TRNG architectures - literature summary

Contents

1 Introduction

TRNG based on an **entropy source** ([3]).

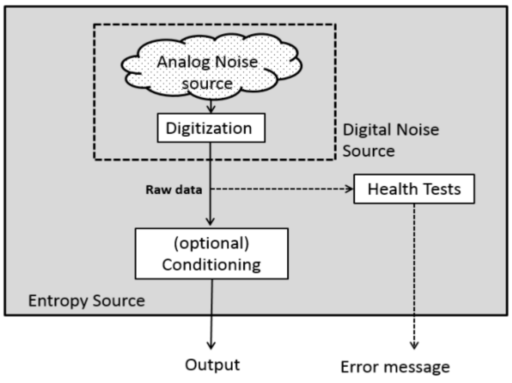


Figure 1: Entropy Source model

- **Noise Source** : the component that provides the actual entropy; it can be analog (digitization required) or digital → output: **raw data**. Noise sources can be **physical** (= dedicated hardware) or non-physical (= system data, such as output of Application Programming Interface (API) functions, RAM data or system time or human input).
- **Conditioning** : optional component, useful for **increasing the entropy rate** of output bits
- **Health Tests** : to detect **failure conditions**. Three categories: start-up tests, continuous tests (primarily on the noise source) and on-demand tests.

2 Noise Source

Most common noise sources:

- **thermal noise** → due to its small amplitude, high-gain and transimpedance amplifiers required + voltage comparators for digitization → **expensive, poor robustness**
- **chaos theory of nonlinear systems** → **complex structures and a high cost**
- **metastability** → Programmable Delay Lines (PDLs) used to carefully plan the arrival of the signal to FFs, hence forcing the violation of setup or hold time

- jitter → random bit given by FF sampling in presence of phase jitter. TRNG can be based on **jitter accumulation** (= increasing the number of transition events) or **jitter quantization** (= optimization of single propagation event).

Metastability and jitter events are often exploited together.

3 Possible implementations

3.1 Ring Oscillator approach - [9], [8], [6], [7]

A ring oscillator is composed of an odd number of inverters or delay elements connected in ring configuration. The basic idea is to exploit the oscillations caused by power supply variations, cross talks, semiconductor noise, temperature variations, and propagation delays. The period of these oscillations vary from cycle to cycle causing jitter of the rising and falling edges; **jitter follows the Gaussian distribution**. A DFF can be used for sampling the output of a high frequency oscillator, hence generating a stream of truly random bits. The digital value of the oscillators output changes with a period of approximately 2DL (D = delay of a single inverter, L = number of inverters in an oscillator).
A XOR tree after the ring oscillator or the sampling DFF can be used as conditioning circuit.
Higher number of inverters → expanded signal jitter (increased randomness) but lower frequency → **Trade-off when determining the number of LUTs in a RO**.

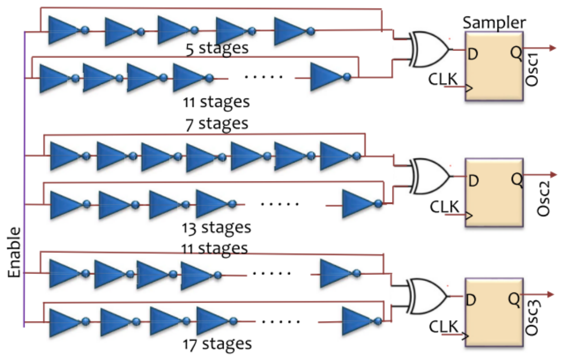


Figure 2: Ring Oscillator generic approach

In a FPGA, a LUT approach can be used to create a programmable delay inverter or path.

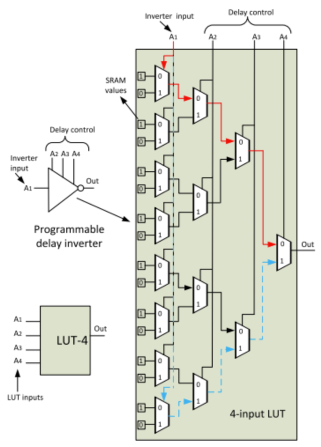


Figure 3: Programmable delay line using a 4-input LUT

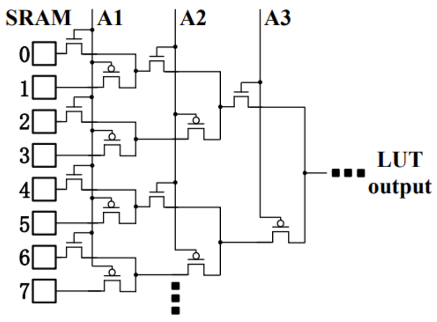


Figure 4: LUT-based Programmable delay path

3.1.1 Transition Effect Ring Oscillator (TERO) - [9]

Key idea : exploit the oscillatory metastability of latches.
CTRL signal transition → oscillations in feedback loop until latch in steady state.
Random variable : number of oscillations for each transition of CTRL input, due to intrinsic noise in semiconductors.
Output is 0 or 1 depending on even/odd oscillations.

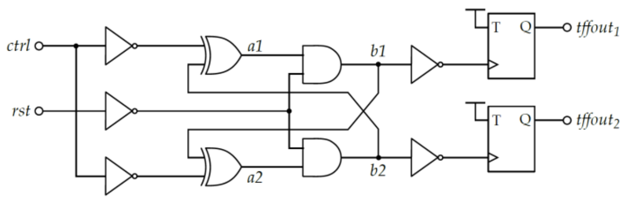


Figure 5: Transition Effect Ring Oscillator

Main problem: low throughput (Solution: asymmetry in branches but consequent reduction of standard deviation of random variable)

3.1.2 Metastable Ring Oscillator (Meta-RO) - [9]

Key idea: metastability of inverters.
mux_sel = 0 → metastable state (the metastable voltage is perturbed by low amplitude noise)
mux_sel = 1 → low amplitude noise is amplified by inverters; the ring starts in a random state; once the oscillator stabilizes to full-logic levels → sampling; delay line to accurately control the sampling instant

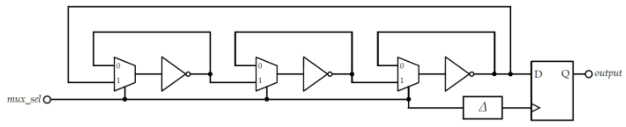


Figure 6: Metastable Ring Oscillator

3.1.3 Fibonacci and Galois Ring Oscillators (FiRO, GaRO, FiGaRO) - [9]

Key idea: DFF in Fibonacci and Galois LFSR configurations replaced by inverters to obtain an asynchronous circuit → randomness due to inverters delay depending on temperature and supply voltage. Additional randomness can be added in the sampling phase (metastability due to setup and hold violations).
Systems described by the characteristic polynomial:

$$P(x) = \sum_{i=0}^{r} f_i x^i \qquad with \quad f_0 = f_r = 1 \tag{1}$$

In both configurations, conditions must be satisfied to avoid fixed points, hence ensuring continuous oscillations. For FiRO, the conditions are:

$$\begin{cases} P(x) = (x+1)h(x) \\ h(1) = 1 \end{cases} \tag{2}$$

with $h(x)$ quotient polynomial of feedback polynomial $P(x)$.

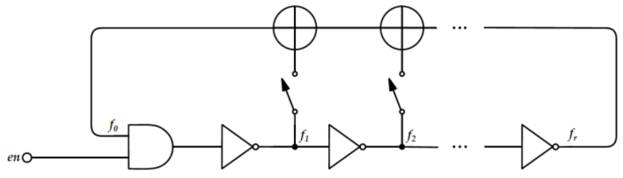


Figure 7: Fibonacci Ring Oscillator

For GaRO:

$$\begin{cases} P(1) = 0 \\ r \ is \ odd \end{cases} \tag{3}$$

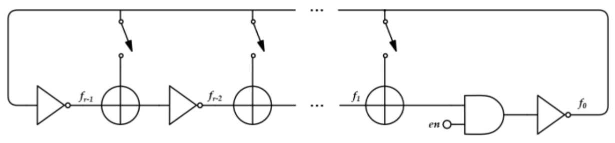


Figure 8: Galois Ring Oscillator

To further increase randomness and reliability, a mixed structure of both FiROs and GaROs can be used. The higher the number of parallel RO stages, the higher the throughput.

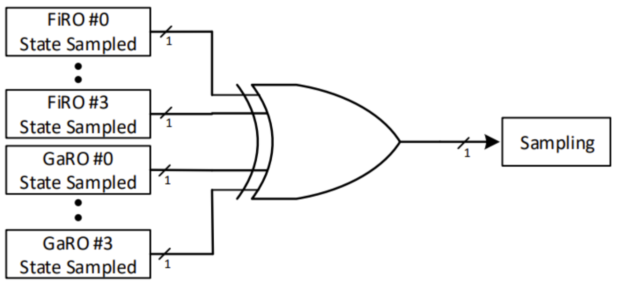
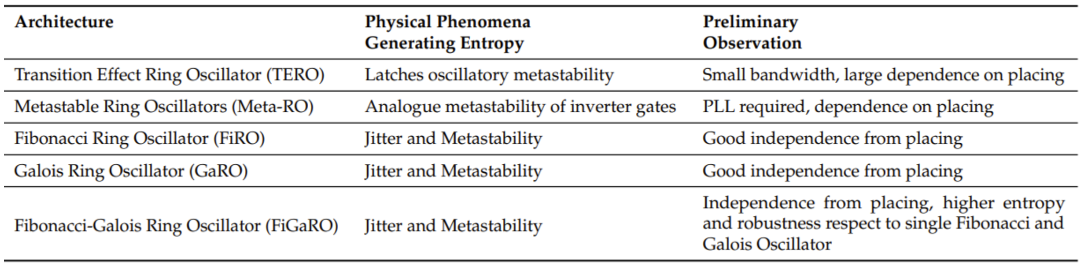


Figure 9: Fibonacci-Galois Ring Oscillator

An architecture and performance comparison of the previous five described circuits is reported in the following tables.



Architecture	Physical Phenomena Generating Entropy	Preliminary Observation
Transition Effect Ring Oscillator (TERO)	Latches oscillatory metastability	Small bandwidth, large dependence on placing
Metastable Ring Oscillators (Meta-RO)	Analogue metastability of inverter gates	PLL required, dependence on placing
Fibonacci Ring Oscillator (FiRO)	Jitter and Metastability	Good independence from placing
Galois Ring Oscillator (GaRO)	Jitter and Metastability	Good independence from placing
Fibonacci-Galois Ring Oscillator (FiGaRO)	Jitter and Metastability	Independence from placing, higher entropy and robustness respect to single Fibonacci and Galois Oscillator

Figure 10: Architecture comparison of different RO-based TRNG

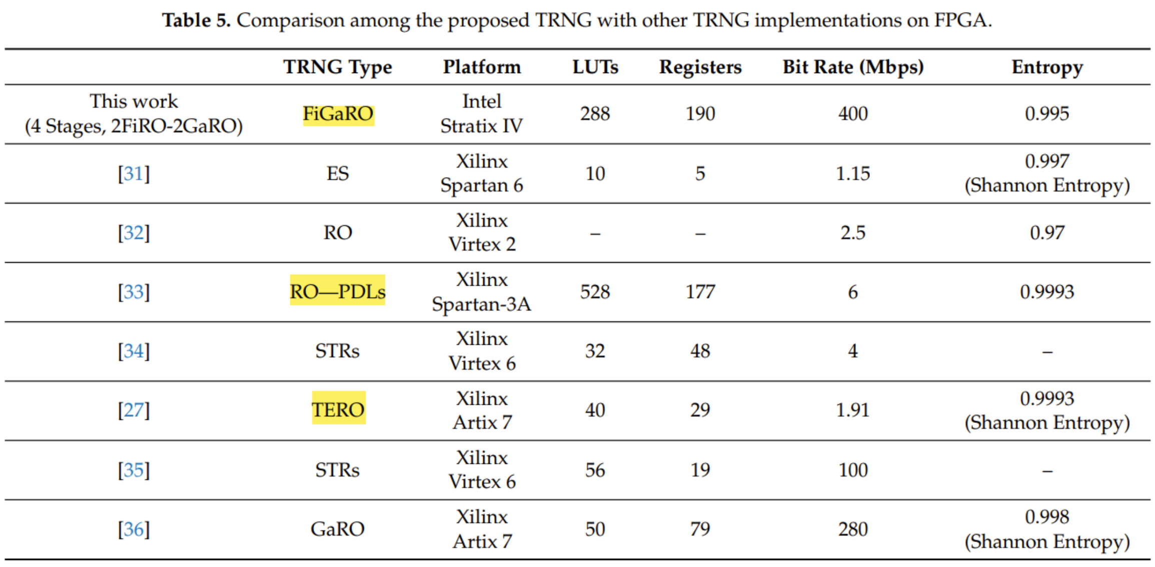


Table 5. Comparison among the proposed TRNG with other TRNG implementations on FPGA.

	TRNG Type	Platform	LUTs	Registers	Bit Rate (Mbps)	Entropy
This work (4 Stages, 2FiRO-2GaRO)	FiGaRO	Intel Stratix IV	288	190	400	0.995
[31]	ES	Xilinx Spartan 6	10	5	1.15	0.997 (Shannon Entropy)
[32]	RO	Xilinx Virtex 2	–	–	2.5	0.97
[33]	RO—PDLs	Xilinx Spartan-3A	528	177	6	0.9993
[34]	STRs	Xilinx Virtex 6	32	48	4	–
[27]	TERO	Xilinx Artix 7	40	29	1.91	0.9993 (Shannon Entropy)
[35]	STRs	Xilinx Virtex 6	56	19	100	–
[36]	GaRO	Xilinx Artix 7	50	79	280	0.998 (Shannon Entropy)

Figure 11: Performances comparison of different types of TRNG

3.1.4 Multiphase sampler - [7]

Key idea : RO-based entropy source only provides effective entropy at the sampling point; most of the time, the fast ROs are idling and wasting energy → multi-LUT RO as the sampler with multiple uncorrelated sampling points.

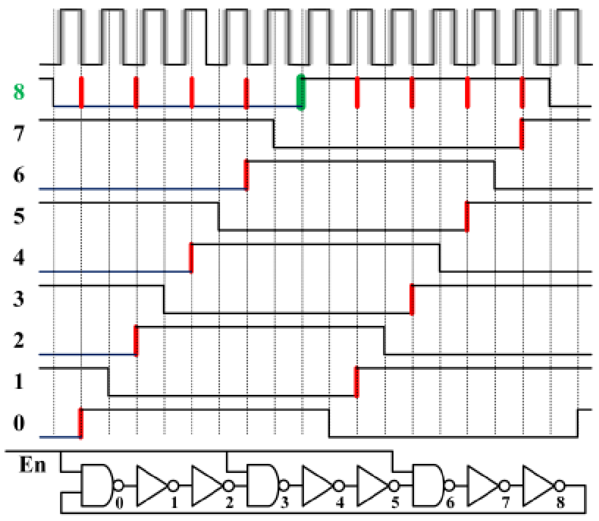


Figure 12: Multiphase sampler with multiple sampling points

Proposed architecture composed of:

- **2-LUT RO**(one inverter + one buffer) → optimal entropy
- **9-LUT RO** as sampler

Advantages:

- Improve energy efficiency
- No PLL → higher portability
- High throughput

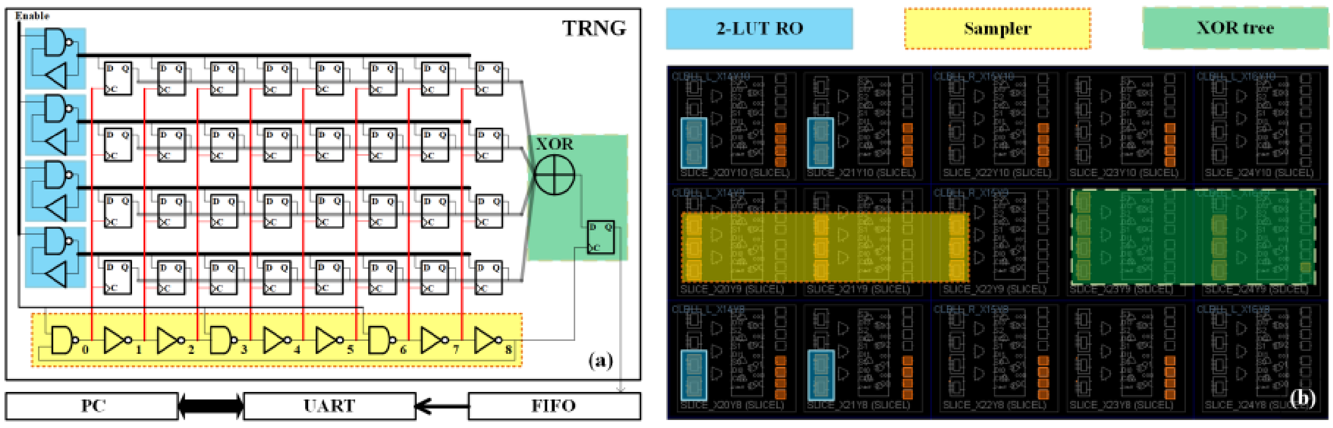
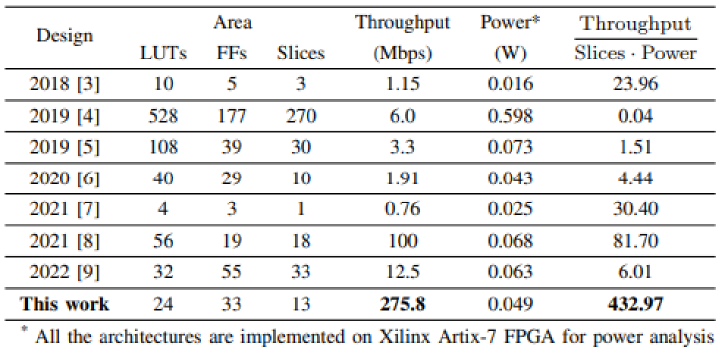


Figure 13: Architecture of the proposed TRNG.and FPGA implementation with automated placement and routing

Performances:



Design	Area			Throughput	Power*	$\frac{\text{Throughput}}{\text{Slices} \cdot \text{Power}}$
	LUTs	FFs	Slices	(Mbps)	(W)	
2018 [3]	10	5	3	1.15	0.016	23.96
2019 [4]	528	177	270	6.0	0.598	0.04
2019 [5]	108	39	30	3.3	0.073	1.51
2020 [6]	40	29	10	1.91	0.043	4.44
2021 [7]	4	3	1	0.76	0.025	30.40
2021 [8]	56	19	18	100	0.068	81.70
2022 [9]	32	55	33	12.5	0.063	6.01
This work	24	33	13	**275.8**	0.049	**432.97**

* All the architectures are implemented on Xilinx Artix-7 FPGA for power analysis

Figure 14: Resulting performances

3.1.5 RO + MLFSR - [6]

Key idea: multiple ring oscillators with a variant phase jitter, sampler, and higher-order MLFSR using primitive polynomials with an order of 67, 91, and 97 to make the robust random sequence.
NLF is a majority function of all three different outcomes of sampled output. The entropy pool is used for the post-processing block (Keccak in [6])

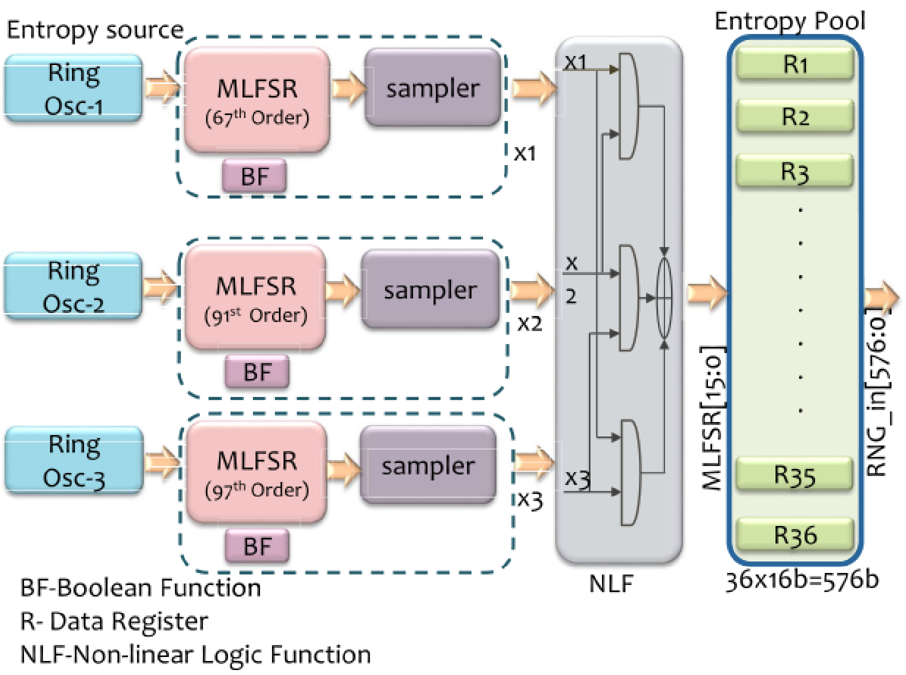


Figure 15: RO+MLFSR-based TRNG

3.2 DCM approach (FPGA only) - [5]

Digital Clock Managers (hardware primitives) are used to produce one or more output clock signals with a programmable frequency to be sampled.
Main problem : if Dynamic Partial Reconfiguration port (DRP) of DCMs is used to tune the frequency of signals driving the data and the clock input of a FF → slow random bit production rate (hundreds of clock cycles elapse between two consecutive samplings).
Solution : dynamic phase shifting (DPS) → no tuning of the frequency of the output clocks but their phase compensate the difference in routing delays of the signals arriving at the FFs.
The phase shift resolution achieved by the DCM could be not as fine as required → a carry chain primitive and four FF replicas for finer phase shift resolution control at the destination.

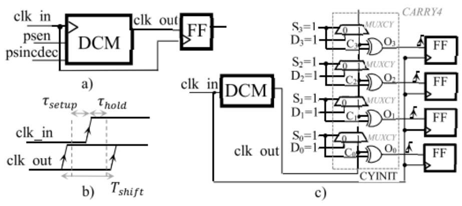


Figure 16: DCM approach

Conditioning required to pass the NIST tests and achieve sufficient entropy.

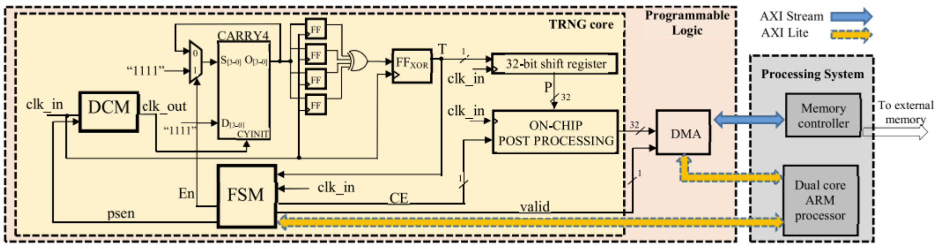


Figure 17: DCM-based TRNG

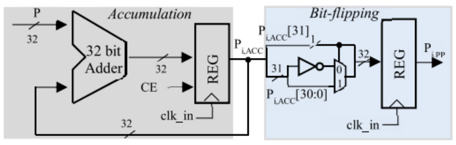


Figure 18: DCM-based TRNG - post-processing block

- **XOR gate** : to merge the outputs of the four FF;

- **FSM** : impose phase shifting transaction in DCM until signal T = 1, i.e. at least one FF in metastability;
- **Post-processing block** : accumulator + 'bit flipping' machine

Performances results:

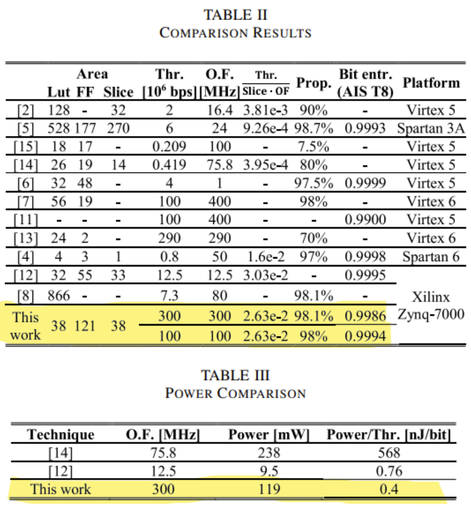


TABLE II
COMPARISON RESULTS

	Area			Thr.	O.F.	$\frac{\text{Thr.}}{\text{Slice} \cdot \text{OF}}$	Prop.	Bit entr.	Platform
	Lut	FF	Slice	$[10^6$ bps]	[MHz]			(AIS T8)	
[2]	128	-	32	2	16.4	3.81e-3	90%	-	Virtex 5
[5]	528	177	270	6	24	9.26e-4	98.7%	0.9993	Spartan 3A
[15]	18	17	-	0.209	100	-	7.5%	-	Virtex 5
[14]	26	19	14	0.419	75.8	3.95e-4	80%	-	Virtex 5
[6]	32	48	-	4	1	-	97.5%	0.9999	Virtex 5
[7]	56	19	-	100	400	-	98%	-	Virtex 6
[11]	-	-	-	100	400	-	-	0.9900	Virtex 5
[13]	24	2	-	290	290	-	70%	-	Virtex 6
[4]	4	3	1	0.8	50	1.6e-2	97%	0.9998	Spartan 6
[12]	32	55	33	12.5	12.5	3.03e-2	-	0.9995	Xilinx Zynq-7000
[8]	866	-	-	7.3	80	-	98.1%	-	
This work	38	121	38	300	300	2.63e-2	98.1%	0.9986	
				100	100	2.63e-2	98%	0.9994	

TABLE III
POWER COMPARISON

Technique	O.F. [MHz]	Power [mW]	Power/Thr. [nJ/bit]
[14]	75.8	238	568
[12]	12.5	9.5	0.76
This work	300	119	0.4

Figure 19: DCM - DPS performances (highlighted)

3.3 PLL approach - [4], [2]

A high-frequecy clock ($clk1$ in figure 20) is generated by a PLL and sampled by a DFF working with a lower frequency clock ($clk0$).
The PLL is characterized by a divison factor K_D and a multiplication factor K_M. Parameters:

- $f_{out} = \frac{K_M}{K_D} f_0$;
- Bit rate R = $\frac{f_0}{K_D}$;
- Sensitivity to jitter S = $\Delta^{-1} = \frac{K_D}{T_{out}}$

Ideal case: no jitter → periodic sampler output with period $T_P = K_D T_0$. Solution: synchronous counter to restart the decimator after each period T_P. The decimator is used for XOR-ing K_D samples and obtain 1-bit output.
Real case: jitter due to local noises → random variable : period of clk1.

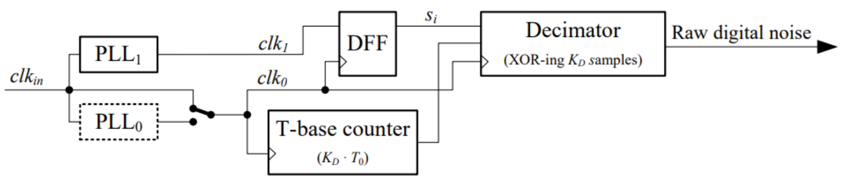


Figure 20: PLL-based TRNG

By reconstructing the period of the sampled signal $clk1$ thanks to the coherent sampling principle, it can be observed that only few samples will have a random value. The entropy rate will depend on the initial phase ϕ_0, the time resolution Δ and the standard deviation σ of the random variable $clk1$. This is why the decimator is needed. Higher K_D means lower time resolution, higher sensitivity to jitter and higher entropy rate. Problem : **Tradeoff entropy rate - output bitrate** $\rightarrow$ solution: since $S = K_M f_0$, increase the entropy rate by increasing K_M and/or f_0. An additional PLL (PLL0 in figure 20) can be used to obtain suitable K_M and K_D factors ($K_M = K_{M1}K_{D0}$, $K_D = K_{D1}K_{M0}$).
Algorithms exist to find suitable values for K_M and K_D ([1]).

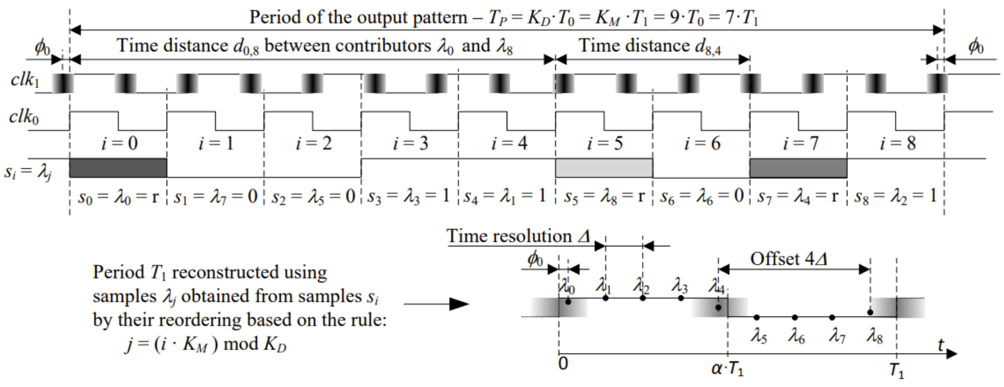


Figure 21: Reconstruction of one sampled clock period T1 by reordering successive K_D samples, ϕ_0 is the initial relative phase shift between clk1 and clk0

The basic architecture can be improved (figure 22) with a series of modifications:

- **More PLL outputs** to increase entropy;
- second DFF to solve possible metastability;
- 1-bit decimator replaced by an **m-bit time-to-digital converter** (TDC) $\rightarrow$ useful for testing;

- the least significant converter bit cnt(0) is used as the raw random signal, which is temporarily saved in a FIFO memory, whose depth depends on the latency of the total failure test;

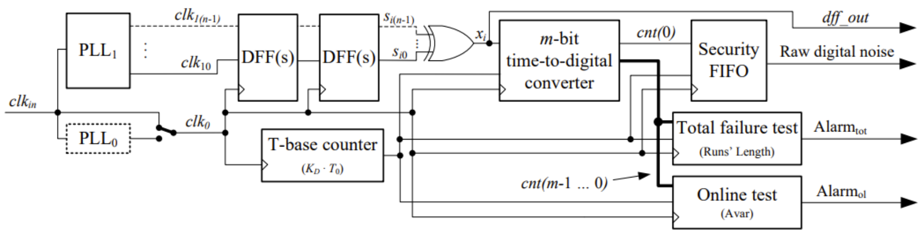


Figure 22: Enhanced PLL-based TRNG

3.3.1 Minimum complexity implementation - [2]

A low complexity and low power PLL-based TRNG is composed of a PLL, delay lines, a shift register and a counter. The delay line (IDELAYE3 in figure 24) is used to adjust the the two clocks synchronization to obtain a proper jitter value. The second delay line (IDELAYCTRL) is used for a supply voltage/temperature drift compensation.

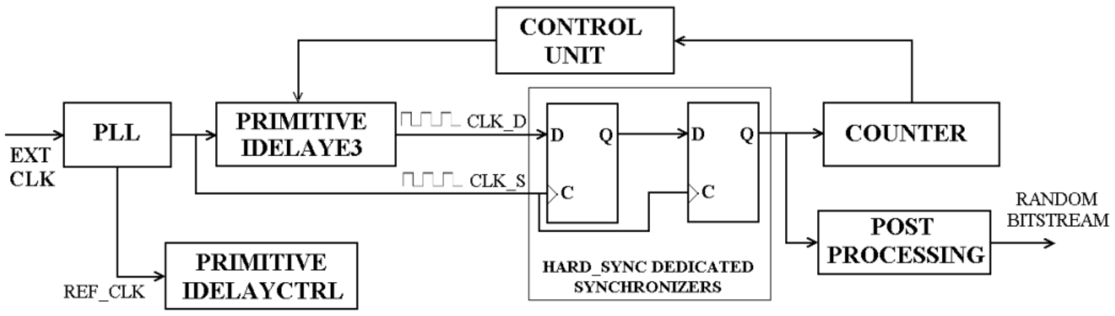
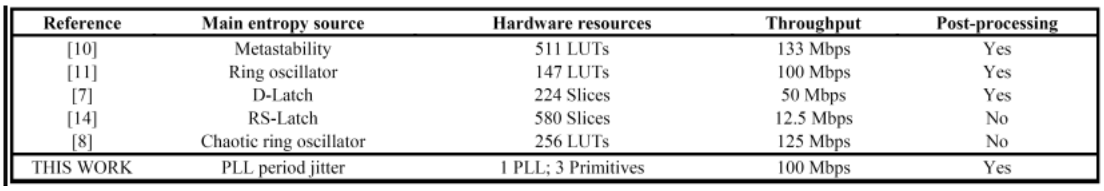


Figure 23: PLL-based TRNG - minimum complexity implementation

Results for a XOR-based post-processing:



Reference	Main entropy source	Hardware resources	Throughput	Post-processing
[10]	Metastability	511 LUTs	133 Mbps	Yes
[11]	Ring oscillator	147 LUTs	100 Mbps	Yes
[7]	D-Latch	224 Slices	50 Mbps	Yes
[14]	RS-Latch	580 Slices	12.5 Mbps	No
[8]	Chaotic ring oscillator	256 LUTs	125 Mbps	No
THIS WORK	PLL period jitter	1 PLL; 3 Primitives	100 Mbps	Yes

Figure 24: Performance results

4 Conditioning

Possible implementations of conditioning circuits have already been presented. In general, the most common ones are:

- **XOR reduction** : lower throughput, lower output correlation
- **Von Neumann corrector** : output divided in pairs $\rightarrow$ if the two bits are equal, output not generated; if differemt, only the first bit as output, second discarded. **Problem:** no-constant bit rate. **Solution**: additional buffer block.

More sophisticated conditioning circuits can be used.

4.0.1 Hash algorithm based accelerator - [11], [6]

Any hash function could be used as additional conditioning, [11] proposes a **SHA-256** implementation able to achieve a maximum output rate of **330 Mbps** at 100 MHz; [6] implements **Keccak algorithm** based on **SHA-512** hash function, maximum throughput of **2.4 Gbps** at 100 MHz (power dissipation of 380 mW).

4.0.2 Tunable distribution - [10]

In order to obtain a specific type of distribution, specific conditioning circuits can be used. Supposing to start with two indipendent gaussian distributions, **Cordic algorithm**can be used to obtain a **uniform distribution** to realize the Box-Muller transformation:

$$Z_U = \frac{1}{2\pi} \arctan \frac{Z_{GB}}{Z_{GA}} \tag{4}$$

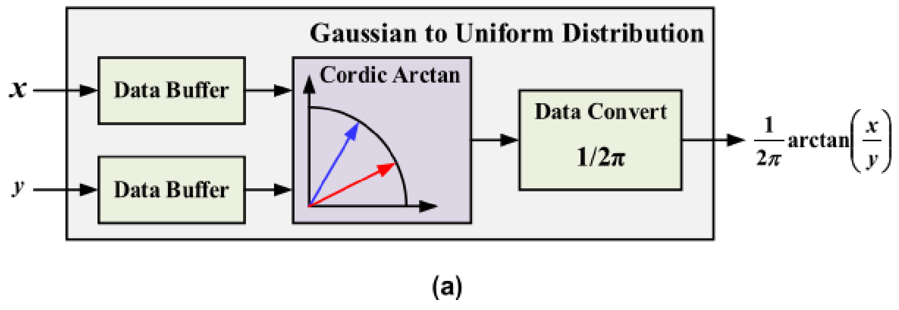

Gaussian to Uniform Distribution
x
y
Data Buffer
Data Buffer
Cordic Arctan
Data Convert
1/2π
$\frac{1}{2\pi}\arctan\left(\frac{x}{y}\right)$
(a)

Figure 25: Possible gaussian-to-uniform distribution hardware implementation

The conditioning circuit to generate a **0-1 distribution** (Bernoulli) is directly derived from the definition:

$$Z_{0-1} = \begin{cases} 1, & Z_U \geq \frac{1}{N} \sum_{j=0}^{N-1} Z_U(j) \\ 0, & Z_U < \frac{1}{N} \sum_{j=0}^{N-1} Z_U(j) \end{cases} \tag{5}$$

A RAM is needed to perform the average over N samples.

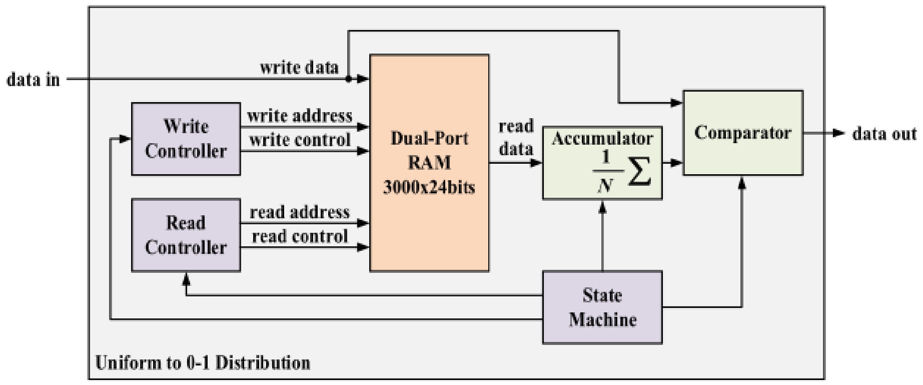


Figure 26: Possible uniform-to-0-1 distribution hardware implementation

References

[1] Elie Noumon Allini, Oto Petura, Viktor Fischer, and Florent Bernard. Optimization of the pll configuration in a pll-based trng design. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1265–1270, 2018.

[2] Guido Di Patrizio Stanchieri, Andrea De Marcellis, Marco Faccio, and Elia Palange. An fpga-based architecture of true random number generator for network security applications. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, 2018.

[3] Kerry McKay Allen Roginsky Meltem Sönmez Turan Elaine Barker, John Kelsey. Nist sp 800-90b: Recommendation for the entropy sources used for random bit generation. 2022.

[4] Viktor Fischer, Florent Bernard, Nathalie Bochard, Quentin Dallison, and Maciej Skórski. Enhancing quality and security of the pll-trng. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(4):211–237, Aug 2023.

[5] Fabio Frustaci, Fanny Spagnolo, Stefania Perri, and Pasquale Corsonello. A high-speed fpga-based true random number generator using metastability with clock managers. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 70(2):756–760, 2023.

[6] Annapurna Kamadi and Zia Abbas. Implementation of trng with sha-3 for hardware security. *Microelectronics Journal*, 123:105410, 2022.

[7] Zhaojun Lu, Houjia Qidiao, Qidong Chen, Zhenglin Liu, and Jiliang Zhang. An fpga-compatible trng with ultra-high throughput and energy efficiency. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2023.

[8] N. Nalla Anandakumar, Somitra Kumar Sanadhya, and Mohammad S. Hashmi. Fpga-based true random number generation using programmable delays in oscillator-rings. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(3):570–574, 2020.

[9] Pietro Nannipieri, Stefano Di Matteo, Luca Baldanzi, Luca Crocetti, Jacopo Belli, Luca Fanucci, and Sergio Saponara. True random number generator based on fibonacci-galois ring oscillators for fpga. *Applied Sciences*, 11(8):3330, Apr 2021.

[10] Gang Su, Changchun Ding, Sida Li, Zijian Liu, Zheng Gao, Junfeng Song, Shuxu Guo, and Min Tao. A method for generating true random numbers with multiple distribution characteristics. *IEEE Access*, 11:81753–81762, 2023.

[11] Kamil Witek, Massimo Caccia, Wojciech Kucewicz, Mateusz Baszczyk, Piotr Dorosz, and Łukasz Mik. A method for implementing a sha256 hardware accelerator inside an quantum true random number generator (qtrng). In *2023 30th International Conference on Mixed Design of Integrated Circuits and System (MIXDES)*, pages 251–256, 2023.