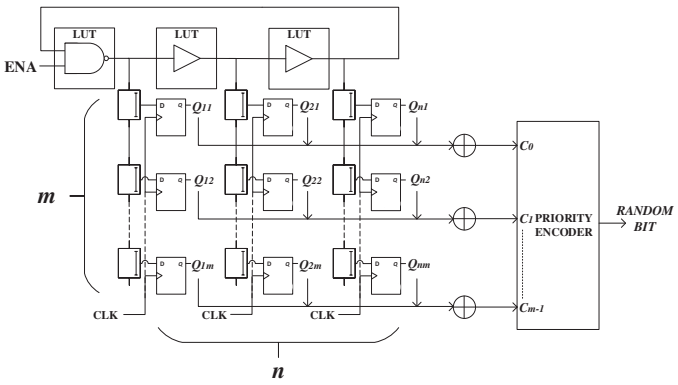


Fig. 1. Architecture of the DC-TRNG.

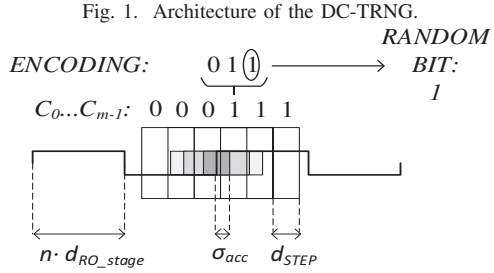


Fig. 2. Entropy extraction process.

voltage and temperature variations. Measuring of physical parameters should be performed preferably on the target FPGA itself, in order to obtain reliable values, without influence of the noise introduced by the measuring equipment.

*3) Initial implementation parameters selection:* Only after measuring the values of physical parameters on the target FPGA, the designer can select implementation parameters for it. These parameters can be the number of metastable elements, the number of ring-oscillators, length of the delay lines, the number of stages of the ring-oscillator etc. The choice of implementation parameters is not only important from security point of view, but also in terms of area, throughput and power.

*4) Architecture modifications:* Modifications of the original TRNG design are necessary if there are substantial differences in structure of native and target FPGA, if physical parameters of target FPGA have values that hinder straightforward implementation or if the throughput can be improved. However, the modifications should be performed carefully, so that the claimed level of TRNG security is maintained. After this step, it might be needed to tweak implementation parameters.

*5) TRNG implementation:* Because of their asynchronous nature, most TRNG designs require designer's manual intervention at some point of the design flow. This includes instantiations of low-level FPGA primitives, manual placement and routing. While it is usually feasible to automate placement of TRNG components, some designs will still require manual routing, which cannot be easily automated.

## III. Case study: DC-TRNG

### A. Behavioral Model and Operation Principle

DC-TRNG, originally proposed in [6], is a TRNG that uses fast carry-logic (*CARRY4* primitives) in Xilinx FPGAs for time-to-digital conversion in order to precisely sample the timing jitter of the ring-oscillators thereby enhancing the amount of extracted entropy. The general architecture of the DC-TRNG is given in Fig. 1. The entropy source is an $n$-stage ring-oscillator, implemented using LUTs. The entropy extractor consists of delay-chains constructed by cascading CARRY4 primitives, the flip-flops and the priority encoder.

First, the oscillations in the ring-oscillator are enabled by asserting signal ENA, and timing jitter starts to accumulate over time. At the output of each delay stage of the ring-oscillator there is a delay-chain, along which the ring-oscillator signals propagate. After time $t_A$, when sufficient jitter is accumulated $\sigma_{acc}(t_A)$, the rising edge of the signal CLK is asserted and states of the delay-chains are recorded in corresponding flip-flops. Values of all flip-flops at the same position in all parallel delay-chains are xored and the resulting $m$-bit vector is connected to the input of the priority encoder block. This block encodes the edge position. The LSB of the edge position is the raw random bit at the TRNG output.

Platform-specific physical parameters that are of concern for the DC-TRNG implementation are: delay of one stage of a ring-oscillator – $d_{RO\_stage}$, standard deviation of LUT delay – $\sigma_{LUT}$ and delay of one stage of delay-chain – $d_{step}$.

The design parameters of the DC-TRNG are: the number of stages of the ring-oscillator – $n$, the number of stages in each delay-chain – $m$ and the jitter accumulation time – $t_A$. Several restrictions exist for choosing the values of these parameters. The value of parameter $m$ has to be chosen such that the total delay of the delay-chain $m \cdot d_{step}$ is greater than $d_{RO\_stage}$. This ensures that the jittery signal edge is always detected in at least one delay-chain. Higher accumulation time $t_A$ results in higher $\sigma_{acc}$, which in turn increases the amount of entropy per bit. However, higher $t_A$ also means a lower throughput. The number of stages of the ring-oscillator – $n$ is the only implementation parameter that has no effect on the amount of entropy per bit or correct operation of the DC-TRNG, and thus can be chosen arbitrarily in order to achieve the most compact implementation. Fig. 2 illustrates the process of extracting a random bit from the position of the captured jittery edge. Raw bits of the DC-TRNG are not statistically perfect, and are enhanced by parity filter post-processing, to reduce biases and pass NIST statistical tests [6], [7].

### B. DC-TRNG on Intel Cyclone IV and V FPGA

*Step 1:* At this step, we identify which FPGA logic units the DC-TRNG requires. Basic units in Cyclone IV - LEs (logic elements) and in Cyclone V - ALMs (adaptive logic modules) contain LUTs, flip-flops and carry-chains, needed for the DC-TRNG porting. Carry-chain in Cyclone IV is implemented as a small three-input LUT [8], while a carry-chain in Cyclone V is implemented with only a few dedicated gates [9].

*Step 2:* The first physical parameter that we measure is the jitter contribution of a single LUT. We implement an on-chip jitter-measurement circuit, as recommended in [10]. The setup consists of two ring-oscillators and two long tapped delay-chains connected to their outputs. Accumulated jitter

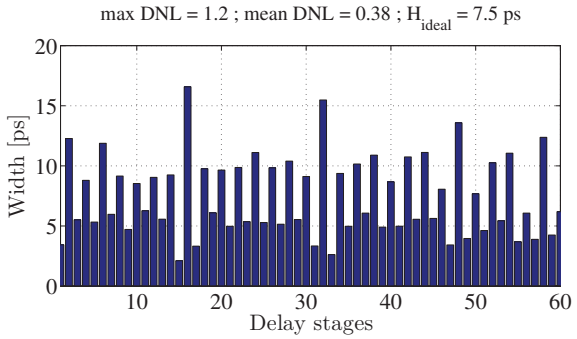


Fig. 3. Cyclone V - width of delay stages.

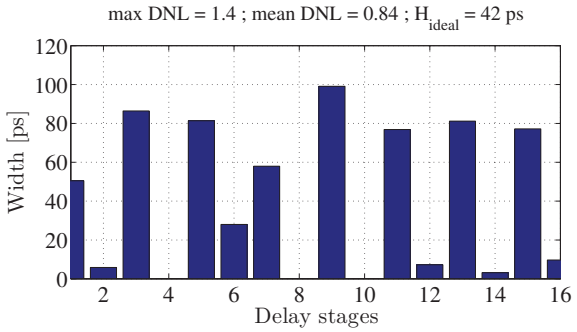


Fig. 4. Cyclone IV - width of delay stages.

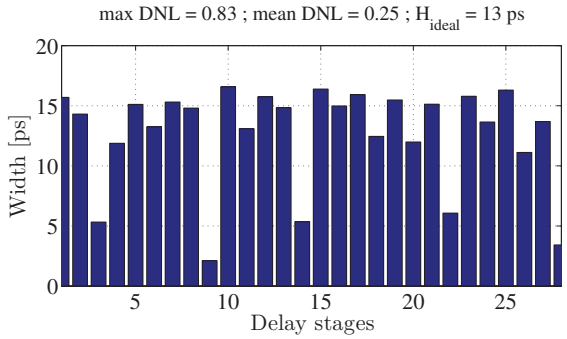


Fig. 5. Cyclone V - width of *virtual* delay stages.

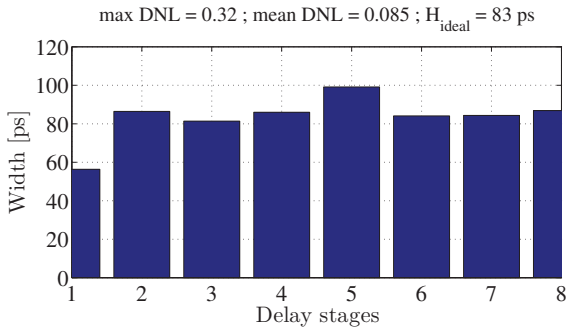


Fig. 6. Cyclone IV - width of *virtual* delay stages.

can be determined by comparing the difference between edge positions in the captured outputs of the delay-chains. Because of the differential measurement setup, the effect of the global noise on jitter measurement is minimized. Secondly, we determine the delay of a single stage of the ring-oscillator. We can do that by counting number of oscillations between two rising edges of the system clock, using the ripple counter. Finally, we determine the average delay of one stage - bin of the delay-chain. Bins of the delay-chain can be characterized by code density test [11], [3]: an output of the ring-oscillator, whose frequency is not correlated with the system clock frequency, is used as input to the delay-chain, and the state of the delay-chain is sampled at every system clock cycle. Considering that edge of the signal can occur at any position relative to the system clock period with equal probability, the proportion of the number of detected edges in a bin corresponds to the time width of that bin. Bin widths are generally not uniform, and can differ considerably. Fig. 3 and Fig. 4 show bin widths of the delay-chain in Cyclone V and IV, respectively. Parameter $d_{step}$ value is computed as the average width of bins.

*Step 3:* Since the number of ring-oscillator stages can be chosen arbitrarily, we decided to implement it with only 3 stages, so that TRNG has a small resource consumption. Value of jitter accumulation time is selected as the smallest multiple of the system clock period - $10ns$, for which the amount of accumulated jitter is higher than $d_{step}/2$. For Cyclone IV and V we obtained $t_A = 30ns$ and $t_A = 10ns$, respectively. The number of stages in the delay-chain is chosen so that condition $m \cdot d_{step} > d_{RO\_stage}$ is fulfilled. For Cyclone IV this parameter is $m > 10$, and for Cyclone V it is $m > 58$. Since Cyclone IV has 16 LEs in one LAB (logic array block), and each LE has one carry-chain stage, we decided to use all 16 stages of carry-chain in one LAB. In Cyclone V, there are 10 ALMs in one LAB, and each ALM has two stages of carry-chain, meaning that we need to use 3 LABs (60 stages).

*Step 4:* Unequal bin widths in delay-chains lower the performance of the TRNG, since more bits need to be xored together in parity filter post-processing block so that the bias is reduced. Therefore, at this step, we propose architectural modifications of the DC-TRNG, specifically in the priority encoder, that will improve throughput of the DC-TRNG without affecting its security level. Differential nonlinearity (DNL) of the bin $i$ in the delay-chain is defined as $DNL_i = \mid H[i] - H_{ideal} \mid / H_{ideal}$, where $H[i]$ is the actual width of bin $i$, and $H_{ideal}$ is the ideal bin width – average of all bins in delay-chain [11]. We perform adaptive bin calibration on the bins of the delay-chain. Adaptive bin calibration maps bins of the original delay-chain to bins of the new *virtual* delay-chain so that the bins of the *virtual* delay-chain have smaller average $DNL$. One bin of the original delay-chain can only be mapped to one bin of *virtual* delay-chain, but several consecutive bins of the original delay-chain can be mapped to the same bin of *virtual* delay-chain. Average bin width of the *virtual* delay-chain will be larger, resulting in larger accumulation time for jitter $t_A$, but since average $DNL$ is smaller, the bias of the output bits will be lower. Adaptive bin width calibration is performed iteratively. First, we set $H_{ideal}$ of *virtual* delay-chain 1.5 times the size of $H_{ideal}$ of the original delay-chain, then we selected

TABLE I
IMPLEMENTATION RESULTS AND PERFORMANCES

| | Implementation results and performances | | |
|---|---|---|---|
| **Platform** | **Resources** | **Parity filter order** | **Throughput [Mb/s]** |
| *Spartan-6 [6]* | 67 Slices | 7 | 14.3 |
| *Cyclone IV* | 149 LEs | *2/8 | *5 / 4.16 |
| *Cyclone V* | 230 ALMs | *4/9 | *12.5 / 11.1 |
| | **Physical parameters [ps]** | | |
| **Platform** | $d_{RO\_stage}$ | $\sigma_{LUT}$ | $d_{step}$ |
| *Spartan-6 [6]* | 480 | 2 | 17 |
| *Cyclone IV* | 400 | 2.6 | *83 / 42 |
| *Cyclone V* | 435 | 1.5 | *13 / 7.5 |
| | **Implementation parameters** | | |
| **Platform** | $m$ | $n$ | $t_A$ **[ns]** |
| *Spartan-6 [6]* | 36 | 3 | 10 |
| *Cyclone IV* | 16 | 3 | *100 / 30 |
| *Cyclone V* | 60 | 3 | *20 / 10 |

*with adaptive bin width calibration

$t_A$ in the same way as in Step 3 and parity filter order big enough to reduce biases and after that calculated throughput. We then gradually increased $H_{ideal}$ of the *virtual* delay-chain, consequently increasing $t_A$ and lowering parity filter order. We stop our procedure when further increasing of $H_{ideal}$ does not contribute to increase in throughput. Information about mapping of the bins from original delay-chain to bins of *virtual* delay-chain is used to design the priority encoder block, which encodes the edge position according to the bin of *virtual* delay-chain in which it occurred. Bins of the *virtual* delay-chains for Cylcone V and IV are shown in Fig. 5 and Fig. 6, respectively.

*Step 5:* In the final step, the DC-TRNG is implemented on the target FPGA. Delay elements of the ring-oscillator have to be instantiated by using low-level primitives. Since the delay of LUT changes depending on which input is used for the incoming signal, it is necessary to precisely control which input is chosen by changing the routing constraints file. It is also necessary to enforce the desired placement of LUTs by using custom placement assignment entries for LUT of the ring-oscillator. Stages of the ring-oscillator should be placed in neighboring LABs, but in such way that carry-chains of the needed length can be implemented below each stage. Due to process variations inside one FPGA, LUTs tend to have different delays depending on the location they are placed. Thus, various locations should be examined for the position of the ring-oscillator. Therefore, careful placing and routing of the ring-oscillator is of key importance for the DC-TRNG. A designer can instantiate dedicated carry-chains in Cyclone IV and V as delay-chains by using the behavioral HDL model of an adder. In order to get a homogeneous structure of the delay-chain it is necessary to manually enforce placement of carry-chains and their corresponding flip-flops. If there is a connection between a ring-oscillator stage output and the input to its corresponding delay-chain that is longer than other connections, then there exists the possibility that the jittery edge will not be captured in any delay-chain. In order to prevent this, identical routing of these connections is performed.

### C. Implementation Results and Inter-Vendor Comparison

Table I shows implementation and performance results, platform specific physical and implementation parameters of the DC-TRNG on Xilinx FPGA and Intel FPGA. The order of parity filter is selected as small as possible so that output random bits pass all NIST tests [2]. As expected, values of the parameters have significant differences because of different FPGA structures and technology processes. Resource consumption of the DC-TRNG on both Cyclone IV (EP4CGX150DF31C7) and Cyclone V (5CEBA4F17C8) is well below 2%, satisfying the condition of compact TRNG implementation.

## IV. CONCLUSIONS

In this paper, we proposed a methodology for inter-vendor migrating of TRNGs for FPGAs. The proposed methodology was demonstrated by migrating the DC-TRNG on Intel Cyclone IV and V FPGAs. We conclude that the underlying principle of efficient entropy extraction by using carry-chain primitives is general enough to allow inter-vendor migrating of the DC-TRNG. Information about non-linearities of the delay-chain on the target FPGA was used to design a new priority encoder and in this manner improve the DC-TRNG performances.

## ACKNOWLEDGMENTS

This work is supported in part by the Research Council KU Leuven: C16/15/058. In addition, this work is supported by the Flemish Government through G.0130.13N and FWO G.0876.14N, the Hercules Foundation AKUL/11/19, and through the Horizon 2020 research and innovation programme under grant agreement No 644052 HECTOR and Cathedral ERC Advanced Grant 695305.

## REFERENCES

[1] W. Killmann and W. Schindler, "A proposal for: Functionality classes for random number generators, version 2.0," [Available online: https://www.bsi.bund.de/EN/Home/home_node.htm], 2011.
[2] A. Rukhin et al., "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," *SP 800-22, NIST*, 2010.
[3] B. Yang, V. Rožić, M. Grujić, N. Mentens and I. Verbauwhede, "On-chip jitter measurement for true random number generators," *AsianHOST 2017*, 6 pages, 2017.
[4] O. Petura, U. Mureddu, N. Bochard, V. Fischer and L. Bossuet, "A survey of AIS-20/31 compliant TRNG cores suitable for FPGA devices," *2016 26th FPL* , pp. 1-10, 2016.
[5] J.-L. Danger, S. Guilley and P. Hoogvorst, "High speed True Random Number Generator based on Open Loop Structures in FPGAs," *Microelectronics Journal*, Vol. 40, Iss: 11, pp. 1650-1656, 2009.
[6] V. Rožić, B. Yang, W. Dehaene and I. Verbauwhede, "Highly Efficient Entropy Extraction for True Random Number Generators on FPGAs," *2015 52nd DAC*, pp.1-6, 2015.
[7] M. Majzoobi, F. Koushanfar and S. Devadas, "FPGA-based True Random Number Generation Using Circuit Metastability with Adaptive Feedback Control," *CHES 2011*, pp. 17-32, 2011.
[8] Intel, "Cyclone IV Device Handbook," 2016.
[9] Intel, "Cyclone V Device Handbook," 2016.
[10] V. Fischer, F. Bernard, N. Bochard and M. Varchola, "Enhancing security of ring oscillator-based TRNG implemented in FPGA," *2008 FPL*, pp. 245-250, 2008.
[11] R. Szplet, "Time-to-digital converters," *Springer-Verlag*, pp.211-246, 2014.