True random number generators for cryptography : Design, securing and evaluation

Oto Petura

N° d'ordre NNT : 2019LYSES053

THESE de DOCTORAT DE L'UNIVERSITE DE LYON

opérée au sein de

Laboratoire Hubert Curien

Ecole Doctorale N° 488

Sciences Ingénierie Santé

Specialité de doctorat:

Microélectronique

Soutenue publiquement le 23 octobre 2019, par :

Oto Petura

Générateurs de nombres aléatoires pour la cryptographie : Conception, sécurisation et évaluation

---

Devant le jury composé de :

Danger, Jean-Luc	PR	Télécom Paritech	Rapporteur
Dutertre, Jean-Max	PR	Ecole de Mines de Saint-Etienne	Rapporteur
Bossuet, Lilian	PR	Université Jean Monnet	Examinateur
Dumas, Cécile	Ingénieur-Chercheur	CEA-Leti	Examinateur
Haddad, Patrick	PhD	STMicroelectronics	Examinateur
Fischer, Viktor	PR	Université Jean Monnet	Directeur de thèse
Aubert, Alain	MC	Université Jean Monnet	Co-encadrant

PhD thesis from UNIVERSITE DE LYON

carried out at

Laboratoire Hubert Curien

Doctoral school N° 488

Sciences Engineering Health

PhD topic:

Microelectronics

Publicly defended on October 23^{rd}, 2019, by:

Oto Petura

True random number generators for cryptography: Design, securing and evaluation

---

In front of the jury consisting of:

Danger, Jean-Luc	Télécom Paritech	Reviewer
Dutertre, Jean-Max	Ecole de Mines de Saint-Etienne	Reviewer
Bossuet, Lilian	Université Jean Monnet	Examiner
Dumas, Cécile	CEA-Leti	Examiner
Haddad, Patrick	STMicroelectronics	Examiner
Fischer, Viktor	Université Jean Monnet	Thesis supervisor
Aubert, Alain	Université Jean Monnet	Thesis co-supervisor

This work has received funding from the European Union's Horizon 2020 research and innovation programme in the framework of the project HECTOR (Hardware Enabled Crypto and Randomness) under grant agreement No 644052.

Contents

Sommaire

List of Figures

List of Tables

List of Symbols

ASIC	Application Specific Integrated Circuit
COSO-TRNG	Coherent sampling ring oscillator based TRNG
CSV	Comma Separated Values
DFF	D flip-flop
DRC	Design Rules Check
DRNG	Deterministic Random Number Generator
EA	Evolutionary Algorithm
ERO-TRNG	Elementary ring oscillator based TRNG
FIT	Failure In Time
FPGA	Field Programmable Logic Array
GA	Genetic Algorithm
KAT	Known Answer Test
LUT	Look-up table
LVS	Layout Versus Schematic check
MAC	Multiply and accumulate
MPW	Multi Project Wafer
MTBF	Mean Time Between Failures
MURO-TRNG	Multi-ring oscillator based TRNG
NPTRNG	Non-Physical True Random Number Generator
PLL	Phase-locked loop
PLL-TRNG	Phase-locked loop based TRNG

PRNG	Pseudo-Random Number Generator
PTRNG	Physical True Random Number Generator
RNG	Random Number Generator
STR	Self-timed ring
STR-TRNG	Self-timed ring oscillator based TRNG
TERO	Transient Effect Ring Oscillator
TERO-TRNG	Transient effect ring oscillator based TRNG
TFF	T-flip flop
TRNG	True Random Number Generator

Introduction

Random numbers are widely used in many areas of our lives. We use them to pick who starts on serve in a tennis match, they control our fate in a board game, and they play an integral role in cryptography and information security. To chose the starting player in tennis, we can simply flip a coin. To play a board game, we need more than two random values and thus we use a dice. Cryptography, on the other hand, requires more than just rolling a dice in order to secure our data, digital communication, bank transactions, etc.

One of the fundamental principles of modern cryptography is the Kerchkhoffs's principle, which states that a cryptographic system must remain secure even if everything about it, except the key, is public knowledge. In other words, the whole security of all our digital information stands and falls on the security of the key. This principle puts strong requirements on the characteristics of cryptographic keys. The first requirement is that the key must never leave the cryptographic system in clear and it must be safely stored to prevent any unauthorized access to it. But it is not sufficient to only store it securely if an adversary could just guess the key. To prevent anyone from guessing the key, two conditions must be met:

— A key must be very difficult or ideally impossible to guess. If a key is based on any kind of known data, it makes the guesswork much easier. But if a key is not derived from any known data (*i.e.* random), the only way to guess it is a brute force.

— A key must be periodically renewed to prevent anyone from brute forcing it. Even if the only choice left is to brute force the key, it is still possible to do with sufficient resources and time. To prevent this from happening, we need to renew the key before an adversary can guess it.

A key satisfying these conditions can be generated by a random number generator (RNG).

In this thesis, we will deal with true random number generators (TRNGs) exploiting physical phenomena in hardware (Physical TRNGs). Since implementation of the post-processing algorithms in logic devices is quite straightforward, we aim our attention at implementation of the TRNG core. The difficulty is to find and exploit physical random phenomena, which are intrinsically random, inside logic devices that are designed to implement deterministic systems. It is a challenging task to find a proof that a TRNG is indeed using intrinsically random phenomena.

This was one of the main objectives of the HECTOR project presented in the next paragraph.

HECTOR project

The work presented in this thesis was done in the framework of European research project HECTOR (Hardware Enabled CrypTo and Randomness). Main research topics of this project are development of cryptographic primitives, such as TRNGs, PUFs (Physical Unclonable Functions), and authenticated encryption algorithms, and their integration in a complete cryptographic system. The project efforts are heavily driven by industrial requirements since 6 out of 9 partners are industrial partners, three of them small or middle enterprises. Partners of HECTOR project include:

— Technikon Forschungs–und Planungsgesellschaft mbH, Austria
— Katholieke Universiteit Leuven, Belgium
— Université Jean Monnet Saint-Etienne, France
— Thales Communications & Security SAS, France
— STMicroelectronics Rousset SAS, France
— STMicroelectronics SRL, Italy
— Micronic AS, Slovakia
— Technische Universität Graz, Austria
— Brightsight BV, Netherlands

The main objectives of the project were:

— Efficient implementations of state-of-the-art cryptographic algorithms, as well as resistance to physical attacks. Efficiency objectives can be multi-dimensional: low-area or low memory footprint (e.g. for portable embedded applications), high throughput (e.g. for cloud applications), power-efficiency (when limited power supply or cooling of high end systems), energy-efficiency (for battery operated devices), or low latency (for real-time applications). Multiple objectives may have to be combined.
— Cryptographic algorithms, protocols and many countermeasures against physical attacks expect perfect random numbers, yet in reality they are difficult to generate. A major objective of the HECTOR project was to provide robust and high entropy random numbers including quality metrics. HECTOR provided design, models, implementation, evaluations and advanced tests and robustness evaluations of random number generators and PUFs. It also provided methods and procedures for on-the-fly entropy testing.
— Cryptographic algorithms and many physical security countermeasures such as masking, will fail with poor quality random numbers (e.g. after manipulation). HECTOR aimed at mastering gradual degradation of security levels of cryptographic primitives and hardware

security countermeasures as a function of randomness quality. Existing state-of-the-art algorithms (e.g. AES or KECCAK) were investigated and novel cryptographic primitives that are error or noise tolerant were developed.

— Cryptographic implementations also need to be resistant to attacks. Countermeasures against physical attacks, such as side-channel attacks, fault and electro-magnetic perturbation attacks are expensive in terms of silicon area, execution time, power or energy consumption. Thus another objective of this project was to balance efficiency and robustness and to aim at more efficient countermeasures.

— Efforts were driven by practical challenges, requirements and use cases provided by the industrial partners of the project.

— HECTOR provided inputs to standardization and certification efforts such as hardware-friendlier/friendliest cryptographic algorithms or protocols, e.g. light weight algorithms or authenticated encryption. It also provided inputs towards certification and standardization regarding quality testing and evaluation of random numbers.

Most of the effort in HECTOR project was focused on random number generation – TRNG and PUF implementation and testing. Practical requirements driven by industrial partners required a new specialized evaluation platform. So HECTOR evaluation boards [3] were created especially for TRNG and PUF testing.

Thesis objectives

This thesis deals with TRNG development within the frame of industrial requirements of HECTOR project. The objectives of the thesis are hence based on the objectives of the HECTOR project, namely:

— Implementation and evaluation of high entropy TRNGs and design, implementation and evaluation of dedicated embedded tests for TRNGs.

— On top of the HECTOR requirements, we also focus on study and optimization of the PLL-TRNG design and oscillator based TRNGs in general.

— We wish to provide automated tools for the PLL-TRNG design, which would enable rapid development of high quality TRNGs within different technological constraints.

— For oscillator based TRNGs, we are searching for efficient methods of randomness extraction and embedded testing, which would improve both bit rate and quality of random numbers produced.

Introduction

Les nombres aléatoires sont largement utilisés dans de nombreux domaines de notre vie. Nous nous en servons pour choisir qui commence au service dans un match de tennis, ils contrôlent notre destin dans un jeu de société et ils jouent un rôle essentiel dans la cryptographie et la sécurité de l'information. Pour choisir le premier joueur de tennis, il suffit de lancer une pièce de monnaie. Pour jouer à un jeu de société, nous avons besoin de plus de deux valeurs aléatoires et nous utilisons donc un dé. La cryptographie, en revanche, nécessite plus que de lancer un dé pour sécuriser nos données, nos communications numériques, nos transactions bancaires, etc.

L'un des principes fondamentaux de la cryptographie moderne est le principe de Kerchkhoff, selon lequel un système cryptographique doit rester sécurisé même si tout ce qui le concerne, à l'exception de la clé, est de notoriété publique. En d'autres termes, toute la sécurité de toutes nos informations numériques repose sur la sécurité de la clé. Ce principe impose de fortes exigences sur les caractéristiques des clés cryptographiques. La première exigence est que la clé ne doit jamais sortir du système cryptographique en clair et qu'elle doit être stockée en toute sécurité pour empêcher tout accès non autorisé à celle-ci. Mais il ne suffit pas de la stocker en toute sécurité si un adversaire peut deviner la clé. Pour empêcher quiconque de deviner la clé, deux conditions doivent être remplies :

— Une clé doit être très difficile ou idéalement impossible à deviner. Si une clé est basée sur n'importe quel type de données connues, c'est beaucoup plus facile de la deviner. Mais si une clé n'est dérivée d'aucune donnée connue (*i.e.* aléatoire), le seul moyen de la deviner est par la force brute.

— Une clé doit être renouvelée périodiquement pour empêcher quiconque de la deviner par la force brute. Même si le seul choix qui reste est de deviner la clé par la force brute, c'est encore possible de le faire avec les ressources et le temps suffisants. Pour éviter complètement cette possibilité, nous devons renouveler la clé avant qu'un adversaire puisse la deviner.

Une clé qui remplit ces conditions peut être générée par un générateur de nombres aléatoires (RNG).

Dans cette thèse, nous nous intéressons aux générateurs de nombres véritablement aléa-

toires (TRNGs) exploitant des phénomènes physiques dans les circuits électroniques (TRNG physiques). L'implémentation des algorithmes de post-traitement dans les circuits logiques étant assez simple, nous concentrons notre attention sur l'implémentation du noyau du TRNG. La difficulté est de trouver et d'exploiter des phénomènes aléatoires physiques, intrinsèquement aléatoires, à l'intérieur de circuits logiques, destinés à l'implémentation de systèmes déterministes. C'est un défi de prouver qu'un générateur de nombres aléatoires utilise effectivement des phénomènes intrinsèquement aléatoires. C'était un des objectifs principaux du projet HECTOR présenté dans le paragraphe suivant.

Projet HECTOR

Le travail présenté dans cette thèse a été réalisé dans le cadre du projet de recherche européen HECTOR (CrypTo et Randomness Enabled Hardware). Les sujets principaux de recherche de ce projet sont le développement de primitives cryptographiques, telles que les TRNG, les PUF (fonctions physiques non clonables) et les algorithmes de chiffrement authentifiés, ainsi que leur intégration dans un système cryptographique complet. Les efforts du projet sont fortement motivés par les exigences industrielles puisque six partenaires sur neuf sont des partenaires industriels, dont trois petites ou moyennes entreprises. Les partenaires du projet HECTOR comprennent :

— Technikon Forschungs-und Planungsgesellschaft mbH, Autriche
— Katholieke Universiteit Leuven, Belgique
— Université Jean Monnet Saint-Etienne, France
— Thales Communications & Security SAS, France
— STMicroelectronics Rousset SAS, France
— STMicroelectronics SRL, Italie
— Micronic AS, Slovaquie
— Technische Universität Graz, Autriche
— Brightsight BV, Pays-Bas

Les principaux objectifs du projet étaient:

— L'implémentation efficace d'algorithmes cryptographiques de l'état de l'art, ainsi que leur résistance aux attaques physiques. Les objectifs d'efficacité peuvent être multidimensionnels : faible surface ou faible empreinte mémoire (par exemple pour les applications embarquées portables), débit élevé (par exemple pour les applications sur le cloud), efficacité en puissance (lorsque l'alimentation électrique ou le refroidissement des systèmes haut de gamme sont limités), efficacité énergétique (pour les appareils alimentés par batterie), ou faible latence (pour les applications temps réel). Il peut être nécessaire de combiner plusieurs objectifs.

— Les algorithmes cryptographiques, les protocoles et de nombreuses contre-mesures contre les attaques physiques attendent des nombres aléatoires parfaits, mais ils sont en réalité difficiles à générer. L'un des objectifs principaux du projet HECTOR était de fournir des nombres aléatoires robustes à entropie élevée, ainsi que des mesures de qualité. HECTOR a fourni la conception, les modèles, la mise en œuvre, les évaluations, ainsi que les tests avancés et les évaluations de robustesse des générateurs de nombres aléatoires et des PUF. Il a également fourni des méthodes et des procédures pour les tests d'entropie à la volée.

— Les algorithmes cryptographiques et de nombreuses contre-mesures de sécurité physique, telles que le masquage, échoueront avec des nombres aléatoires de qualité médiocre (par exemple, après manipulation). HECTOR visait à maîtriser la dégradation progressive des niveaux de sécurité des primitives cryptographiques et des contre-mesures de sécurité matérielles en fonction de la qualité du caractère aléatoire. Des algorithmes de pointe existants (par exemple, AES ou KECCAK) ont été étudiés et de nouvelles primitives cryptographiques qui tolèrent les erreurs ou le bruit ont été développées.

— Les implémentations cryptographiques doivent également être résistantes aux attaques. Les contre-mesures contre les attaques physiques, telles que les attaques par canaux auxiliaires, les attaques en fautes ou par perturbations électromagnétiques sont coûteuses en termes de surface de silicium, de temps d'exécution, de consommation énergétique. Un autre objectif de ce projet était donc de trouver un équilibre entre efficacité et robustesse et de rechercher des contre-mesures plus efficaces.

— Les efforts ont été motivés par les défis pratiques, les exigences et les cas d'utilisation fournis par les partenaires industriels du projet.

— HECTOR a contribué aux efforts de normalisation et de certification en proposant des algorithmes ou des protocoles cryptographiques plus conviviaux d'un point de vue matériel ou algorithmique, comme les algorithmes légers ou le cryptage authentifié. Il a également contribué à la certification et à la normalisation de tests de qualité et d'évaluation de nombres aléatoires.

Le projet HECTOR s'est principalement concentré sur la génération de nombres aléatoires – mise en œuvre et tests de TRNG et de PUF. Les exigences pratiques dictées par les partenaires industriels nécessitaient une nouvelle plate-forme d'évaluation spécialisée. Les cartes d'évaluation HECTOR [3] ont donc été crées spécialement pour les tests de TRNG et PUF.

Objectifs de la thèse

Cette thèse traite du développement de TRNG dans le cadre des exigences industrielles du projet HECTOR. Les objectifs de la thèse sont donc basés sur les objectifs du projet HECTOR,

à savoir:

— Implémentation et évaluation de TRNG à haute entropie et conception, implémentation et évaluation de tests embarqués dédiés pour les TRNG.

— Outre les exigences d'HECTOR, nous nous concentrons également sur l'étude et l'optimisation de la conception du TRNG basé sur les PLLs et des TRNG basées sur les oscillateurs en général.

— Nous souhaitons fournir des outils automatisés pour la conception du PLL-TRNG, qui permettraient le développement rapide de TRNG de haute qualité avec différentes contraintes technologiques.

— Pour les TRNG basés sur les oscillateurs, nous recherchons des méthodes efficaces d'extraction d'aléa et des tests intégrés permettant d'améliorer à la fois le débit et la qualité des nombres aléatoires produits.

Chapter 1

Random number generators in cryptography – state of the art

Cryptography is used in every information system nowadays and random number generators (RNGs) are an essential part of any cryptographic system. In a cryptographic system, RNGs are used (not only) to generate cryptographic keys, but also nonces, initialization vectors, and random masks for protection against side channel attacks.

Despite there being a lot of different applications of random numbers in a cryptographic system they all share two basic requirements:

Good statistical properties, namely uniform probability distribution. Every value of any random number used in a cryptographic system must be equally likely to appear. This requirement is of utmost importance since a biased probability distribution would open the door to an attacker e.g. by making frequency attacks possible.

Unpredictability of random numbers. Random numbers, especially those used for secret parameters such as keys, must be unpredictable to prevent an attacker from being able to compute future or preceding values from the already generated and captured data.

Given the vast spectrum of RNG applications in cryptography, it is only natural that many different RNG principles exist to satisfy their various needs. Based on the method used to generate random numbers, we distinguish two fundamental RNG types:

Deterministic/Pseudo random number generator (DRNG, PRNG) is a system, which produces random-looking sequence mathematically. Produced numbers seem random in short term, but the sequence is periodic, usually with a long period. In order to produce less predictable output, these RNGs use initialization values called seeds to start from. For

every seed, there is a different sequence generated. Output sequence of a good DRNG is perfectly uniformly distributed. DRNGs achieve high output bit rates.

True random number generators (TRNGs), on the other hand, are not algorithmic, but instead they are systems, which extract randomness from non-algorithmic random phenomena. These phenomena may be temperature fluctuations, radioactive decay, ambient radio noise, hard disk access times, or user interactions with the PC. Since the phenomena used are intrinsically unpredictable, TRNGs produce real random data instead of just random-looking periodic sequences. The behavior of a TRNG is not defined by a mathematical formula, which is the case of DRNGs. Since the quality of generated random sequence depends on physical properties, the output sequence may exhibit some statistical defects such as bias. TRNGs are in general slower than DRNGs.‘ Based on the source used, they can be further divided to:

— *Physical TRNG (PTRNG)* uses physical noise on electron level present in all semiconductors. A PTRNG is a physical device and uses physical noise.

— *Non-physical TRNG (NPTRNG)* may not be a physical device, but instead a piece of software. It uses non-physical randomness source such as user interactions with an operating system.

Unpredictability of deterministic random number generators is guaranteed computationally, while unpredictability of truly random generators is guaranteed by random physical phenomena and characterized by the entropy rate at generator output.

Both TRNGs and DRNGs have their advantages and disadvantages and hence many cryptographic systems use hybrid RNGs, which combine the strengths of TRNGs and DRNGs. Based on their implementation, there are two types of hybrid RNGs:

Hybrid true random number generators, which combine a TRNG with a cryptographic post processing. The cryptographic post processing assures the forward and backward secrecy of produced random numbers (neither past, nor future values can be computed from the present value). If the physical source fails, it also guarantees perfect statistical properties of the output data since the core of a cryptographic post processing is usually a cipher. The output bit rate of a hybrid TRNG is limited by that of the TRNG core.

Hybrid deterministic random number generators. They use a TRNG to periodically generate seeds for a DRNG. Since the output of a DRNG is predictable if we know its seed, often reseeding using a TRNG can reduce predictability of a hybrid DRNG. Additionally, the output sequence of such a generator is perfectly uniform, which might not be a case for a pure TRNG. Their output bit rate is determined by the bit rate of the underlying DRNG because random numbers may be produced until the repetition period of a DRNG is not

reached.

In order to prevent attacker to access to the confidential keys, the keys must be generated inside the cryptographic system. Since the current cryptographic systems implement essentially cryptographic algorithms and protocols, they are mostly implemented in logic devices and digital systems. Therefore, it was quite natural that we oriented our research towards implementation of random number generators in logic devices, namely Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs), both of which have hardware support for digital logic synthesis.

In the following, we will therefore limit our focus only on RNGs suitable for implementation in logic devices.

1.1 Sources of randomness in logic devices

TRNGs may use either physical or non-physical noise sources. In logic devices, physical noise sources are quite limited, since logic devices are supposed to always be in a well-defined state. In order to generate random numbers, we need an uncontrollable random phenomenon. Physical phenomena most commonly used to generate random numbers in logic devices are:

Clock jitter, which is a variation of the clock edge from its ideal position.

Metastability is an ability of a circuit to persist in an undefined state for indefinite period of time.

Chaos is an unpredictable behavior of a deterministic system, which is highly sensitive to its initial conditions.

Analog signals such as shot noise of a diode, thermal noise, etc.

Clock jitter and metastability will be discussed further in this chapter.

Generating random numbers using analog signals is out of scope of this thesis because they are difficult to exploit in logic devices. An analog to digital converter is needed in order to use an analog signal in a digital device and most of the digital logic devices we focus on (FPGAs and ASICs) do not have such an analog interface.

Chaotic behavior is a kind of behavior of a seemingly deterministic system. A chaotic system is extremely sensitive to its initial conditions, which means that even the slightest change in the initial state produces very different results. This behavior has been studied for TRNG implementation, since the divergence of results of different initial states breaks the dependencies in output sequence. Such systems need analog components such as A/D converters [4] or switched capacitors [5]. In this thesis, we will focus on the sources of randomness which do not need such components since they are not available in logic devices in general.

1.1.1 Clock jitter

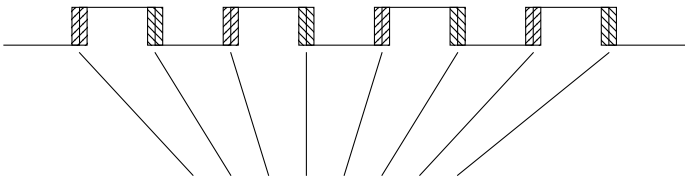
An ideal clock signal in digital logic devices is supposed to be a rectangular signal with a 50% duty cycle and stable period. But due to various noises affecting electronic devices, the clock signal is never absolutely stable and its edges move from their stable position. In other words, the phase of the clock signal fluctuates. This fluctuation can be seen as a clock jitter in time domain and as a phase noise in the frequency domain. In logic devices, the clock jitter is usually unwanted, but inevitable. Since the jitter is negatively affecting high-frequency communications and high-speed systems, it has been well-studied and characterized.

In analog systems, the jitter is best characterized in the frequency domain. This way, the phase and amplitude components can be studied and characterized separately. In digital systems, on the other hand, temporal properties of the jitter are more important and thus the jitter is characterized in time domain.

Clock jitter in a digital system is a deviation of the actual clock edge from an ideal clock edge. An ideal clock signal is defined by Equation (1.1), where $t_(n)$ represents the time of n-th period of a clock signal and T is the period of a clock signal.

$$t(n) = n \cdot T \tag{1.1}$$

A real clock signal does not arrive always at integer multiples of its period, as the ideal one does, but its edges are fluctuating around this value because of the jitter. The jitter is caused by various physical phenomena including thermal noise, power supply noise, ambient electromagnetic noise, etc. . Figure 1.1 shows how jittered clock looks like.



Depending on the jitter size, the clock edge may arrive anywhere within these regions

Figure 1.1: Clock jitter

Figure 1.2 shows the main cause of the jitter in digital circuits. Digital circuits use a reference level, usually in the middle of operating voltage range, in order to detect clock edges. This reference level should be as stable as possible, but in reality it fluctuates because of various noises. When the reference level shifts, it causes the clock edge to be detected sooner or later as it normally would be. This temporal shift in clock detection moment is observed as the clock jitter.

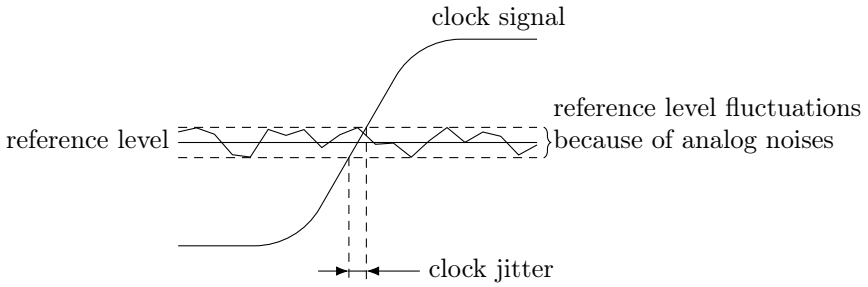


Figure 1.2: Reference level fluctuations originating from analog noises causing clock jitter in digital circuits

In the following sections, we explain different jitter measurements that we observe in digital circuits and relations between them.

Phase jitter

The phase jitter is a difference of the time of n-th real clock edge ($t_r(n)$) and the time (phase) of n-th ideal clock edge. Equation (1.2) defines this relation.

$$\delta_\varphi(n) = t_r(n) - n \cdot T_{ref} \tag{1.2}$$

Figure 1.3 illustrates this jitter for $n = 3$. For clarity, we illustrate the phase jitter of rising edges only, but it should be noted that the phase jitter affects every edge of the clock.

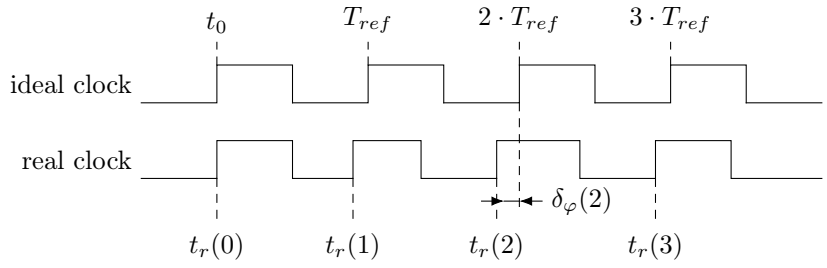


Figure 1.3: Illustration of the phase jitter of the second rising edge of the clock signal

In Figure 1.3, we can see that the displayed phase jitter $\delta_\varphi(2)$ is not affected only by the phase deviation of $t_r(2)$ but contains also contributions of the deviation of $t_r(1)$. This is referred to as jitter accumulation and it causes that with larger n the observed phase jitter rises.

Period jitter

The period jitter is the difference between a real clock period and that of an ideal one. It is also, as Equation (1.3) defines, a first order difference of the phase jitter.

$$\begin{aligned} \delta_T(n) &= [t_r(n) - t_r(n-1)] - T_{ref} \\ \delta_T(n) &= \delta_\varphi(n) - \delta_\varphi(n-1) \end{aligned} \tag{1.3}$$

Figure 1.4 shows the period jitter. We can see, that real periods change over time as opposed to ideal periods, which stay constant.

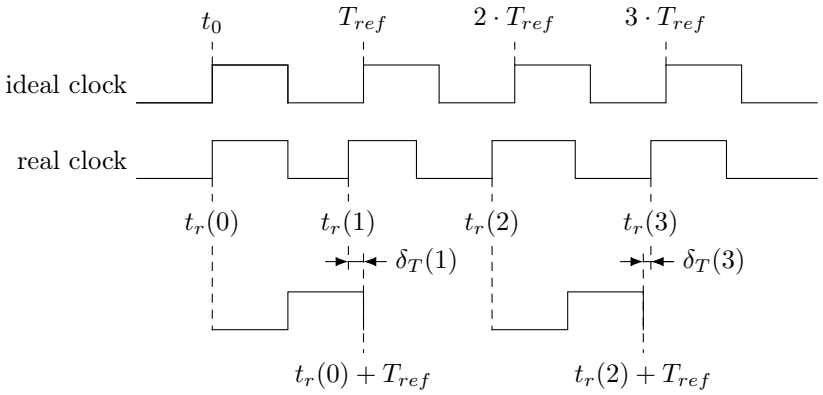


Figure 1.4: Illustration of the period jitter of a real clock signal compared to the ideal clock

Cycle to cycle jitter

The cycle to cycle jitter is a difference of two consecutive real clock periods, as defined by Equation (1.4).

$$\begin{aligned} \delta_c &= T_r(n) - T_r(n-1) \\ &= [t_r(n) - t_r(n-1)] - [t_r(n-1) - t_r(n-2)] \\ \delta_c &= \delta_T(n) - \delta_T(n-1) \end{aligned} \tag{1.4}$$

Figure 1.5 illustrates this jitter.

All these jitter measurements are mutually related: the period jitter is the first order difference of the phase jitter and the cycle to cycle jitter is the first order difference of the period jitter. Therefore, it is in general sufficient to measure only one and compute any other, which might be needed.

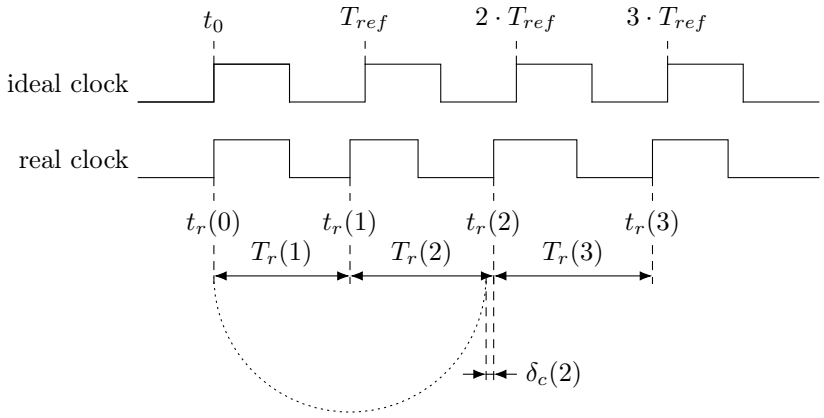


Figure 1.5: Illustration of the cycle to cycle jitter

Jitter components

Jitter has various components, which are caused by different phenomena. These components can be either random, or deterministic. Random components, such as those originating from the thermal or $1/f$ noise, are unpredictable and follow some kind of the probabilistic law. Deterministic components are implementation dependent, which means they depend on specific effects such as processed data and power supplies used. The deterministic components do not follow any probabilistic law because of their deterministic nature, which makes them generally impossible to characterize. Figure 1.6 shows both deterministic and random jitter components and how they add up to compose the overall jitter present in logic devices.

Both random and deterministic jitter can have local or global sources. Local jitter sources affect only a limited area in the electronic system, while global jitter sources affect the whole system. Local jitter sources are usually present in a vicinity of high frequency and high power components such as oscillators and amplifiers. Global jitter sources are ambient noises, noises originating from power supplies, etc.

In the context of TRNGs, the deterministic jitter components are unwanted, since they do not provide any real randomness. The randomness can be extracted only from the jitter caused by random noises. But before generating random numbers from the jitter, we need to know its statistical properties.

From a statistical point of view, we recognize independent and dependent noises. Independent noises are generally not manipulable and are fairly easy to characterize. That is why most TRNG designs use the sum of independent noises, referred to as a Gaussian noise, as a source of randomness. The challenge of designing a TRNG using the Gaussian noise is that we need to estimate the contribution only of the non-correlated (Gaussian) noises to the resulting random numbers. This means ruling out contributions of dependent noises to the resulting randomness.

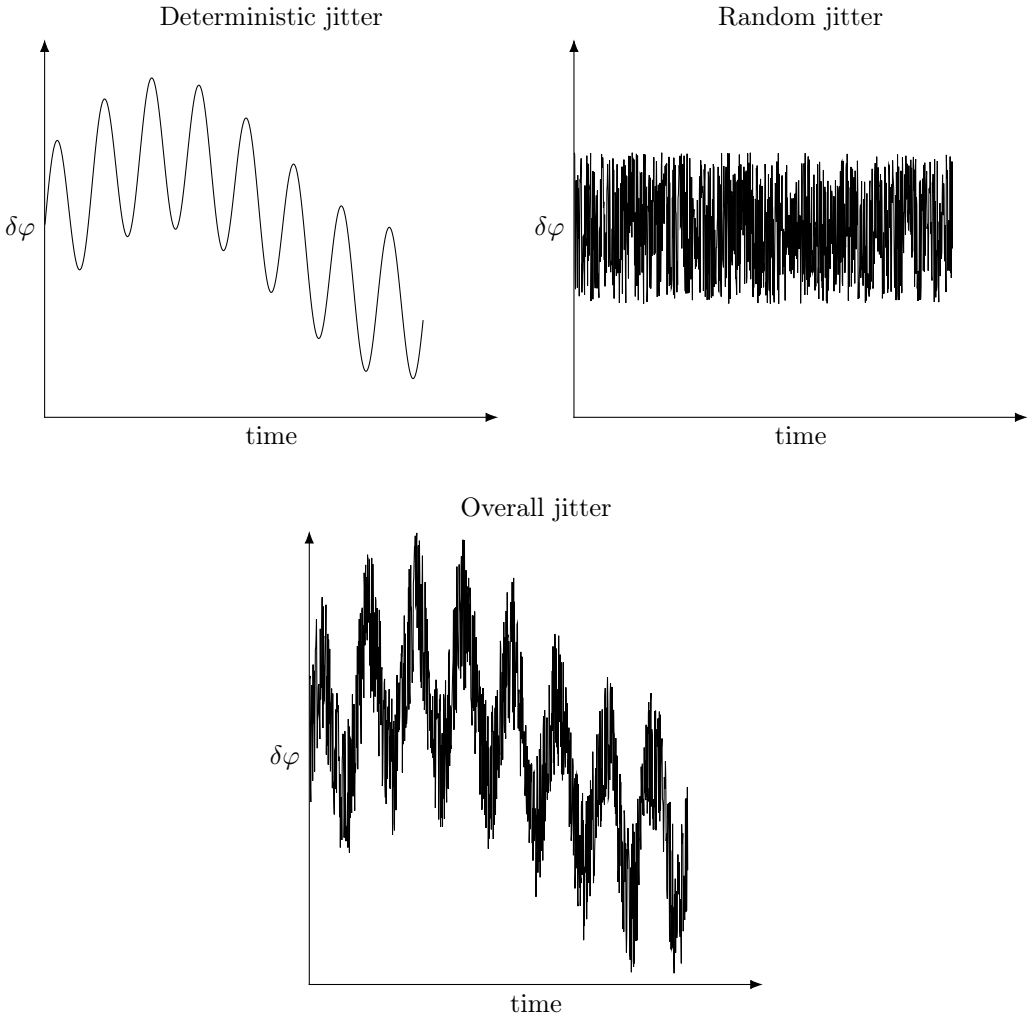


Figure 1.6: Overview of deterministic and random jitter components

There are many dependent noises. Some of them are deterministic, which we already mentioned, and they are not usable to generate random numbers. There are, however, also non-deterministic autocorrelated noises such as 1/f noise, otherwise called the flicker noise. The flicker noise is a long-known phenomenon in semiconductors. It was studied since 1950s and 1960s [6, 7], through 1990s [8], until recent time [9], when one of the research interests became TRNG application of such noise. Statistically, the flicker noise is low frequency and autocorrelated noise, which makes its use for random number generation questionable since its contribution to entropy is difficult to estimate. Even though this noise has been studied for a long time, it is still hard to explain its physical cause as well as to characterize it. So in TRNG design, we try to exclude the contribution of the flicker noise to the entropy rate estimation and use uncorrelated thermal noise only.

From the statistical point of view considering only uncorrelated random noises, the time of

arrival of the n-th clock edge is a random variable X_{t_n}. The probability distribution function of each variable has its mean value at $n \cdot T_{ref}$. The variance of these functions gives us an insight of how much the clock edge fluctuates. When we observe a clock signal, we see one particular realization of each variable X_{t_n}, as was illustrated in previous sections.

1.1.2 Metastability

Metastability is an ability of a system to persist in an illegal state for an indefinite period of time. As an example, we can take a coin flip as illustrated in Figure 1.7. When we flip a coin, we want it to land on either of the two faces. So its faces are the two legal states. But when a coin lands on its side, the result of a coin flip is indecisive, and thus illegal. This state is called a metastable state. When a coin lands on its face, it's a stable state. In order for a coin to stay in a metastable state, it must be in perfect equilibrium. Even a slightest force applied to it will cause it to fall on either one of the faces.

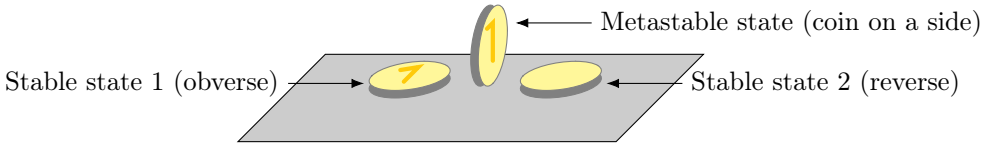


Figure 1.7: Metastability of a coin flip

1.1.2.1 Metastability in logic devices

In logic devices, metastability can occur in registers during normal operation of the device, when register setup and hold times are violated. Registers (e.g. flip-flops) require an input signal to be stable for certain time before the clock edge (setup) and also for some time after the clock edge (hold). Only then, the register is guaranteed to have a well defined value within a specified delay at the output. If the setup and hold times are not respected, the register can fall into a metastable state, where it hovers for unspecified period of time before it resolves to either its previous value (see Output 2 in Figure 1.8), or the new one (see Output 1 in Figure 1.8).

Whether the register ends up in new, or previous state is determined by random factors. Thus the result of such a transition is also random. The challenge in generating random numbers this way is, that it is extremely difficult to achieve sufficiently precise timing of two signals, that they arrive at the register at the same time. This is mostly due to significant efforts invested by device manufacturers in reducing setup and hold times and thus preventing metastable states of registers. Device manufacturers use extensive device lifetime studies, where they determine a failure in time (FIT) rate of the device. One FIT corresponds to one failure in 10^9 hours. [10]

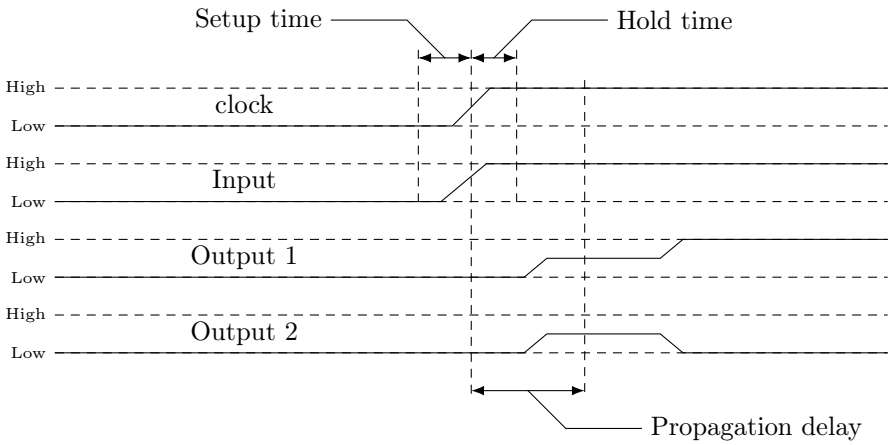


Figure 1.8: Example waveforms of a metastable register

The FIT rating can be further used to calculate the mean time between failures (MTBF) for a specific device and design. MTBF serves as an estimate of time between two system failures due to metastability. It is calculated according to the device and design specifications and its typical order of magnitude is tens of years. [11]

Taking into account typical MTBF, it would take long time and even years to generate one random bit using just the metastability of a circuit as a source of randomness. That's why we consider that the metastability itself cannot be used to generate large quantities of random data.

1.1.2.2 Oscillatory metastability

Another kind of metastability, which occurs in electronic devices is an oscillatory metastability. This kind of metastability, as opposed to a metastable register behavior, does not cause a system to fall into an undefined state, but rather to oscillate between low and high states for undetermined period of time. In [12], it is shown, that oscillatory metastability can be achieved, when an additional delay is introduced to an RS-latch circuit. This circuit is then initialized to an illegal state in order to obtain its metastable behavior.

In [13], the transient effect ring oscillator (TERO), which uses oscillatory metastability to generate randomness, was introduced. It consists of a modified RS-latch, which is periodically set to an illegal state by setting and resetting it at the same time, effectively violating setup and hold times of such a latch. Internal structure of a TERO is depicted in Figure 1.9.

When a *ctrl* signal is asserted, the TERO goes into an oscillatory state, where it stays for a random period of time. After the oscillatory transition, the cell settles to one of logic levels (high or low). This final state is also random. Figure 1.10 shows, that the number of oscillations at the output of the TERO as well as its final state are not constant. As opposed to analog metastability shown in Figure 1.8, here the output oscillates between two defined states.

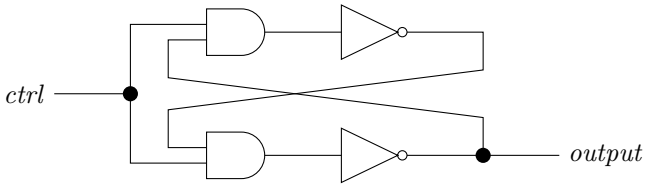


Figure 1.9: Internal structure of a TERO

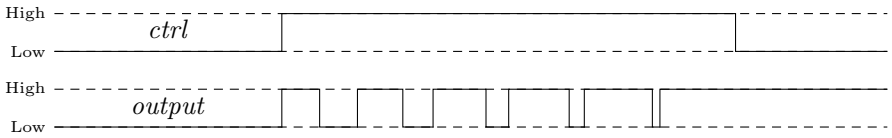


Figure 1.10: Example waveforms of a TERO

1.2 Extraction of randomness from the clock jitter

The clock jitter is considered to be a good source of randomness in digital devices, since it is always present and contains intrinsic random elements. When generating random numbers from the jitter, we need to digitize it in some way in order to produce random bits. The most commonly used method of randomness extraction from the clock jitter is based on sampling the jittered clock edge. Figure 1.11 shows this method of randomness extraction.

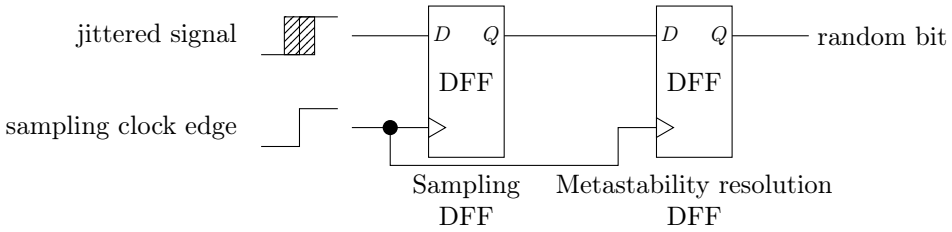


Figure 1.11: Randomness extraction from the jittered clock signal by its sampling on the rising edge of the reference clock signal

In order to produce random bits, the clock signal must arrive during the jitter-affected edge of the jittered signal. This requires substantial precision of clock timing, because the jitter is very small (usually in order of picoseconds or $\frac{1}{1000}$ of the clock period). As we already mentioned, the jitter is inevitable, so even the sampling clock signal is not jitter-free. This also adds to the difficulty of precise timing. Last, but not least, random bits generated using sampling method are prone to be biased. The bias of such bits depends greatly on the duty cycle of the sampled clock. If the duty cycle is 50%, then we have 50% probability that the output bit will be 1. But when the duty cycle is unbalanced, the probability of obtaining 1 is not equal to the probability of obtaining 0 but it is proportional to the duty cycle. Despite the drawbacks of this method, it is still the most often used method to extract randomness from the clock jitter [1, 14–16].

One of the ways to convert random jitter to random bits is to accumulate the jitter until its size is bigger than the sampled signal period [14]. In that case, every sampling of such a signal would produce a completely unpredictable result.

One of the TRNGs using the jitter accumulation is the elementary ring oscillator based TRNG (ERO-TRNG), proposed and modelled in [14]. Its internal structure is depicted in Figure 1.12. It consists of two ring oscillators, a frequency divider, and a D flip-flop.

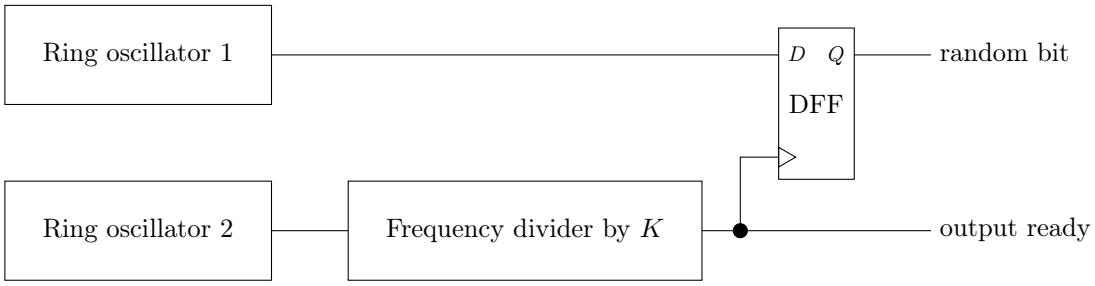


Figure 1.12: Elementary ring oscillator TRNG

A frequency divider allows to set a longer period between two samplings of Oscillator 1, which allows its phase jitter to accumulate. When the K value is sufficiently large, every output bit is completely unpredictable because of the accumulated jitter.

Instead of waiting for a longer time in order to accumulate enough jitter, we can use multiple phase-shifted clocks to reduce the jitter accumulation time. There are two common principles to produce multiple phase-shifted clocks: use multiple independent oscillators as proposed by Sunar *et al.* in [1] or use multiple outputs of one multi-phase oscillator as proposed by Cherkaoui *et al.* in [16].

Multiple oscillators are used in a multi-ring oscillator based TRNG (MURO-TRNG) proposed in [1] and further enhanced in [2] and [17]. Figure 1.13 shows a block diagram of such a TRNG.

The basic principle of this kind of TRNG lies in implementation of multiple clock sources, in this particular case ring oscillators, which produce a set of clocks with uniformly distributed phases. When the number of clock sources m satisfies Condition (1.5), then the total jitter size of all the oscillators will be bigger than the oscillation period. Consequently, we could sample this signal at any time and always sample at least one clock during its jittered edge.

$$m > \frac{T}{\sigma} \tag{1.5}$$

The second method to obtain multiple phase-shifted clock signals is to use multiple outputs of one oscillator. The design proposed in [16] uses a self-timed ring (STR) oscillator. Figure 1.14 shows the block diagram of STR-TRNG, which uses only two oscillators, one of which has multiple outputs. The advantage of an STR is that if configured correctly, it guarantees that

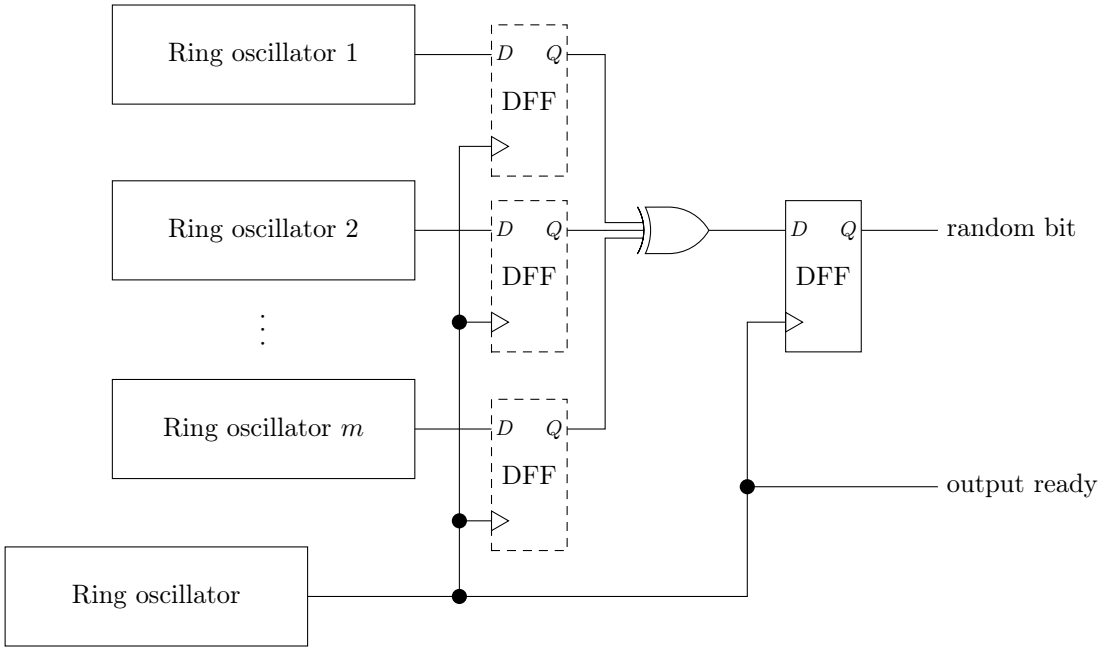


Figure 1.13: Block diagram of multi-ring oscillator based TRNG proposed by Sunar *et al.* [1] and enhanced (dashed DFFs) by Wold *et al.* [2]. Reference clock is generated by a ring oscillator.

output clock phases are equally spaced. This greatly reduces risks of using ring oscillators, where we can only assume uniformly distributed phases, but cannot guarantee them.

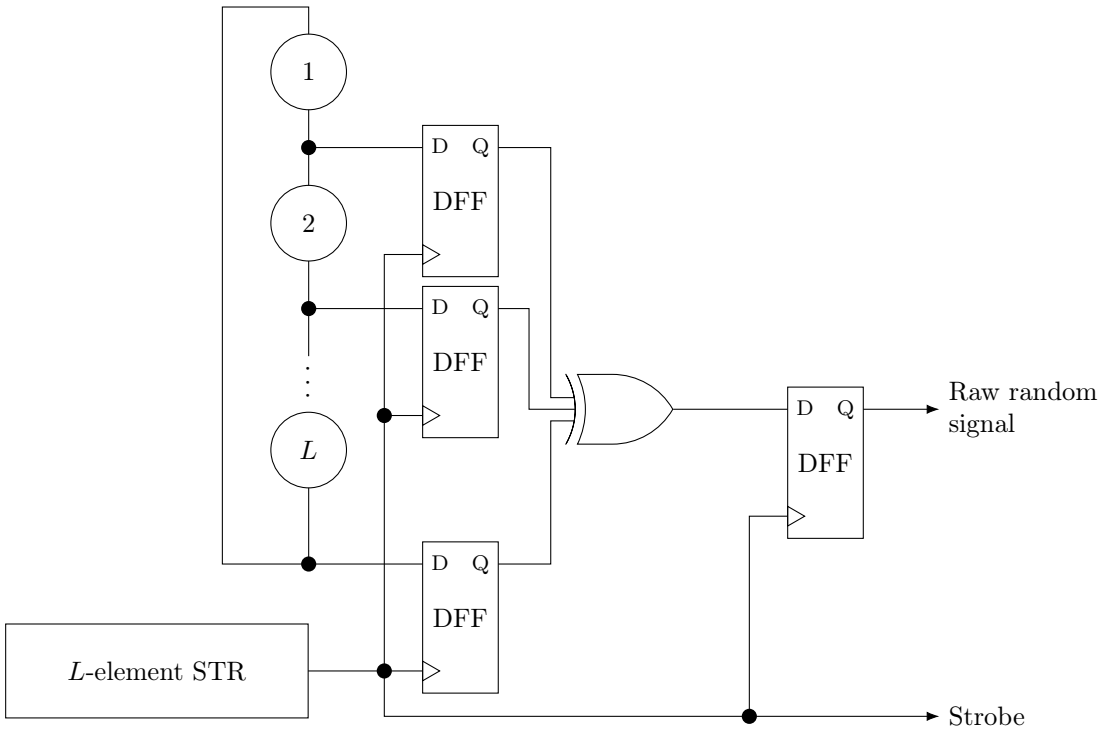


Figure 1.14: Block diagram of STR-TRNG. Reference clock is generated by an L-element STR.

Although both ring oscillators and STRs are prone to locking as shown in [18], the locking detection mechanism is much simpler when using STRs since we need to detect locking between two oscillators whereas in MURO-TRNG we would need to detect locking between each pair of existing rings, which would be very expensive.

Jitter accumulation time can be set using frequency divider as presented in Figure 1.12 or by setting the periods T_1 and T_2 by design so that their difference is smaller than the jitter standard deviation as defined in Eq. (1.6). This method is used in the coherent sampling based TRNG using two ring oscillators (COSO-TRNG) proposed by Kohlbrener and Gaj in [19]. Instead of sampling, this method uses a counter to extract randomness from the clock jitter.

$$\sigma > |T_1 - T_2| \tag{1.6}$$

Coherent sampling, also called subsampling, is a method of sampling which allows to increase sampling precision without increasing the sampling clock's frequency. Conventional sampling reconstructs an image of every period of the sampled signal. Subsampling method reconstructs an image of the sampled signal using multiple periods of the sampled signal to take samples from. In order to do this, two conditions have to be satisfied: the sampled signal must be periodic and the ratio of the sampling and sampled frequencies must be known. If the sampled signal is periodic, we can take its samples from different periods and then still be able to reconstruct the original signal. Sampling this way is generally slower than conventional sampling, but it allows us to use lower sampling frequencies and still achieve high sampling precision.

Phase-locked loops (PLLs) guarantee the rational relation between input and output signal frequencies and hence the subsampling principle is well illustrable on an example of a PLL based TRNG (PLL-TRNG), which also uses subsampling to generate random numbers.

PLL-TRNG was first proposed in [20] and its stochastic model was proposed in [21]. Figure 1.15 shows the PLL-TRNG block diagram.

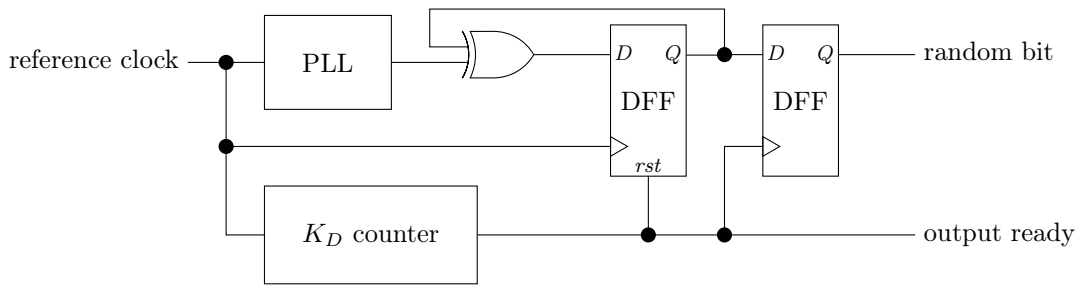


Figure 1.15: PLL-TRNG block diagram

The output signal of the PLL has the frequency given by Eq. (1.7).

$$f_{out} = \frac{K_M}{K_D} \cdot f_{ref}, \tag{1.7}$$

where f_{ref} is the reference (input) frequency of the PLL, f_{out} is the output frequency, and K_M, K_D are multiplication and division factors.

Because of the relation between frequencies of the sampling and sampled signal, the sampler output contains a deterministic pattern. Figure 1.16 shows how the subsampling in PLL-TRNG works.

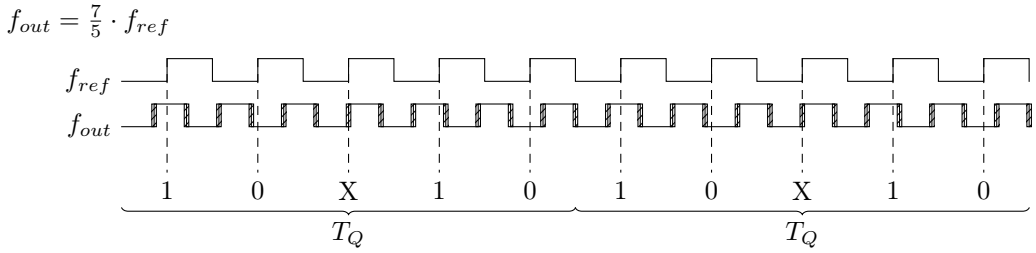


Figure 1.16: PLL-TRNG subsampling principle

In Figure 1.16, we can see the period T_Q, which corresponds to K_D periods of the reference clock signal. The T_Q is the repetition period of the output pattern. We can also notice, that in every T_Q period we should obtain at least one jitter affected sample (marked X in the figure), for which we cannot predict its value.

Since every bit at the input of an XOR operation has an impact on the output value, we XOR all the bits in the T_Q period. The output of the XOR operation is then determined by the random sample(s). This procedure is equivalent to counting the number of ones at the output of the sampler and taking only the LSB of such counter, which is similar to the original design of COSO-TRNG [19].

Consequently, we can generalize that the coherent sampling based randomness extraction removes the deterministic pattern from the generated data. Indeed, this is true in both [20] and [19].

1.3 Models and dedicated tests

A stochastic model of a TRNG specifies a family of probability distributions that contains all the possible distributions of the raw random numbers. Its main objectives are to characterize the probability that an output bit will be equal to one ($P(X = 1)$) and the probability of a vector of bits of certain value at the output ($P(X_1 = x_1, X_2 = x_2, ..., X_n = x_n)$), and from them to estimate the entropy rate at generator output. The model is only practical when the

probabilities it defines are based on some measurable parameters. We will take a closer look at stochastic models through an example of a free running oscillators based elementary TRNG [14]. Authors in [14] propose a model based on the phase of the clock signal generated by a free running oscillator and they define the probability of an output bit being one by Eq. (1.8).

$$P(X=1) = \frac{1}{2} - \frac{2}{\pi} sin(2\pi(\mu t + \varphi(0)))e^{-2\pi^2\sigma^2 t} + O(e^{-4\pi^2\sigma^2 t}) \tag{1.8}$$

From this probability, the lower bound of entropy per bit at the output of the generator can be denoted by Eq. (1.9).

$$H_{min} \approx 1 - \frac{4}{\pi^2 \ln(2)} e^{-4\pi^2 Q} = 1 - \frac{4}{\pi^2 \ln(2)} e^{\frac{-4\pi^2\sigma_{jit}^2 T_2}{T_1^3}} \tag{1.9}$$

We can see that the entropy depends on three measurable parameters: oscillator periods (T_1, T_2) and combined jitter (σ_{jit}). So by measuring these parameters we can estimate the entropy at the generators output. This is important for two reasons: the TRNG can be characterized at design stage and the entropy rate at its output can be monitored during normal operation of generator. Periods of the two oscillators are easy to measure even inside the device. The main difficulty in online entropy estimation is related to the measurement of σ_{jit}^2, knowing that only the contribution of the thermal noise should be taken into account in entropy rate computation. This is done in [22], where authors propose a method of embedded evaluation of randomness (entropy rate per bit) based on measurement of the jitter variance.

We can conclude that if the size of the above mentioned three model parameters can be obtained from the required lower entropy bound, an online measurement of these parameters can be used to verify that the generator does not go below this value. This measurement and comparison with the thresholds obtained from the model constitutes a basis for the dedicated online tests.

1.4 Post-processing

The purpose of the post-processing block is to render the output sequence indistinguishable from the ideal random sequence, which is uncorrelated and uniformly distributed. Two main types of post-processing exist:

Algorithmic post-processing uses some data processing algorithm to enhance statistical parameters of the generated numbers. It may be XOR-ing several output bits [23], Von Neumann correction [24], various types of compression algorithms, etc.

Cryptographic post-processing uses some cryptographically secure algorithm in order to ensure unpredictability of generated numbers in forward and/or backward direction if the

physical source of randomness fails. Using a cryptographic post-processing improves the robustness of the TRNG against attacks.

The post-processing can improve the entropy rate (entropy per bit) at the generator output at the cost of reducing the output bit rate, however it can never generate entropy (see CPG.4 in [25]).

Our aim is to produce high quality raw random numbers so that the post-processing is not needed. Such approach is particularly useful in high security applications.

1.5 Standards for TRNG design and certification

Many different use cases of cryptography require different levels of security, hence there is a need to standardize usage of cryptography for specific applications. Many standards already exist, which provide standard algorithms for encryption (AES [26], RSA [27]) or hash functions (SHA-3 [28]). But because of the nature of TRNGs, which are technology and platform specific, there is no way of providing a standardized design. That is why various certification authorities have developed different standard approaches towards TRNG certification.

The first attempt in TRNG design certification required only testing statistical properties of generated numbers. Many test suites were proposed in order to achieve this, such as FIPS 140-1 [29], DIEHARD [30], NIST 800-22 [31], etc.. The idea behind statistical testing of TRNGs is, that a TRNG should produce an output sequence that is undistinguishable from the ideal one. An ideal random sequence is stationary, uniformly distributed, and its samples are independent. Indeed, statistical properties of generated numbers can be easily tested by suitable statistical tests, but not their independence. The problem with this kind of TRNG testing is, that it treats a TRNG as a black box and considers only its output. Now if we place a good pseudo-random number generator in place of a TRNG, its output will pass all kinds of statistical tests we can throw at it. That's because good pseudo-random number generators are made of algorithms that produce sequences with perfect statistical properties. However, their output is not truly random, it only seems to be random. They are algorithms after all and as such, their behavior is clearly defined and predictable.

Evaluating statistical properties of a TRNG is necessary, but clearly not sufficient. So newer standards exist, which require noise source characterization in addition to statistical testing. Such standards are AIS-20/31 made by German Federal Office for Information Security (BSI) [25] and NIST 800-90B by U.S. National Institute of Standards and Technology [32]. Although the two standards use slightly different evaluation methodology, they both share the same basic idea of a TRNG architecture. Figure 1.17 depicts a block diagram of a TRNG as specified by both AIS-20/31 and NIST 800-90B.

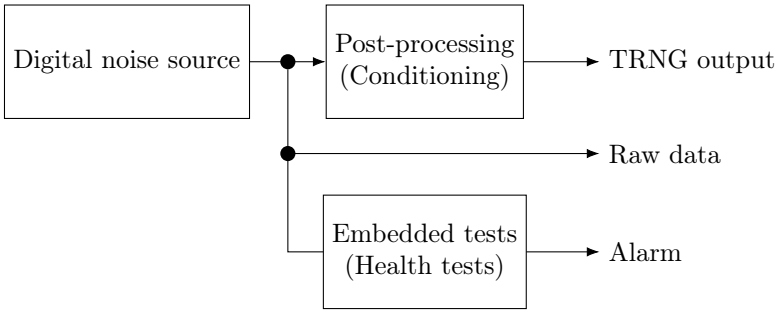


Figure 1.17: TRNG block diagram according to AIS-20/31 and NIST 800-90B (NIST 800-90B terminology in parentheses)

The digital noise source is composed of an analog noise source and a digitizer. This block outputs the digital noise, to which we refer as a raw binary signal. It is also the only block that extracts the true randomness from the underlying process, hence the only block that generates the entropy. Its output must be available for evaluation purposes in order to test the quality of the raw signal and to estimate the entropy rate at the output. Note that the post-processing is an algorithmic process and therefore we can easily compute entropy rate at its output (which is also output of the generator) from the entropy rate at its input (which is known from the model and which can be verified by testing the raw signal data). We underline that according to the AIS-20/31 standard, the post-processing must not reduce entropy.

Both standards consider post-processing, which is described in the previous section, as an option. Ideally, a good TRNG would not require an algorithmic post-processing at all.

The embedded tests are the third required part of the TRNG design. According to AIS-20/31, these tests contain at least two kinds of tests: the total failure test and online tests. The total failure test should very quickly report the complete loss of entropy at the source with a low probability of false alarms. Such a complete loss of the entropy source might be a rapid change or total loss of the physical connection to the source. Online tests, on the other hand, should be able to detect irreparable intolerable defects in the output sequence. The irreparable defect is a statistical flaw, which cannot be corrected by the post-processing. Such a flaw may be produced when the device is operating outside its operating parameter range (*e.g.* over/under voltage, extreme temperatures, etc.). Online tests can be run either continuously, on demand, or triggered by specific internal event. However, tests have to pass successfully each time the TRNG is started/restarted, effectively working also as power-up tests.

1.5.1 Summary of AIS-20/31 requirements on TRNGs

AIS-20/31 standard recognizes several different classes of true random number generators depending on their working principle:

— Three PTG classes for the physical true random number generators – PTRNG (classes PTG.1 to PTG.3),
— One NPTG class for the non-physical true random number generators (class NPTG.1),
— Four DRG classes for the deterministic (pseudo) random generators (classes DRG.1 to DRG.4).

Since we are focusing on the physical TRNGs, we will have a closer look at the PTG classes.

All of the TRNG classes are required to pass the black-box statistical tests. These tests are divided into two test procedures described in AIS-20/31:

— Test procedure A contains statistical tests T0 to T5. These tests verify general statistical properties such as bias and are intended to test post-processed data.
— Test procedure B contains tests T6 to T8. Tests T6 and T7 are intended to detect dependencies between generated numbers. Test T8 compares the estimated Shannon entropy per bit with a threshold of 0.997. Test procedure B is intended to test raw data.

1.5.1.1 PTG.1 low security TRNG class

PTG.1 is a low security physical TRNG class intended for applications that are not security critical. Figure 1.18 shows the block diagram of such generator and test points required by the standard.

Requirements on the source of randomness

No stochastic model is required for a PTG.1 TRNG. The noise source must be, however, physical, clearly defined and described so that it is clear where the randomness originates.

Embedded tests

PTG.1 class requires the implementation of both total failure and online tests. The total failure test must detect a complete failure of randomness source. Online tests should continuously monitor and ensure the statistical quality of produced random numbers. The PTG.1 class requires the online tests to monitor the quality of internal random numbers (i.e. at generator's output).

Post-processing

PTG.1 class does not require the TRNG to use any post-processing. It does not discourage the use of post-processing either. However, a PTG.1 TRNG is required to pass the statistical tests of the Test Procedure A, so the post-processing may be required if the raw binary signal cannot pass said tests.

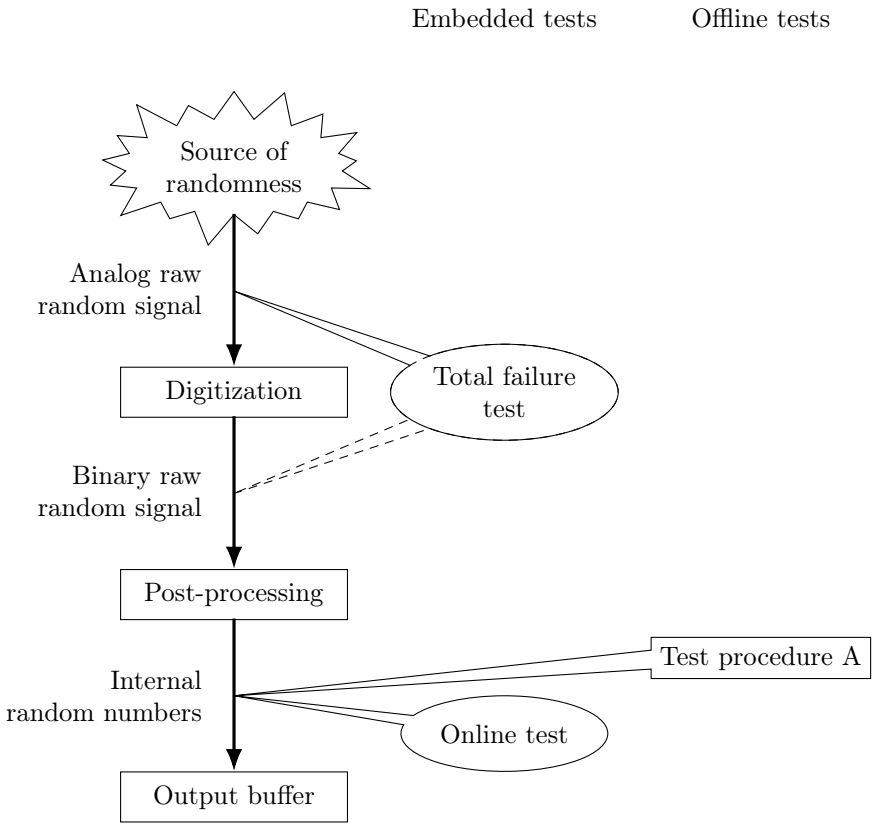


Figure 1.18: PTG.1 TRNG class

1.5.1.2 PTG.2 class

PTG.2 is a physical TRNG class, which may be used to generate cryptographic keys, nonces, seeds for DRNGs, etc. Compared to the low-security PTG.1 class, the PTG.2 TRNG must ensure the secrecy of produced random numbers (their unpredictability). Figure 1.19 shows the internal structure and required tests for the PTG.2 TRNG.

Requirements on the source of randomness

All the requirements of the PTG.1 class apply to the PTG.2 class, too. Additionally, a stochastic model for randomness source is required. The stochastic model must take into account the behavior of the randomness source. Based on parameters of the source, the model estimates the entropy of the raw binary signal. Shannon entropy of the raw binary signal must be above 0.997 per bit according to the AIS-20/31 standard.

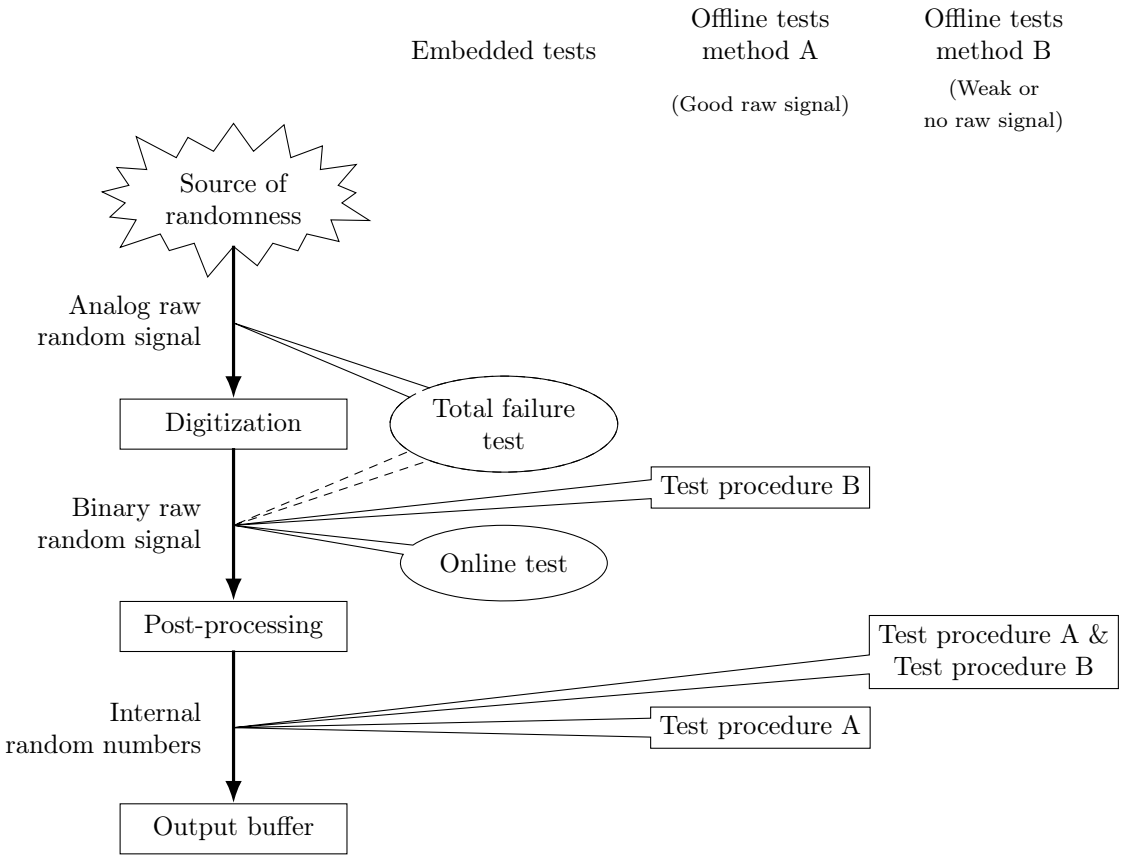


Figure 1.19: PTG.2 TRNG class

Embedded tests

Both, total failure and online tests are required to be implemented for a PTG.2 TRNG. The total failure test must detect a total randomness source failure.

The online tests must detect intolerable statistical weaknesses of the raw binary signal. They must operate on a raw binary signal because the use of post-processing might mask some potentially dangerous defects. The online tests should be tailored to the stochastic model. This way they can detect the defects specific to the used randomness source very efficiently.

Post-processing

Similarly to the PTG.1 class, the post-processing is not required by a PTG.2 class when the raw binary signal provides sufficient quality random numbers. If the post-processing is necessary, the PTG.2 class does not put any restriction on the algorithm used, however, it should not reduce entropy.

1.5.1.3 PTG.3 high-security TRNG class

PTG.3 is a hybrid TRNG class for high-security TRNGs. TRNGs of this class do not rely solely on the security provided by the randomness source, but add a second security anchor in the form of cryptographically secure post-processing. A hybrid TRNG is a TRNG composed of a physical TRNG, which continuously reseeds the deterministic TRNG. The deterministic TRNG in this case serves as a post-processing for the physical TRNG generating entropy. Figure 1.20 shows the block diagram of such a hybrid TRNG.

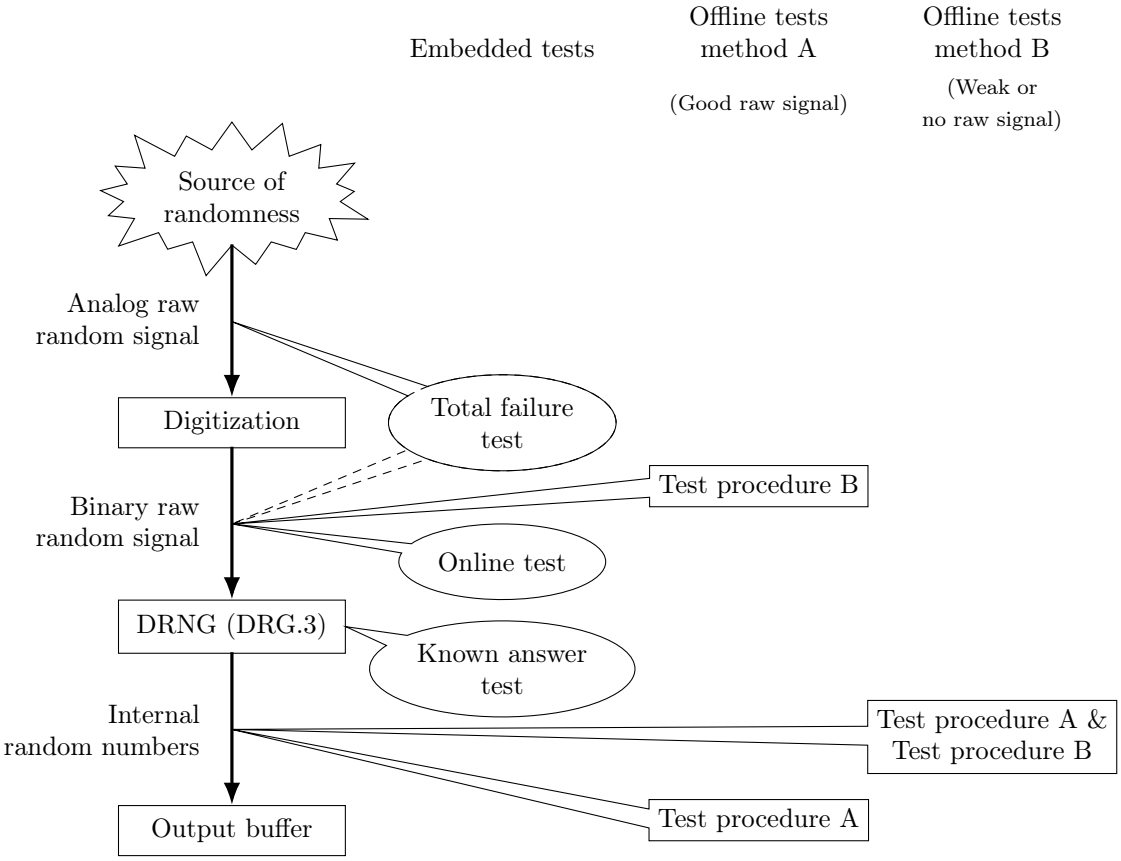


Figure 1.20: PTG.3 TRNG class

Requirements on source of randomness

The randomness source of the PTG.3 TRNG must comply with all the requirements of the PTG.2 class. Shannon entropy of the raw binary signal must be above 0.997 per bit, which must be guaranteed by the stochastic model of the source.

Embedded tests

In addition to total failure and online tests required also for PTG.2 TRNG, the PTG.3 class requires also the known answer test (KAT) for the post-processing. This test must successfully pass every time the TRNG is started/restarted in order to verify the correct functionality of the post-processing algorithm.

Post-processing

In contrast to PTG.1 and PTG.2 requirements, the PTG.3 class requires the use of a (cryptographic) post-processing. Moreover, it requires the use of a DRNG of class DRG.3, which provides forward and enhanced backward secrecy. This effectively means, that the post-processing for a PTG.3 class TRNG must be a cryptographic function. More discussion on the topic of DRNGs is out of scope of this thesis and we kindly refer the interested reader to [25].

1.5.2 Summary of NIST 800-90B requirements on TRNGs

NIST 800-90B requires, similarly to AIS-20/31, that the TRNG output sequence passes black-box tests. These black-box tests are divided into two tracks:

— IID track is used for independent and identically distributed (IID) data.

— Non-IID track is used for data, which fail the IID detection test.

Besides passing black-box tests, NIST 800-90B puts requirements on individual blocks of the TRNG as well.

Noise source

NIST 800-90B has following requirements on noise source:

— Its behavior must be described.

— Its output must be stationary.

— Expected output entropy must be stated.

— It should be protected from adversarial observation and influence.

— The noise source must exhibit random behavior.

Although the standard requires output stationarity and expected entropy statement, it does not require any mathematical proof of these claims. Only technical description of why the noise source is believed to have claimed behavior is necessary.

Health tests

Three types of health tests are required by NIST 800-90B.

Start-up tests should verify whether all the necessary noise source components are functioning properly. No data should be output from the noise source before start-up tests are completed successfully.

Continuous tests check for defects and failures in noise source's behavior. These tests are performed continuously on all the samples output from the noise source. NIST 800-90B requires two approved continuous tests to be used. In addition to approved tests, developer defined tests can be used too. The standard allows developers to not use approved tests, but other continuous tests must be used instead, which can detect the same kind of defects that approved tests do.

On-demand tests are not performed, until asked for. There must be a way of running on-demand tests on the noise source. Samples used for on-demand testing must not be output until the tests are completed successfully.

Conditioning

NIST 800-90B understands conditioning component as a "deterministic function responsible for reducing bias and/or increasing the entropy rate of the resulting output bits" [32]. The conditioning component is completely optional, so it may be omitted whatsoever.

Output entropy of the TRNG is estimated after the conditioning component. The standard provides a list of vetted conditioning algorithms, for which developers can claim full entropy, although this claim has to be validated. These algorithms are hash functions combined with block ciphers (for a complete list, please refer to [32]).

Use of non-vetted conditioning components is permitted, but these components are penalized in terms of entropy estimate. When using a non-vetted conditioning component, the entropy at the output is multiplied by a constant 0.999, effectively banning it from achieving full entropy.

1.5.3 Conclusions of TRNG security certification

There are two major standards concerning TRNGs in effect to this day:

— AIS-20/31 used in many European countries. [25]

— NIST 800-90B used in the U.S. [32]

An international standard is being prepared under the identification ISO/IEC PRF 20543 but it is not released at the time of writing this thesis since it is still in the approval phase.

Both currently existing standards require a TRNG design to be well documented and its inner workings well described in addition to statistical testing. They also require embedded tests to be implemented, so that a TRNG is continuously monitored during its operation.

However, AIS-20/31 goes deeper into the problem of describing the TRNG and requires also stochastic model to be developed. Embedded tests, according to AIS-20/31, also must be

implemented according to the stochastic model. Such a requirement is not enforced by NIST 800-90B, which gives more freedom in the TRNG design, but not in its testing.

All in all, AIS-20/31 is more thorough and its requirements are stricter than those of NIST 800-90B. Since one of the HECTOR project objectives is to propose TRNG designs compliant with AIS-20/31, we will focus only on AIS-20/31 compliant TRNGs in the rest of the thesis. Namely, we will deal only with PTG.2 class TRNGs since PTG.1 does not need a model and is intended for not security critical applications and PTG.3 requires a DRNG, which is out of scope of this thesis.

1.6 Conclusions

Random numbers are required in almost all cryptographic systems nowadays and random number generators (RNGs) are used to obtain them. Four fundamental kinds of RNGs exist depending on the way, in which random numbers are generated:

- Deterministic random number generators (DRNGs), which generate seemingly random sequence of numbers. This sequence is generated by a deterministic algorithm.
- True random number generators (TRNGs), which generate random numbers from unpredictable phenomena. The phenomenon used may be physical (e.g. thermal noise, electromagnetic fluctuations, radioactive decay, etc.) or non-physical (e.g. user input from keyboard and mouse, hard drive read/write operation delay, etc.).
- Hybrid true random number generators (HTRNGs), which use a TRNG as a source of random numbers and then use cryptographic post-processing to further enhance the security and statistical properties of generated numbers.
- Hybrid deterministic random number generators (HDRNGs), which use a TRNG to periodically seed a DRNG, which allows a DRNG to generate a sequence based on real randomness.

Our focus is on TRNGs implemented in logic devices (ASICs and FPGAs) using physical sources of randomness.

Various sources of randomness exist, which could be used to generate random numbers in logic devices. But the most commonly used source is the clock jitter, which we will also focus on in this thesis.

Many TRNG principles exist, but only few are AIS-20/31 compliant, mainly they are difficult or impossible to model. Furthermore, implementation of different TRNGs in different technologies published in the literature gives incomparable results. So our first objective will be to select TRNGs compliant with AIS-20/31 standard and to implement them in the same technologies (FPGAs and ASICs) for a fair comparison and then the selection of the most suitable ones.

Résumé

Les nombres aléatoires sont nécessaires dans presque tous les systèmes cryptographiques d'aujourd'hui et des générateurs de nombres aléatoires (RNGs) sont utilisés pour les obtenir. Il existe quatre types fondamentaux de RNG, en fonction de la manière dont les nombres aléatoires sont générés :

— Les générateurs de nombres aléatoires déterministes (DRNG), qui génèrent une séquence de nombres apparemment aléatoire. Cette séquence est générée par un algorithme déterministe.

— Les générateurs de nombres véritablement aléatoires (TRNG), qui génèrent des nombres aléatoires à partir de phénomènes imprévisibles. Le phénomène utilisé peut être physique (bruit thermique, fluctuations électromagnétiques, décroissance radioactive, etc.) ou non physique (entrée utilisateur à partir du clavier et de la souris, délai de lecture/écriture sur un disque dur, etc.).

— Les générateurs de nombres véritablement aléatoires hybrides (HTRNG), qui utilisent un TRNG comme source de nombres aléatoires, puis utilisent un post-traitement cryptographique pour améliorer davantage les propriétés de sécurité et statistiques des nombres générés.

— Les générateurs de nombres aléatoires déterministes hybrides (HDRNG), qui utilisent un TRNG pour réinitialiser périodiquement un DRNG, ce qui permet à un DRNG de générer une séquence basée sur le vrai aléa.

Nous nous concentrons sur les TRNGs implémentés dans des circuits logiques (ASIC et FPGA) utilisant des sources physiques aléatoires.

Il existe diverses sources d'aléa, qui pourraient être utilisées pour la génération de nombres aléatoires dans des circuits logiques. Mais la source la plus couramment utilisée est le jitter d'horloge, sur lequel nous allons également nous concentrer dans cette thèse.

Il existe de nombreux principes de TRNGs, mais rares sont ceux qui sont conformes à la norme AIS-20/31, parce qu'ils sont difficiles ou impossibles à modéliser. En outre, l'implémentation de différents TRNGs dans des technologies différentes publiées dans la littérature donne des résultats impossibles à comparer. Notre premier objectif sera donc de sélectionner les TRNGs conformes à la norme AIS-20/31 et de les implémenter dans les mêmes technologies (FPGA et ASIC) afin de permettre une comparaison équitable, puis la sélection des plus appropriées.

Chapter 2

Selection and evaluation of TRNGs cores

TRNGs are fundamental building blocks of larger cryptographic systems. Figure 2.1 shows an example of such a system.

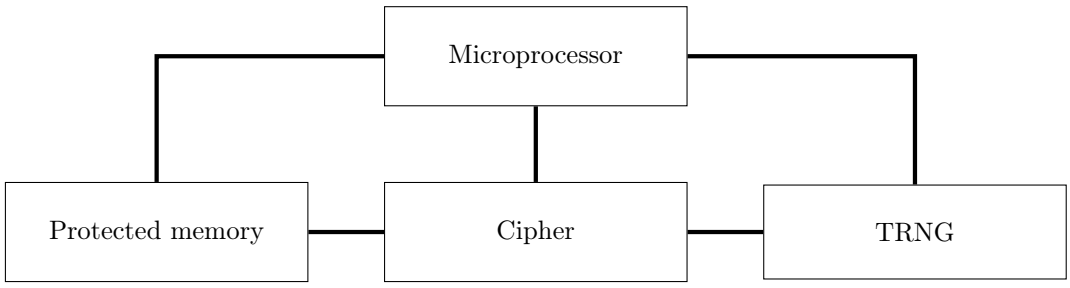


Figure 2.1: Cryptographic system integrating multiple components

A TRNG in a cryptographic system may be used to generate keys, initialization vectors and nonces in cryptographic protocols, masks for side channel attack countermeasures, etc. All these applications have specific requirements on the TRNG (*e.g.* initialization vector generator requires lower bit rate than side channel mask generator does). That is why it is essential to select the right TRNG principle for the application, since no TRNG can fulfill all the requirements.

The main goal of this chapter is to propose a methodology to evaluate TRNGs in order to pick the one that fits specific application needs. We demonstrate said methodology on TRNGs evaluated within the HECTOR project.

2.1 Evaluation methodology

2.1.1 Choice of TRNG cores

First of all, we need to pre-select a range of TRNG cores suitable for the target system. As we established in previous chapters, we will study only AIS-20/31 compliant TRNGs. We will evaluate their suitability for programmable logic devices (FPGAs).

Compliance with AIS-20/31 means that the selected TRNGs must have a clearly defined source of randomness, which is well described. The stochastic model must be feasible and the raw data output must be available for testing. The randomness source should also be quantifiable, which would allow its measurement inside the device. Such a measurement can form a solid base for fast and efficient embedded tests.

In addition to the AIS-20/31 compliance, we want to evaluate only designs that are feasible in any logic device. Since we are looking for a general design, we will avoid features (for example analog components), which are specific only for certain technologies. Because general designs would be technology independent, they should be feasible in FPGAs too. Therefore, in the first step, we decided to implement selected designs in FPGAs, since the design flow is much simpler and faster for this kind of devices. In the next step, we planed to confirm observed behavior of selected TRNGs in ASICs.

Based on general feasibility criteria and compliance requirements with the AIS-20/31, we pre-selected TRNG cores that use oscillating structures as classified in [33]:

- Single-event ring oscillators
  - Elementary ring oscillator based TRNG [14] (ERO-TRNG)
  - Coherent sampling ring oscillator based TRNG [19] (COSO-TRNG)
  - Multi-ring oscillator based TRNG [1] (MURO-TRNG)
- Multi-event ring oscillators with signal collisions
  - Transient effect ring oscillator based TRNG [13] (TERO-TRNG)
- Multi-event ring oscillators without signal collisions
  - Self-timed ring based TRNG [16] (STR-TRNG)
- Phase-locked loops
  - PLL based TRNG [20] (PLL-TRNG)

All the pre-selected TRNGs should be feasible in all recent and future FPGA families since they do not use any family-specific features.

2.1.2 Hardware used for evaluation

TRNGs are very sensitive to noises originating from power supplies, communication interfaces, etc. So in order to fairly compare different TRNGs, our objective was to use the same hardware to implement all of them. That applies for the logic device used as well as the evaluation board and hardware support in general.

We developed the Evariste III development platform [34] to evaluate selected TRNGs. To reduce the deterministic noise created by the communication components, we designed a two board system. One board is used to implement only the TRNG core and the other board implements data acquisition with USB communication. Figure 2.2 shows the evaluation platform in more detail.

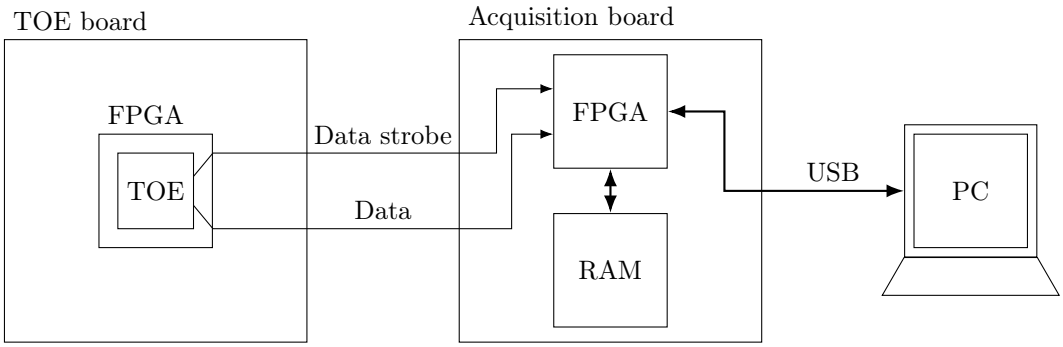


Figure 2.2: Hardware platform used for TRNG evaluation

A target of evaluation (TOE), in our case a TRNG core, is implemented in the TOE FPGA. The TOE board is connected to the acquisition board via a simple serial interface using two LVDS links: one aimed at data transmission and the second one for the control interface. Using LVDS link allows us to place the TOE relatively far from the acquisition system (*i.e.* to a Faraday cage).

The acquisition board features Altera Cyclone III FPGA with a 4 MB RAM memory to store acquired data before they are transmitted to the PC via USB bus. The acquisition memory is needed to guarantee the data continuity at high speeds, which is not possible using direct USB connection.

Both the TOE and the acquisition boards use low-noise linear power supplies in order to reduce the power supply noise to a minimum. To further reduce the impact of global and manipulable noises, we did not use any external clock sources for the TOE and all the necessary clocks were generated inside the TOE.

To evaluate thoroughly all the selected TRNG cores, we tested them on three different FPGA families of three major FPGA vendors – Xilinx, Intel and Microsemi:

— Xilinx Spartan-6: a 45 nm SRAM based FPGA family using 6-input look-up tables (LUTs),

— Intel Cyclone V: a 28 nm SRAM based FPGA family using 5-input LUTs,
— Microsemi SmartFusion 2: a 65 nm flash based FPGA family using 4-input LUTs.

2.1.3 Evaluation criteria

The evaluation criteria always depend on the application, for which the TRNGs are evaluated. According to the requirements of the HECTOR project, TRNGs were evaluated according to the following criteria:

— Area
— Power consumption
— Output bit rate
— Power/Energy efficiency
— Entropy

In the frame of the HECTOR project, we proposed to also evaluate two additional parameters:

— Entropy * bit rate product
— Feasibility and repeatability

2.1.3.1 Area

The area required by FPGA designs is often expressed in vendor-specific units such as adaptive logic modules (ALMs) for Intel FPGAs or slices for Xilinx FPGAs. Every ALM or slice is composed of a number of LUTs and registers as well as other components (*e.g.* internal routes, carry chains, etc.).

Because we wanted to evaluate TRNGs across different FPGA families, we decided to express area in LUTs and registers instead. However, it is important to bear in mind, that different FPGA families implement LUTs of different size: 4-input in SmartFusion 2, 5-input in Cyclone V, 6-input in Spartan-6. Hence, a direct comparison of TRNGs implemented in different families is impossible. It is still, however, the fairest way to compare the area requirements of the same design over different FPGA families since LUTs and registers are the fundamental units of all FPGA architectures.

2.1.3.2 Power consumption and energy efficiency

The power consumption of a TRNG core is minuscule compared to the power consumption of an empty FPGA. At first, we tried the naive approach of measuring the power consumption of an uninitialized FPGA, which we would then subtract from the consumption of an FPGA with TRNG implemented. This approach, however, did not work because the power consumption of an uninitialized device was higher than that of a TRNG.

So we decided to implement a reference project with matching inputs and outputs but without a TRNG core. We would then subtract the consumption of the reference project from the consumption of the FPGA including a TRNG. Figure 2.3 shows this approach.

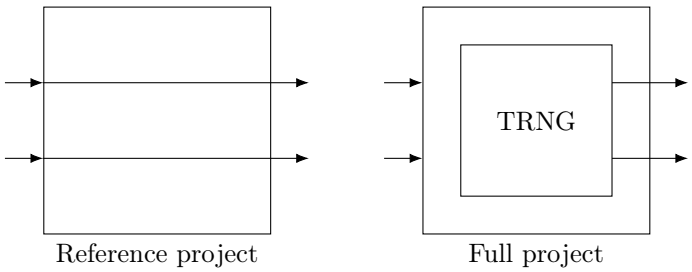


Figure 2.3: Unsuitable power consumption measurement using an empty reference project

In the complete project, the two output signals are the two LVDS links used for data and data strobe signals. The inputs are kept in order to use the same number of inputs and outputs as the reference project does.

The reference project implements only two wires that go straight from inputs to outputs. External inputs are used to prevent the synthesis tool from optimizing the design.

Such an approach did not work well however, because output drivers of the FPGA consumed much more power when they switched rapidly than they do when driving only a constant value. In order to mitigate this issue, we used another method, where we included a multiplexer at the output of the FPGA. Figure 2.4 shows this improved method.

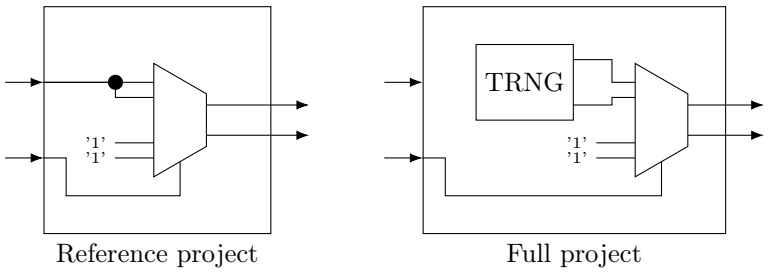


Figure 2.4: Correct power consumption measurement using a multiplexer at the output

This second method effectively mitigates the elevated power consumption of rapidly switching outputs. The multiplexer at the output ensures that during the consumption measurement the outputs stay stable and thus the measured consumption is only the net power consumption of the TRNG.

Power consumption is a very important design constraint, especially for embedded and battery powered devices. For certain applications though, the overall power consumption is not as important as the energy efficiency. Naturally, we want to obtain random numbers at the highest

rate and lowest cost. For this reason, we introduced a measure of energy efficiency, which is defined in Eq. (2.1).

$$Efficiency = \frac{H \cdot R}{P} \quad [\text{Mbits/mWs}], \tag{2.1}$$

where H is the entropy per bit, R is the output bit rate of the TRNG and P is the power consumption. The energy efficiency tells us how much power it takes to generate one Mbit of entropy.

2.1.3.3 Entropy and output bit rate

To estimate the entropy per bit at the output of tested TRNGs, we used the test T8 of the test procedure B of the AIS-20/31 test suite. The output bit rate is easily measured in all the tested TRNGs, since it corresponds to the frequency of the data strobe signal.

These two metrics are very closely related. The output of a TRNG with a low entropy rate can be algorithmically post-processed in order to increase the entropy per bit. This would be desirable in cases when the TRNG cannot fulfil the requirements of the security standard. Post-processing the output would essentially mean data compression, hence the reduction of the output bit rate.

To account for this close relationship, we introduced a new metric: entropy * bit rate product. This metric expresses the potential output bit rate of a TRNG bearing full entropy at the output.

2.1.3.4 Feasibility and repeatability

Basic working principles of TRNGs rely on low-level electronic phenomena. To implement a TRNG featuring a guaranteed and constant entropy rate is usually not an easy task and it requires substantial knowledge of the target technology and design experience, even though a design principle might seem simple enough. For example, many TRNGs will not work properly unless the key components are placed and/or routed manually on the FPGA. Some designs may even exhibit different behavior on different devices.

To express the design difficulty, we introduce a grading system for feasibility and repeatability of a TRNG design. There are six grades, which are awarded based on the following criteria:

5 – Design does not require any manual intervention in order to obtain satisfactory results. The results obtained are consistent through different devices of the same family as well as through different families.

4 – A simple manual setup is required, such as manual placement. Results are repeatable in all devices of the same family.

3 – Design is feasible in all families, but it requires manual optimization. The manual setup is not automatically applicable to different device families, but the design is still repeatable in devices of the same family.

2 – Manual topology optimization, routing, or other complex intervention is necessary. A design must be thoroughly tested and tweaked manually for every family. Once proper balance of parameters is found, the design is repeatable in devices of the same family.

1 – Every individual device, even from the same family, must be optimized and tweaked manually. A satisfactory solution is always possible, even though the design is not repeatable.

0 – Results cannot be guaranteed. Some working configurations may be found but they appear randomly and cannot be repeated in devices of the same family, nor in different families.

2.1.4 Initial measurements

Many TRNG principles share common source of randomness. If we want to compare different principles fairly, we need to characterize the common source of randomness. Since the most of the TRNGs selected for our evaluation use the jitter of ring oscillators as a source of randomness, we measured the period jitter of ring oscillators with various periods implemented in selected FPGA families. We implemented only one single ring in each FPGA device and measured the period jitter (σ_T) using an LVDS output. The jitter was measured using a LeCroy WaveRunner 640ZI oscilloscope (4 GHz bandwidth, 40 GS/s) with a D420 WaveLink 4 GHz differential probe. Figure 2.5 shows the results of the period jitter measurements.

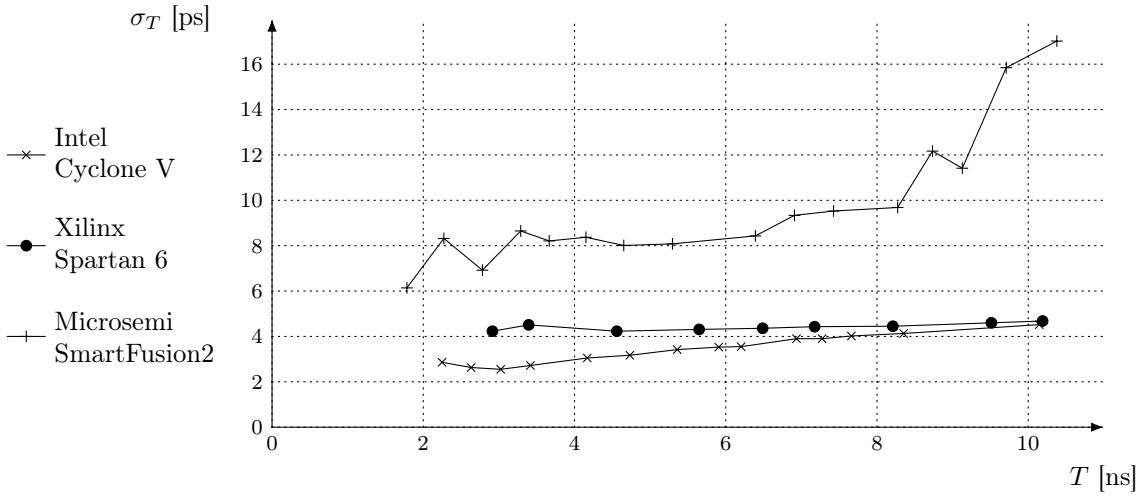


Figure 2.5: Period jitter measured for selected FPGA families

Spartan-6 has the most stable jitter regarding the oscillator frequency. The period jitter stayed around 4 ps for periods between 4 and 6 ns.

Cyclone V has lower or comparable jitter ranging from 3 to 4 ps in the period range of 4 to 6 ns.

SmartFusion2, however, has a stable jitter in two regions. One is for periods between 4 and 6 ns with jitter ranging from 8 to 9 ps. The other is the neighboring region of periods between 7 and 8 ns producing jitter around 10 ps. In any case, the jitter of ring oscillator implemented in a SmartFusion2 device is much higher than the jitter of oscillators implemented in Cyclone V or Spartan-6. This is probably due to some deterministic noise present in this device (*e. g.* the noise coming from the RC oscillator).

For periods below 3 ns, the noise of the measurement equipment starts to dominate over the noise of the ring oscillator. Since only the inherent noise of the electronic device is a suitable source of randomness, we cannot rely on the measurement results below 3 ns.

We will use the results of this characterization to find suitable design parameters for TRNGs selected for evaluation.

2.2 Implementation of selected TRNG cores

In this section, we present each of the selected TRNG cores. We provide a brief overview of the basic principles and discuss particular design challenges of every design.

2.2.1 Elementary ring oscillator based TRNG

The ERO-TRNG was proposed and modeled in [14]. Two identical ring oscillators form the base of the generator. One of them is used to generate a sampling signal, which is then used to sample the output of the other ring oscillator using a D flip-flop (DFF). The frequency of the sampling RO is divided by K in order to obtain a lower frequency of a sampling signal, which would allow the jitter of the sampled RO to accumulate (for more information about jitter accumulation please refer to Section 1.2). Figure 2.1 shows an architecture of the ERO-TRNG as it is implemented in FPGAs.

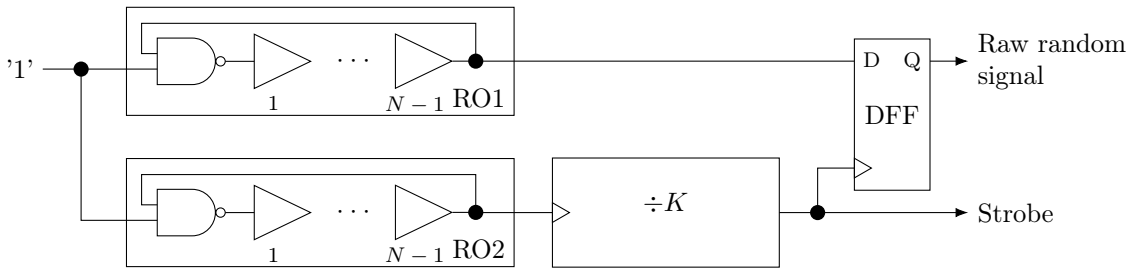


Figure 2.6: Architecture of the elementary ring oscillator based TRNG

We used ring oscillators composed of one NAND gate and $N-1$ non-inverting buffers to construct an N-element ring. The NAND gate can be used to turn the oscillator on or off.

To compare the results obtained on different FPGA families, we chose N such that the oscillating frequency is approximately the same in all the tested families. We obtained oscillating frequencies around 300 MHz using:

— 3 elements (NAND gate with 2 buffers) in Xilinx Spartan-6,

— 5 elements (NAND gate with 4 buffers) in Intel Cyclone V,

— 5 elements (NAND gate with 4 buffers) in Microsemi SmartFusion2

The two fundamental design parameters of the ERO-TRNG are the RO frequency and the reference clock division factor K. The frequency was preselected for the sake of fair comparison.

The second design parameter, the divisor K, determines the jitter accumulation period. The required accumulation period depends on the size of the period jitter and the lower entropy bound. The lower entropy bound is defined by the stochastic model and for the ERO-TRNG it can be calculated using Eq. (2.2) [14].

$$H_{min} = 1 - \frac{4}{\pi^2 ln(2)} e^{\frac{-\pi^2 \sigma_{th}^2 K T_2}{T_1^3}}, \tag{2.2}$$

where σ_{th}^2 is the variance of the jitter due to the thermal noise, K is the reference frequency division factor and T_1, T_2 are oscillating periods of the two ring oscillators.

Since the oscillating period was fixed at about 3 ns for all tested devices, we need only to find the period jitter size of ring oscillators implemented. We used the jitter measurements, shown in Fig. 2.5, to find the period jitter size corresponding to 3 ns period for the three tested FPGA families. We then calculated the division factor K according to Eq. (2.2). The jitter size and corresponding K values are as follows:

— $\sigma_{th} \approx 4$ ps, $K = 80\ 000$ for Spartan-6,

— $\sigma_{th} \approx 3$ ps, $K = 135\ 000$ for Cyclone V,

— $\sigma_{th} \approx 8$ ps, $K = 20\ 000$ for SmartFusion2.

Conclusion: Implementation of the ERO-TRNG is straightforward and results obtained are repeatable without any manual intervention. Manual placement of ROs provides better control of the resulting oscillating frequency, but is not required by the proper TRNG design. Locking the RO placement helps retain the same properties of ROs throughout different projects or different iterations of a project.

The ERO-TRNG provides relatively low output bit rate, because the K has to be relatively high in order to guarantee sufficient entropy. Once set properly though, this TRNG offers high security thanks to the solid stochastic model. The embedded tests need to check only that the ROs are oscillating and that they are not locked [17].

The power consumption of the ring oscillator does not depend on its size (number of elements), because only one event propagates over the ring. So only one element of the ring oscillator is active (changing its state) at a time regardless of the oscillator frequency. The power consumption of the whole TRNG depends on frequencies of the used ring oscillators.

2.2.2 Coherent sampling based TRNG using ring oscillators

The COSO-TRNG was first proposed in [19]. It uses two identically implemented ROs as a source of randomness. Figure 2.7 shows the internal structure of the COSO-TRNG.

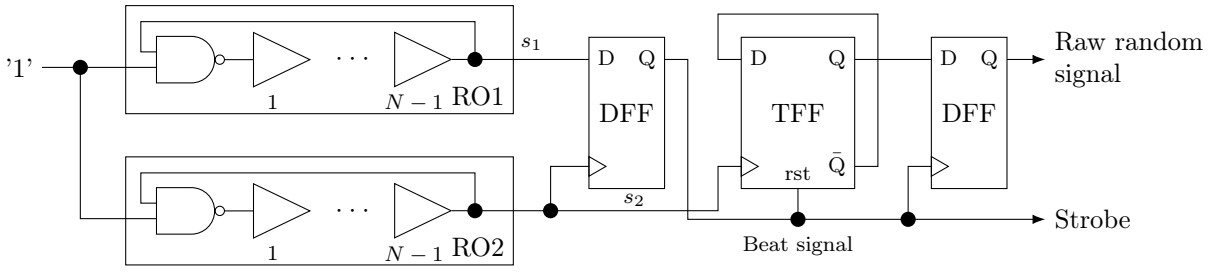


Figure 2.7: Architecture of the coherent sampling ring oscillator based TRNG

Even if the two ring oscillators feature exactly the same internal structure, their frequencies vary a little. This small variation causes a phase difference at the outputs of ROs. By sampling the output of one RO using a DFF clocked by the output of the other RO, we obtain a signal with a variable period, which corresponds to the relative phase shift of the two ROs. This signal is called a *beat signal*.

The second flip flop is a T-flip flop (TFF), which corresponds to a one-bit counter that counts the number of rising edges of the signal s_2 during one half-period of the beat signal. This last bit of the counter is registered in the last DFF and sent to the output of the TRNG as a random bit.

COSO-TRNG extracts randomness from the jitter only if the condition in Eq. (2.3) is met.

$$\begin{aligned} \Delta_T < \quad & \Delta_{T_{max}} \quad , \\ & \Delta_{T_{max}} \quad = \sqrt[3]{\sigma_T^2 \cdot T} \end{aligned} \tag{2.3}$$

Unfortunately, this condition is very hard to fulfill, because the two ROs must oscillate at very close frequencies while avoiding locking to each other, which is difficult to achieve even in ASICs, where one can have complete control over the placement and routing of all components. In an FPGA, satisfying this condition is even harder since we do not have precise control over placement and routing of the rings, it is very difficult if not impossible to implement two rings oscillating at sufficiently close but not identical frequencies. To mitigate this issue, we measured

the clock period T and the period jitter size σ_T while trying different placement and routing options. The placement of the rings can be locked in an FPGA, but many modern FPGAs (e.g. Intel Cyclone V and Microcemi SmartFusion2) do not support manual routing. So every time a project is compiled the compiler routes the elements differently. We placed one RO manually at a fixed place and we used a Tcl script to move the other RO (to change its placement constraints for every recompilation) automatically until a suitable configuration was found. We eventually found following configurations:

— $N = 8$ yielding $T = 6.92$ ns and $\sigma_T \approx 4$ ps in Spartan-6 ($\Delta_{T_{max}} \approx 50$ ps),
— $N = 6$ yielding $T = 3.17$ ns and $\sigma_T \approx 2.5$ ps in Cyclone V ($\Delta_{T_{max}} \approx 30$ ps),
— $N = 10$ yielding $T = 5.4$ ns and $\sigma_T \approx 8$ ps in SmartFusion2 ($\Delta_{T_{max}} \approx 70$ ps),

Conclusion: Due to the extreme sensitivity to Δ_T, the design working on one FPGA is not directly transferable to another FPGA even of the same family because even the slightest change caused by the manufacturing process variation can cause the period difference to swing above $\Delta_{T_{max}}$. So the design must be manually placed and routed for every individual FPGA, which makes it very impractical.

However, COSO-TRNG can provide relatively high output bit rate with low area footprint. Additionally, the design is more suitable for ASIC implementation, where the TRNG can be placed and routed manually and then used as a hard macro.

The power consumption of the COSO-TRNG is very small, because the power drawn by ring oscillators is independent of their size and COSO-TRNG features only three flip flops that increase its power consumption.

2.2.3 Multi-ring oscillator based TRNG

We discussed MURO-TRNG and its working principle in Section 1.2. To summarize, the MURO-TRNG uses multiple ring oscillators, which are supposed to be independent have the same mean frequency and uniformly distributed phases. To reliably extract randomness from a group of ring oscillators, the number of ring oscillators must satisfy the condition in Eq. (1.5), which specifies the relation between the mean oscillating period of all oscillators, their jitter and the number of oscillators used.

The TRNG principle will work only if the phases of ring oscillators are uniformly distributed. However, ring oscillators may lock to each other, in which case the distribution of phases will not be uniform. What is more, the probability of locking is high given the high number of rings required for getting high entropy in MURO-TRNG.

In our implementation, rings were constructed using one NAND gate and three buffers, which produced frequencies ranging from 200 to 350 MHz. We dimensioned the design to Cyclone V

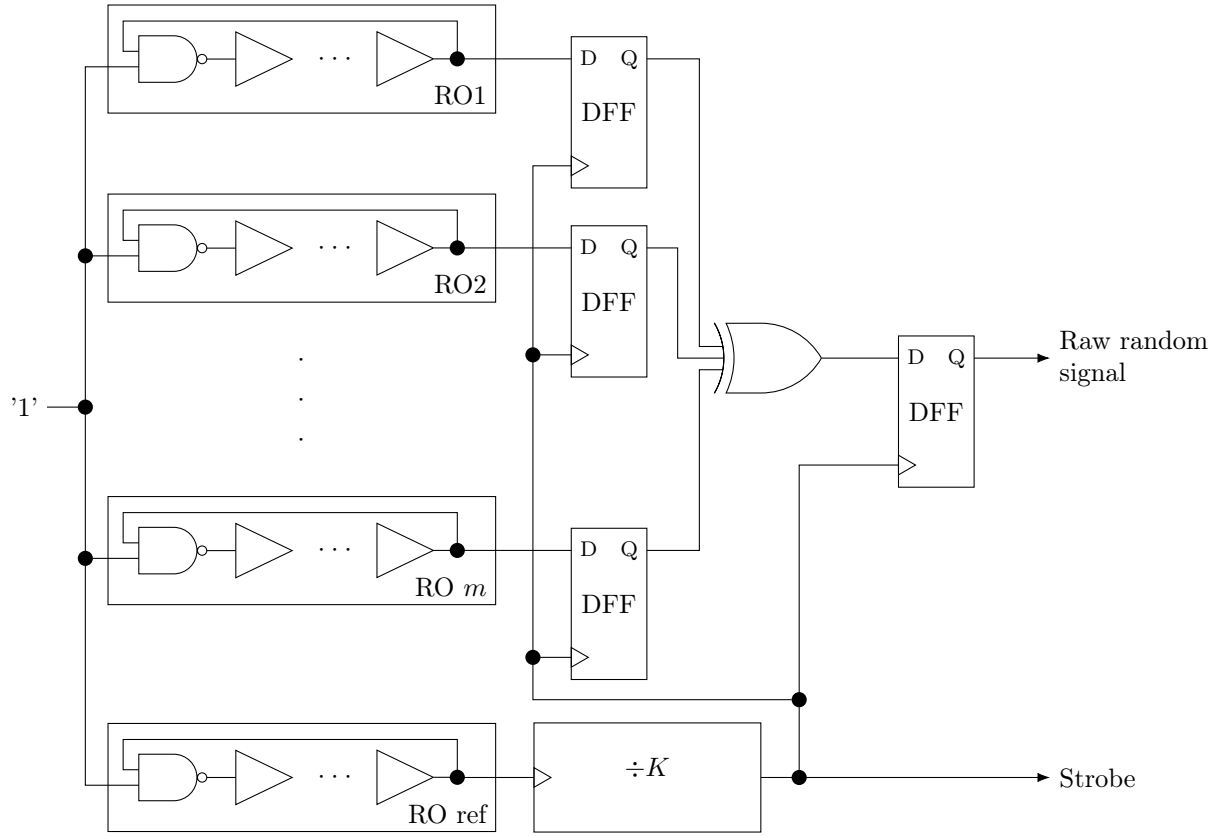


Figure 2.8: Architecture of the implemented MURO-TRNG

FPGA family, which produces the smallest jitter of the three selected families. This way, the generator satisfied Eq. (1.5) for all selected FPGA families.

In Cyclone V, the period jitter size was approximately $\sigma_T \approx 3$ ps. According to Eq. (1.5), the number of rings needed to be $m > 1200$.

To reduce the number of rings, we added a frequency divisor, which allows us to accumulate the jitter for $K = 100$ periods of the reference clock. This effectively increases the accumulated jitter to $\sigma_{acc} \approx 30$ ps and allows us to use $m = 120$ ring oscillators at the expense of smaller output bit rate. Figure 2.8 shows the architecture of the implemented TRNG.

Conclusion: MURO-TRNG does not require any manual placement or routing and its output bit rate as well as entropy rate are very high when sufficient number of ROs is used. Comparing to Sunar et al. [1], who underestimated the jitter, we use comparable number of ROs, but we achieve smaller bit rate. The power consumption of this TRNG is considerable because of the high number of oscillators.

2.2.4 Transient effect ring oscillator based TRNG

The TERO-TRNG generates random bits using oscillatory metastability, which we covered in Section 1.1.2.2. Figure 2.9 shows the TERO-TRNG implementation used for evaluation.

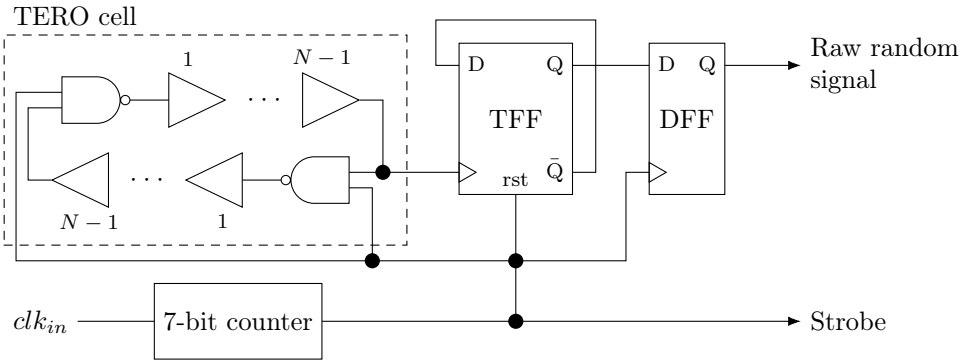


Figure 2.9: Architecture of the TERO-TRNG

The TFF represents the last bit of the counter, which counts the number of oscillations of the TERO cell. Due to the oscillatory metastability of the TERO, the number of oscillations is random. To produce a stream of random bits, the TERO must be restarted periodically. We used a ring oscillator oscillating at approximately 150 MHz to generate clk_{in}. The clk_{in} signal was then divided by a 7-bit counter to generate a control signal, which restarts the TERO cell.

The TERO cell was composed of $N = 11$ elements (one NAND gate and 10 buffers). Such a configuration produced signals with frequencies of approximately:

— 150 MHz in Spartan-6,

— 150 MHz in Cyclone V,

— 90 MHz in SmartFusion2.

To achieve sufficient entropy at the output of the TERO-TRNG, the number of oscillations of the TERO must be within the limits specified by Eq. (2.4).

$$100 < M < \frac{T_{meas}}{T_{osc}}, \tag{2.4}$$

where M is the number of oscillations, T_{meas} is the measurement time and T_{osc} is the period of the output signal of the TERO.

The lower limit for the number of oscillations guarantees that there will be enough oscillations to extract randomness from. The upper limit, on the other hand, prevents cases when the oscillations do not stop before the measurement is restarted. Reliably satisfying this condition for multiple devices is difficult because the TERO cell behaves differently in every device (even within one family) even when the same configuration is used.

Conclusion: The TERO cell itself does not require a big area and there are only a few flip flops needed for the counters and the TRNG core, so the overall area footprint is relatively small.

The power drawn by the TERO does not depend on the number of elements of the TERO but the power consumption of flip flops can increase with increasing clock frequency.

The TERO-TRNG requires manual placement and routing and the design is not repeatable even on devices of the same family.

2.2.5 Self-timed ring based TRNG

A self-timed ring is a multi-event oscillator without signal collisions. Figure 2.10 shows the TRNG implemented for evaluation.

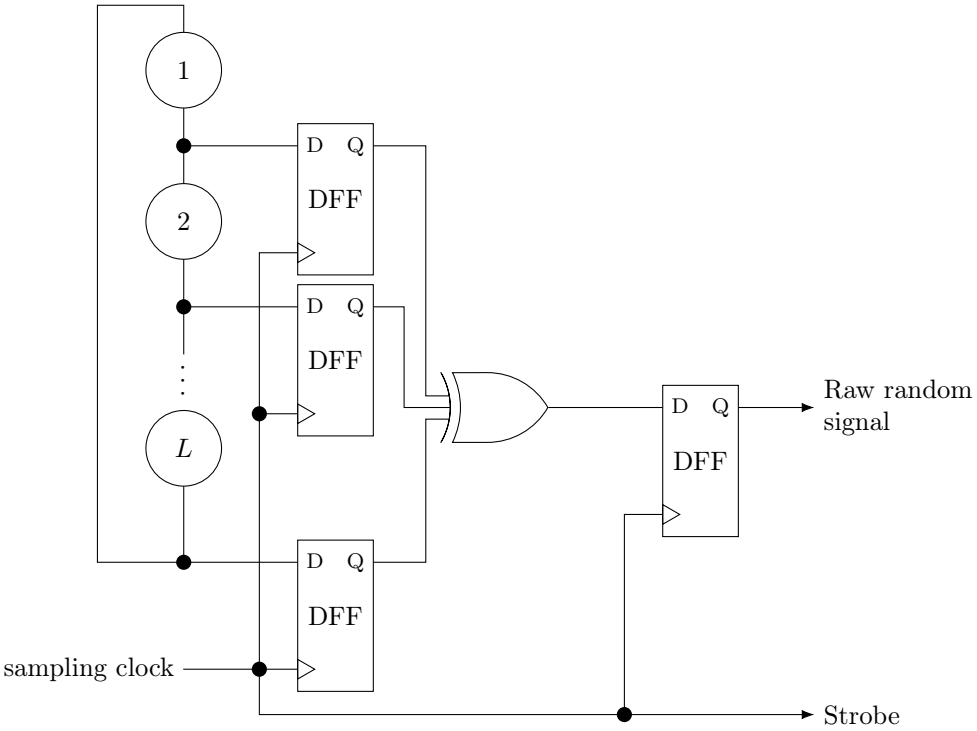


Figure 2.10: Architecture of the STR-TRNG implemented for evaluation

An STR is composed of L Muller cells (C-elements) [16]. Multiple events can propagate across an STR without collisions. Due to temporal properties of STRs, the events can propagate in the following two modes:

— Burst mode – events form a cluster, where the distance between the events is minimal. In this mode, we observe a burst of events with high frequency and then no event for the rest of the ring period.

— Evenly spaced mode – events are evenly spaced within the ring, which produces a periodic signal with a 50% duty cycle.

The principle of randomness extraction of an STR-TRNG is the same as that of MURO-TRNG: the use of a set of clock signals featuring uniformly distributed phases (see Section 1.2). But while in many ROs the phases are distributed statistically, in STR the uniformity of distribution is guaranteed by the principle. In [35], it is shown that when the number of C-elements L and number of events in the STR are coprime, the STR can produce as many equidistant phases as the number of C-elements. In such a case, the phase resolution (phase difference between two neighboring C-elements) can be expressed by Eq. (2.5).

$$\Delta_\varphi = \frac{E}{2L}, \tag{2.5}$$

where E is the number of events in the ring and L is the number of C-elements. The fundamental goal of using multiple phases in a TRNG is to always have at least one signal in transition, which allows to sample the jittered edge of the signal and extract randomness from it. If the jitter is bigger than the phase distance between two signals, the jittered signal can be sampled at any moment. This is expressed by Eq. (2.6).

$$\Delta_\varphi < \sigma_{acc}, \tag{2.6}$$

where σ_{acc} is the size of the jitter accumulated during one period of sampling signal.

The frequency of the sampling clock defines the output bit rate of the TRNG. To maximize the bit rate without jeopardizing the security, we generated the sampling clock at a maximum frequency satisfying the condition of Eq. (2.6). The sampling clock was generated by a ring oscillator inside the FPGA.

Contrary to RO, the frequency of the STR does not depend on the number of C-elements L, but on the number of events. In our case, the STR configured for an evenly spaced mode oscillated at approximately 300 MHz. According to Fig. 2.5, the smallest jitter at this frequency is $\sigma_T \approx$3 ps in Cyclone V FPGA. With such a jitter, we would need an STR of size $L > 550$ according to Equations (2.5) and (2.6). An STR of this size would occupy huge area, so we decided to decrease the sampling clock and use an STR of size $L = 255$.

Conclusion: The STR-TRNG consumes a lot of power because there are many events propagating through the ring. In the evenly spaced mode, an STR oscillates at its maximum frequency. The output bit rate of the STR-TRNG is very high, which also contributes to its high power consumption. The STR-TRNG requires a large area and in order to ensure that it is working in evenly spaced mode, manual placement of STR elements is required.

2.2.6 Phase-locked loop based TRNG

The PLL-TRNG uses subsampling principle to extract randomness from the tracking jitter of the PLL [20]. We covered basics of PLL-TRNG in Section 1.2.

The PLL-TRNG we implemented used two PLLs. Figure 2.11 shows the schematic diagram of a two PLL-TRNG.

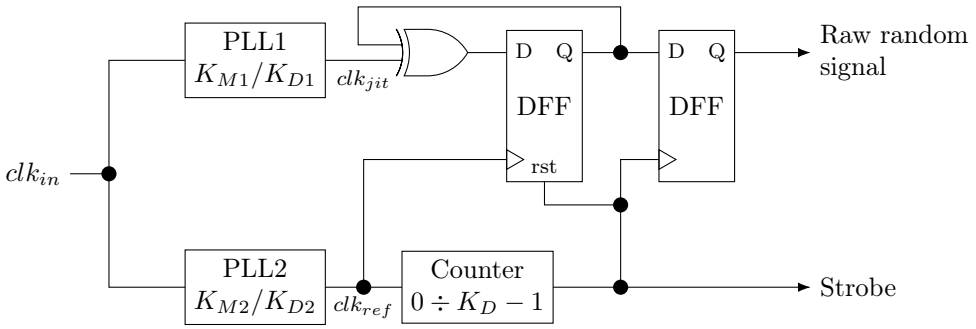


Figure 2.11: Architecture of a PLL-TRNG using two PLLs

The TRNG is based on a subsampling principle, which requires the two frequencies (f_{ref} and f_{jit}) to be mutually related. Given that $f_{jit} = f_{in} \cdot \frac{K_{M1}}{K_{D1}}$ and $f_{ref} = f_{in} \cdot \frac{K_{M2}}{K_{D2}}$, the relation between f_{jit} and f_{ref} is as described by Eq. (2.7).

$$f_{jit} = f_{ref} \cdot \frac{K_{M1}}{K_{D1}} \cdot \frac{K_{M2}}{K_{D2}} = f_{ref} \cdot \frac{K_M}{K_D} \tag{2.7}$$

Two fundamental parameters of a PLL-TRNG design are output bit rate R and sensitivity to jitter S. Both can be calculated from PLL parameters:

$$R = \frac{f_{ref}}{K_D} \tag{2.8}$$

$$S = \Delta^{-1} = \frac{K_D}{T_{jit}} \tag{2.9}$$

The bit rate (Eq. (2.8)) corresponds to the frequency of the strobe signal from Fig. 2.11.

The sensitivity to jitter (Eq. (2.9)) corresponds to the inverse value of the distance between samples Δ. $\Delta = \frac{T_{jit}}{K_D}$ is the subsampling precision, which is the effective distance between samples. To produce random bits with high entropy, the samples must be affected by the jitter, hence satisfy the condition in Eq. (2.10).

$$\Delta \ll \sigma_r, \tag{2.10}$$

where σ_r is the relative jitter between clk_{ref} and clk_{jit}. The security of the PLL-TRNG depends on S. And because we cannot do compromises when it comes to security, the first

and foremost design goal is to keep S within the range specified by Eq. (2.10). The secondary design goal is then to increase the output bit rate R. Equations (2.8) and (2.9) make it clear that there is always a trade-off between R and S because they both depend on K_D. There is still space left though, since the bit rate depends on f_{ref} (PLL2) and jitter sensitivity depends on $T_{jit} = 1/f_{jit}$ (PLL1). Tweaking multiplication and division factors of both PLLs, we found suitable configurations for the three used FPGA families. We used a ring oscillator oscillating at approximately 200 MHz to generate f_{in}. Table 2.1 summarizes the configurations used for evaluation.

FPGA	**PLL1**		**PLL2**		**Total**		Δ
	K_{M1}	K_{D1}	K_{M2}	K_{D2}	K_M	K_D	[ps]
Spartan-6	37	17	17	7	1377	259	4.82
Cyclone V	31	29	23	18	667	558	4.25
SmartFusion2	74	162	18	22	729	407	9.10

Table 2.1: PLL parameters and corresponding distance between samples (Δ) for selected FPGA families

Conclusion: The PLL-TRNG does not require any manual placement or routing. It provides high security and is not affected by global deterministic (*i.e.* data dependent) noise since PLLs are physically isolated from the rest of the FPGA. The output bit rate of the PLL-TRNG is also considerable.

The design of PLL-TRNG is simple, repeatable and it can be automated. However, the choice of PLL parameters is not trivial. Many constraints have to be respected including physical constraints of the PLL manufacturer (*e.g.* maximum K_M and K_D, maximal frequencies, etc.) and security constraints of the TRNG (Eq. (2.10)).

The area footprint of the PLL-TRNG is relatively small if we exclude PLLs, which do not occupy FPGA logic. PLLs themselves are not cheap to implement, because they require considerable silicon area. In FPGAs, hoverer, the PLLs are already provided, hence their use does not cost anything. What is more, most of the FPGA families provide several PLLs, which further reduces the cost of the PLL-TRNG implementation.

Power consumption of the PLL-TRNG depends on the PLLs provided in the particular FPGA. Some of the families have all PLLs turned on by default, even if they are not used (*e.g.* Intel FPGAs). In such a family, PLL-TRNG does not consume much more power than an empty FPGA does, because in either case PLLs are running. The power consumption is considerable in other families, which have PLLs powered down by default (*e.g.* Microsemi FPGAs). A PLL-TRNG implemented in such a family will draw considerably more power than an empty FPGA does.

2.3 Implementation results and their evaluation

Table 2.2 summarizes the implementation results of all selected TRNGs in the three selected FPGA families. For each category, best performing TRNGs are marked **bold** and worst performing ones are marked *italic*. TRNG implementations presented here are not optimized. Optimization of the TRNG design must be done specifically for the target application. Emphasis of this evaluation is to present the evaluation methodology in general terms without any particular application in mind.

FPGA	**Area** (LUTs/ Registers)	**Power cons.** [mW]	**Bit rate** [Mbits/s]	**Power efficiency** [Mbits/mWs]	**Entropy** per bit	**Entropy * Bit rate**	**Feas. & Rep.**
ERO-TRNG							
Spartan-6	46/19	2.16	*0.0042*	*0.002*	0.999	*0.004*	
Cyclone V	34/20	3.24	*0.0027*	*0.001*	0.990	*0.003*	**5**
SmartFusion2	45/19	4	*0.014*	*0.003*	0.980	*0.013*	
COSO-TRNG							
Spartan-6	**18/3**	**1.22**	0.54	0.442	0.999	0.539	
Cyclone V	**13/3**	**0.9**	1.44	**1.598**	0.999	1.438	*1*
SmartFusion2	**23/3**	**1.94**	0.328	0.169	0.999	0.327	
MURO-TRNG							
Spartan-6	*521/131*	*54.72*	2.57	0.046	0.999	2.567	
Cyclone V	*525/130*	*34.93*	2.2	0.062	0.999	2.197	4
SmartFusion2	*545/130*	*66.41*	3.62	0.054	0.999	3.616	
TERO-TRNG							
Spartan-6	39/12	3.312	0.625	0.188	0.999	0.624	
Cyclone V	46/12	9.36	1	0.105	0.987	0.985	*1*
SmartFusion2	46/12	1.23	1	0.812	0.999	0.999	
STR-TRNG							
Spartan-6	*346/256*	*65.9*	**154**	**2.339**	0.998	**154.121**	
Cyclone V	*352/256*	*49.4*	**245**	**4.955**	0.999	**244.755**	2
SmartFusion2	*350/256*	*82.52*	**188**	**2.285**	0.999	**188.522**	
PLL-TRNG							
Spartan-6	34/14	10.6	0.44	0.041	0.981	0.431	
Cyclone V	24/14	23	0.6	0.026	0.986	0.592	3
SmartFusion2	30/15	19.7	0.37	0.017	0.921	0.340	

Table 2.2: Implementation results of selected TRNGs

The most important message passed by the Table 2.2 is that there is no TRNG, which excels in all the evaluated parameters neither there is a generator, which is the worst in all regards. These results confirm that no TRNG can satisfy the needs of every application, hence no vetted TRNG can be proposed. Many design parameters depend on each other, which creates trade-offs

in a design and a compromise must be made in favor of preferred parameters.

It is difficult to tell which design differences are important and which are negligible just from the raw numbers. For this reason, we prepared a scoring system, where a score between 0 and 5 is assigned to each TRNG for every category based on the TRNG's performance. A score of 5 is the best and 0 is the worst. Table 2.3 shows the parameter intervals for every score point.

Score	**Area** (LUTs+ Registers)	**Power cons.** [mW]	**Bit rate** [Mbits/s]	**Power efficiency** [Mbits/mWs]	**Entropy** per bit	**Entropy * Bit rate**
5	< 20	< 0.01	≥ 100	≥ 9970	[0.997,1)	≥ 100
4	[20,100)	[0.01,0.1)	[10,100)	[91.8,9970)	[0.918,0.997)	[10,100)
3	[100,200)	[0.1,1)	[1,10)	[0.57,91.8)	[0.570,0.918)	[1,10)
2	[200,500)	[1,10)	[0.1,1)	[0.00125,0.57)	[0.125,0.570)	[0.1,1)
1	[500,1000)	[10,100)	[0.01,0.1)	[0.003,0.00125)	$[3 \cdot 10^{-7}$,0.57)	[0.01,0.1)
0	≥ 1000	≥ 100	< 0.01	< 0.003	$< 3 \cdot 10^{-7}$	< 0.01

Table 2.3: Scoring system for TRNG comparison

Using this scoring system, we graded all of the evaluated TRNGs. The purpose of scoring is to clearly point out the relative strong and weak points of all devices and to allow a quick comparison of different TRNG cores. From Table 2.2, we can see that even though there are differences between different FPGA families, the relative differences between TRNG cores stay almost the same. For example, the TRNG core with lower power consumption exhibits lower power consumption regardless of the FPGA family. So for simplicity, we decided to use average parameter values from the three families to grade TRNG cores for comparison. Figure 2.12 shows the visual comparison of all TRNGs graded according to Table 2.3.

The graphs point out strong and weak points of every design. It is always possible to optimize a design in favor of certain parameters, which are more important for the application.

The overall area of the graph does not change much though. Every time one of the criteria gets better, some others fall back. Some of the common optimization techniques include:

— *Increase the entropy by reducing the bit rate.* It is possible to use an algorithmic post-processing, which will compress data. The compression may increase the entropy per bit at the expense of lower bit rate.

— *Increase bit rate by increasing area.* We can add more cells, oscillators or whole TRNG cores in order to increase the output bit rate. Doing so will increase the area of the design proportionally.

— *Decrease the area, increase bit rate, and/or decrease power consumption and increase ef-*

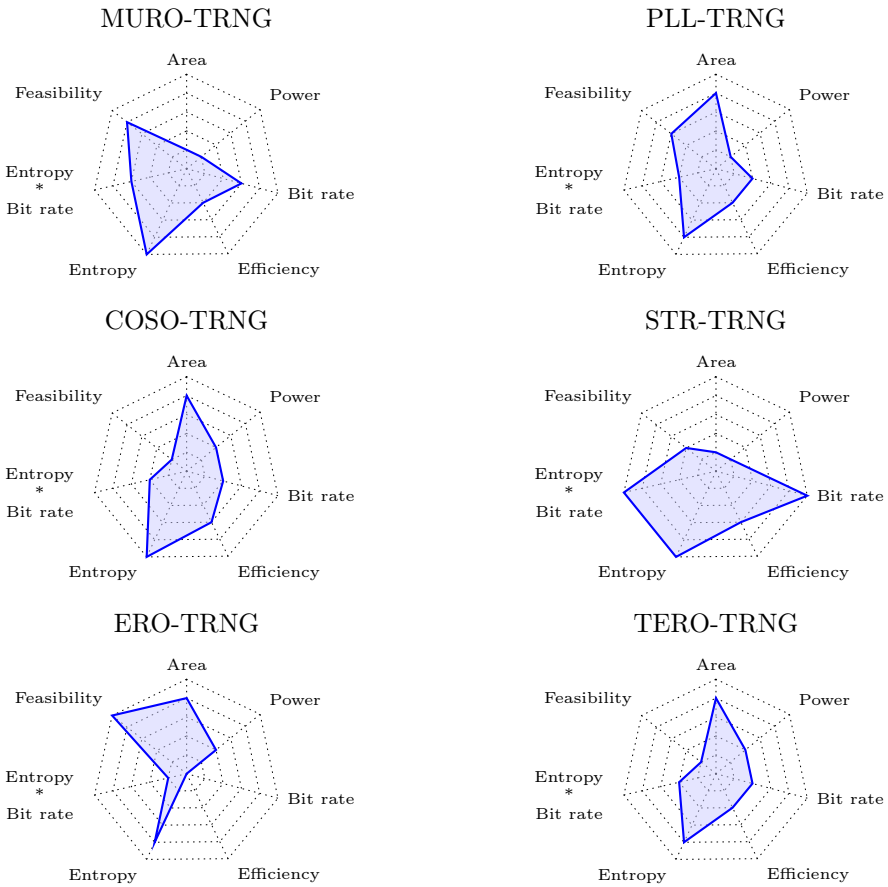


Figure 2.12: Visual comparison of evaluated TRNG cores

ficiency all by reducing the feasibility. Using special properties of a given FPGA family or specific to a particular technology, we can enhance the overall design. At the same time, the technology specific properties will prevent the design to be easily ported to other technologies and may limit its repeatability.

2.4 Conclusion

Embedded systems designers are required to implement security features in different kinds of systems. A TRNG is a root of trust in an embedded security, hence the design of a secure TRNG is imperative for the security of the whole system. Consequently, design of a secure TRNG requires substantial expertise. However, embedded systems designers do not have much of expertise in the security domain since the need of security in embedded systems is recent. In this chapter, we proposed a methodology for fair evaluation of different TRNG cores in order to provide a clearly defined way to choose secure TRNG designs, evaluate them according to application specific criteria, choose the best candidate for the target application and optimize it

to application's needs.

We demonstrated the proposed methodology on an example of TRNG evaluation, which was done in the framework of the European research project HECTOR. In the presented evaluation, we evaluated six TRNG cores:

— Elementary ring oscillator based TRNG (ERO-TRNG)
— Coherent sampling ring oscillator based TRNG (COSO-TRNG)
— Multi-ring oscillator based TRNG (MURO-TRNG)
— Transient effect ring oscillator based TRNG (TERO-TRNG)
— Self-timed ring based TRNG (STR-TRNG)
— Phase-locked loop based TRNG (PLL-TRNG)

We implemented all the selected TRNGs in three FPGA families: Xilinx Spartan-6, Intel Cyclone V and Microsemi SmartFusion2.

During the evaluation, we compared selected TRNG cores according to following design criteria:

— Area
— Power consumption
— Output bit rate
— Power efficiency
— Entropy per output bit
— Entropy rate (bits of entropy per second)
— Design feasibility and repeatability

Every TRNG has its particular design difficulties, which are difficult to foresee. By implementing all of the selected TRNGs in three different FPGA families, we were able to pinpoint strong and weak points of each design. These findings allowed us to compare selected candidates in terms of their feasibility and repeatability, which is very important from practical point of view but not covered in many academic papers. Using the results of the evaluation presented in this chapter, a designer can choose a TRNG, which fits selected application the best. In order to make it easier to adopt one of the presented TRNG designs, the VHDL source code was made publicly available at `https://labh-curien.univ-st-etienne.fr/cryptarchi/HECTOR_TRNG_designs/`.

In addition to the implementation results, we proposed a universal scoring system, which can be adopted also to TRNG cores not evaluated in this chapter. The scoring system helps to visualize the performance of the TRNGs in different design categories. The visual representation makes it much easier to see the strong and weak points of the design.

As a result of the evaluation presented in this chapter, we selected two TRNG cores as targets for our next research in FPGAs:

— PLL-TRNG for its repeatability and optimization potential,

— STR-TRNG for its high performance.

The presented evaluation is restricted to implementation in FPGAs. To evaluate implementation in ASICs will be the objective of Chapter 3.

Work presented in this chapter was published in:

[33] O. Petura, U. Mureddu, N. Bochard, V. Fischer, and L. Bossuet, "A survey of AIS-20/31 compliant TRNG cores suitable for FPGA devices," in *26th International Conference on Field-Programmable Logic and Applications, FPL '16, Lausanne, Switzerland*, Aug. 2016

[36] O. Petura, U. Mureddu, N. Bochard, V. Fischer, and L. Bossuet, "A survey of AIS-20/31 compliant TRNG cores implemented on FPGAs," in *TRUDEVICE – 6th Conference on Trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE 2016), Barcelona, Spain*, Nov. 2016

[37] M. Deutschmann, S. Lattacher, J. Delvaux, V. Rozic, B. Yang, D. Singelee, L. Bossuet, V. Fischer, U. Mureddu, O. Petura, A. A. Yamajako, B. Kasser, and G. BATTUM, "HECTOR deliverable D2.1 – report on selected TRNG and PUF principles," Feb. 2016

Résumé

Les concepteurs de systèmes embarqués doivent implémenter des fonctionnalités de sécurité dans différents types de systèmes. Un TRNG étant un gage de confiance dans une sécurité intégrée, la conception d'un TRNG sécurisé est impérative pour la sécurité de l'ensemble du système. Par conséquence, la conception d'un TRNG sécurisé requiert une expertise considérable. Cependant, les concepteurs de systèmes intégrés n'ont pas beaucoup d'expertise dans le domaine de la sécurité, car le besoin de sécurité dans les systèmes intégrés est récent. Dans ce chapitre, nous avons proposé une méthodologie pour une évaluation équitable des différents noyaux TRNG afin de fournir un moyen clairement défini de choisir des TRNGs sécurisés, de les évaluer en fonction de critères spécifiques à l'application, de choisir le meilleur candidat pour l'application ciblée et de l'optimiser en fonction des besoins de l'application.

Nous avons présenté la méthodologie proposée sur un exemple d'évaluation de TRNG, réalisée dans le cadre du projet de recherche européen HECTOR. Dans l'évaluation présentée, nous avons évalué six noyaux de TRNG :

— TRNG basé sur des oscillateurs à anneau élémentaire (ERO-TRNG)
— TRNG basé sur des oscillateurs à anneau à échantillonnage cohérente (COSO-TRNG)
— TRNG basé sur des oscillateurs à anneaux multiples (MURO-TRNG)
— TRNG basé sur des oscillateurs à anneau à effets transitoires (TERO-TRNG)
— TRNG basé sur des oscillateurs à anneau auto-séquencé (STR-TRNG)
— TRNG basé sur des boucles à verrouillage de phase (PLL-TRNG)

Nous avons implémentés tous les TRNG sélectionnés dans trois familles de FPGA : Xilinx Spartan-6, Intel Cyclone V et Microsemi SmartFusion2.

Au cours de l'évaluation, nous avons comparé les noyaux TRNG sélectionnés selon les critères de conception suivants :

— Surface
— Consommation électrique
— Débit de sortie
— Efficacité énergétique
— Entropie par bit en sortie
— Taux d'entropie (bits d'entropie par seconde)
— Faisabilité et répétabilité de la conception

Chaque TRNG a ses difficultés de conception particulières, difficiles à prévoir. En mettant en œuvre tous les TRNG sélectionnés dans trois familles de FPGA différentes, nous avons pu identifier les points forts et les points faibles de chaque conception. Ces résultats nous ont permis de comparer les candidats sélectionnés en termes de faisabilité et de répétabilité, ce qui

est très important du point de vue pratique mais n'est pas abordé par de nombreux articles universitaires. En utilisant les résultats de l'évaluation présentée dans ce chapitre, un concepteur peut choisir un TRNG qui convient le mieux à l'application sélectionnée. Afin de faciliter l'adoption d'une des conceptions TRNG présentées, le code source VHDL a été rendu public sur `https://labh-curien.univ-st-etienne.fr/cryptarchi/HECTOR_TRNG_designs/`.

Outre les résultats de l'implémentation, nous avons proposé un système de notation universel, qui peut également être adopté pour les noyaux TRNG non évalués dans ce chapitre. Le système de notation permet de visualiser les performances des TRNG dans différentes catégories de conception. La représentation visuelle facilite la visualisation des points forts et des points faibles de conception.

À la suite de l'évaluation présentée dans ce chapitre, nous avons sélectionné deux noyaux TRNG comme cibles pour notre prochaine recherche sur les FPGA :

— PLL-TRNG pour sa répétabilité et son potentiel d'optimisation,

— STR-TRNG pour sa haute performance.

L'évaluation présentée ici se limite à l'implémentation dans les FPGA. L'évaluation de l'implémentation dans les ASIC sera l'objectif du chapitre 3.

Les travaux présentés dans ce chapitre ont été publiés dans :

[33] O. Petura, U. Mureddu, N. Bochard, V. Fischer, and L. Bossuet, "A survey of AIS-20/31 compliant TRNG cores suitable for FPGA devices," in *26th International Conference on Field-Programmable Logic and Applications, FPL '16, Lausanne, Switzerland*, Aug. 2016

[36] O. Petura, U. Mureddu, N. Bochard, V. Fischer, and L. Bossuet, "A survey of AIS-20/31 compliant TRNG cores implemented on FPGAs," in *TRUDEVICE – 6th Conference on Trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE 2016), Barcelona, Spain*, Nov. 2016

[37] M. Deutschmann, S. Lattacher, J. Delvaux, V. Rozic, B. Yang, D. Singelee, L. Bossuet, V. Fischer, U. Mureddu, O. Petura, A. A. Yamajako, B. Kasser, and G. BATTUM, "HECTOR deliverable D2.1 – report on selected TRNG and PUF principles," Feb. 2016

Chapter 3

Implementation of selected TRNGs in ASICs

FPGAs are a great platform for development and testing thanks to their reconfigurability. However, they are not as cost effective for mass production as application specific integrated circuits (ASICs). The initial cost of an ASIC is high but with it, it is possible to produce thousands or even millions of circuits using one mask once it is made. On top of that, ASICs offer some significant design advantages over FPGAs. On FPGAs, the designer cannot control the physical topology of the design. He can lock the design to a specific logic elements of an FPGA and on some FPGAs it is even possible to lock the routing, but beyond that, his possibilities are very restricted because the logical elements are already placed in a certain structure on silicon. On ASIC, however, the designer can be in complete control of the placement and routing of every single component of the design. In this chapter, we will briefly describe the available ASIC design flow, we will present TRNGs designed for HECTOR ASICs and present the results gathered from the manufactured ASICs.

3.1 ASIC design flow

The ASIC design flow is divided into two paths: full custom design flow and digital design flow. These two paths intersect at some points and both can be used to design a single chip. The full custom design flow gives the designer full control of all the design details such as exact placement and routing of design elements. On the other hand, the digital design flow offers good portability to other manufacturing technologies.

Logical design

The design of an ASIC begins with a logical design. This involves a schematic design for a full custom design flow or a VHDL/Verilog design for a digital design flow. The digital design flow usually relies on a standard cell library, which contains basic digital design elements such as logic gates and flip-flops. The high level VHDL/Verilog design can be translated into logical netlist (schematic) composed of standard cells.

A full custom design can still use standard cells, however, the design is done at schematic level already so there is no higher level description. In addition to standard cells, a full custom design may be composed of transistors directly, hence giving the designer more freedom in what and how to implement. This design flow is also usually used to create analog designs, which require precise parameter tuning.

Functional simulation

A functional simulation verifies whether the high level design description functions properly. This step is done only in the digital design flow before the translation of the design description into the logical netlist.

Electrical simulation on the schematic level

After the translation of the design description into the logical netlist in the digital design flow and after the schematic design is completed in a full custom flow, we need to perform an electrical simulation of the schematic (logical netlist). This simulation takes into account electrical models of transistors provided by the chip manufacturer and hence can give accurate prediction of various electrical parameters of the final circuit such as setup and hold times, maximal frequencies, etc.

Layout design

The layout design is a step, where we create a physical implementation out of a logical netlist (schematic). This step can be done automatically, especially when using standard cells, by automatic placement and routing of components. The physical layout is also one of the crucial differences between the FPGA and ASIC designs. On FPGAs, their physical structure is already made by the manufacturer while on ASIC we can do the physical implementation ourselves. By placing and routing the design manually, we can achieve a fine control of delays and critical paths, hence giving us a possibility to very fine tune the design. This advantage is even bigger for oscillatory structures, which are an integral part of most of the TRNGs used in digital devices.

Layout verification

The physical layout must be verified at two levels: design rules check (DRC) and layout versus schematic check (LVS). Design rules are a set of rules laid out by the technology vendor (chip manufacturer). These rules specify the manufacturability of the layout such as minimum spacing between certain elements, etc. Chip designs that do not pass a DRC are not accepted for manufacturing.

LVS compares schematic netlist to the netlist extracted from the layout. This way it provides the logical comparison between layout and schematic. The LVS is indispensable to assure the proper functionality of the design.

Post-layout simulation

Simulating the circuit after laying it out on silicon is very important to verify whether the physical layout changes the crucial parameters or not. The post-layout simulation takes into account the parasitic resistance, capacity and inductance of the layout elements. All these parasitics can affect circuit parameters such as delays.

IO ring construction

After all the design elements of the ASIC are completed, we need to construct an IO ring in order to connect the chip with the outer world. IO pads are provided in the standard cells library by the chip manufacturer, we just need to choose the ones we need and connect them correctly to the core of the chip. During the construction of the IO ring we must keep in mind not only the requirements of the core components but also the requirements of the IOs. There is a limited number of IOs we can place between two power supply pads. Also, some of the IOs have special requirements such as active temperature compensation blocks that must be placed in the core of the chip.

Final core assembly

Before finishing the entire ASIC design, we must place and route the core components. In this step, we lay out all of the core components and route them together. We should bear in mind the IO placement in the IO ring in order to facilitate the final step.

IO ring and core integration

Finally, we integrate the core of the chip with the IO ring. Besides routing the IOs, the most important part of this final step is routing of the power supplies. We need to ensure that all the

blocks are properly supplied with power and that we do not introduce any short circuits in the power supply.

3.2 HECTOR ASIC design

Two ASICs were developed in the frame of the HECTOR project. We selected ST CMOS 65nm as the target technology since ST was one of the industrial partners of the project. The design kit for this technology was provided via a french company CMP (Circuits Multi-Projets), which specializes in multi project wafer (MPW) runs.

Industrial partners of the HECTOR project preferred the digital design flow because of its portability and repeatability. So we tried to use the digital flow as much as possible but TRNG and PUF design, on which the HECTOR ASICs were aimed, requires low level implementation. Hence, we used manually placed standard cells and when it was absolutely inevitable, we did a full custom design.

Our university already had a licence for the Cadence design tools available, so we used these tools in all stages of the HECTOR ASIC design. The ST CMOS 65nm design kit, however, only supported design Calibre verification tools (DRC and LVS) made by Mentor Graphics. We needed to obtain additional licence for these tools, which was delivered only two months before the submission deadline for the ASIC design. So we had to do most of the ASIC design without verification tools and we did the design verification at the very last moment. Additionally, the verification tools lacked the support for some of the cells in the library of the design kit, namely PLLs, LVDS IOs and digital IO blocks, which resulted in our inability to do the formal verification of a complete design.

3.2.1 HECTOR ASIC evaluation platform

We decided to use existing HECTOR evaluation platform [3] for ASIC evaluation. We used existing motherboard of the evaluation platform and developed new daughter board for ASIC. The daughter board used Microsemi SmartFusion2 FPGA, on which we implemented an interface between HECTOR motherboard and the ASIC. Figure 3.1 shows the block diagram of the ASIC daughter board.

The FPGA receives commands from the motherboard and it sends responses and ASIC data back through a synchronous serial interface. It also synthesizes the ASIC clock and controls all of the external signals sent to ASIC.

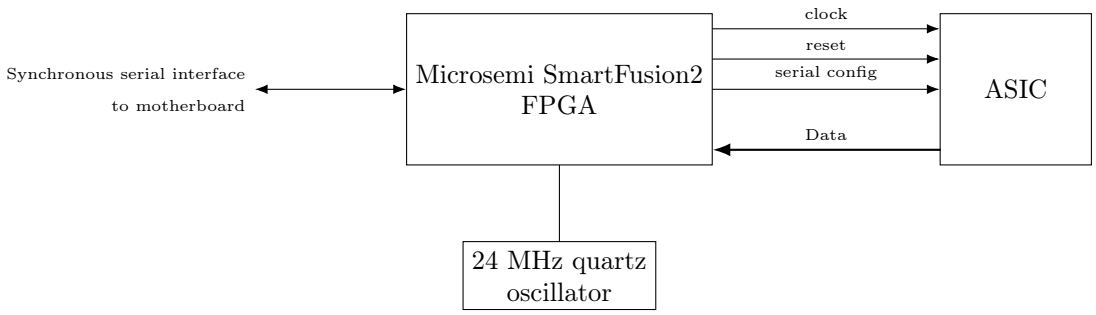


Figure 3.1: Block diagram of HECTOR ASIC daughter board

3.2.2 HECTOR ASIC v1

The first HECTOR ASIC contains the design of the PLL-TRNG and the test modules aimed at study of the TERO element. No more elements were placed in this first ASIC because the PLL-TRNG already took 0.4 mm^2, which was half of the total ASIC core area. In order to interface with the ASIC, we implemented 32-bit data output interface, 1 bit serial input interface and we also used LVDS outputs in order to output the high speed signals. Internally, the blocks are configured using a 88 bit wide parallel bus, which contains all the configuration data for all the blocks. The size of this bus is determined by the most demanding block so that the bus can accommodate all the data for this block. Figure 3.2 shows the block diagram of HECTOR ASIC v1 and Figure 3.3 shows its physical implementation.

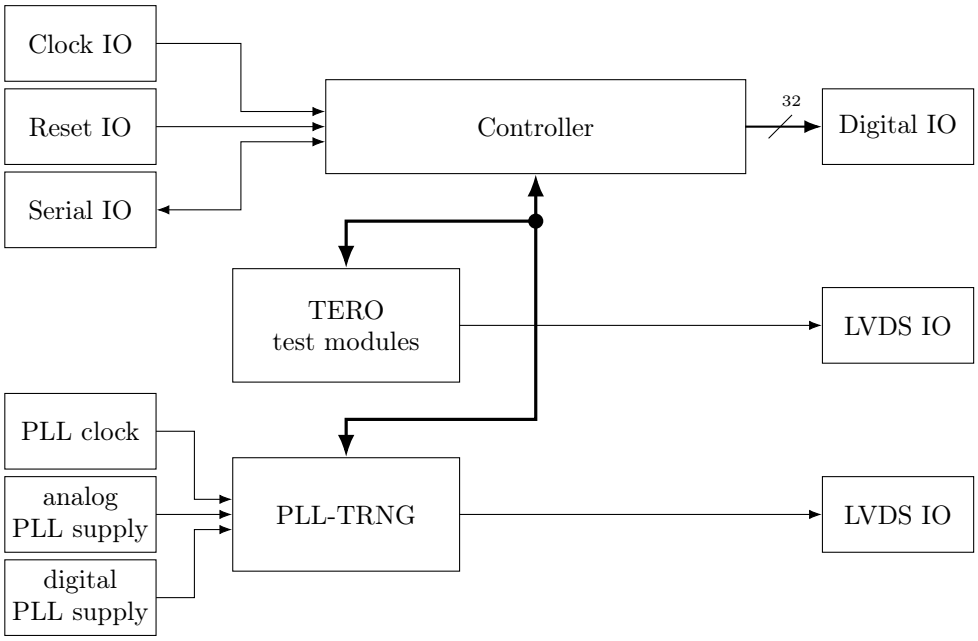


Figure 3.2: Block diagram of HECTOR ASIC v1

The TERO test modules were implemented in order to study TERO as the basic element of

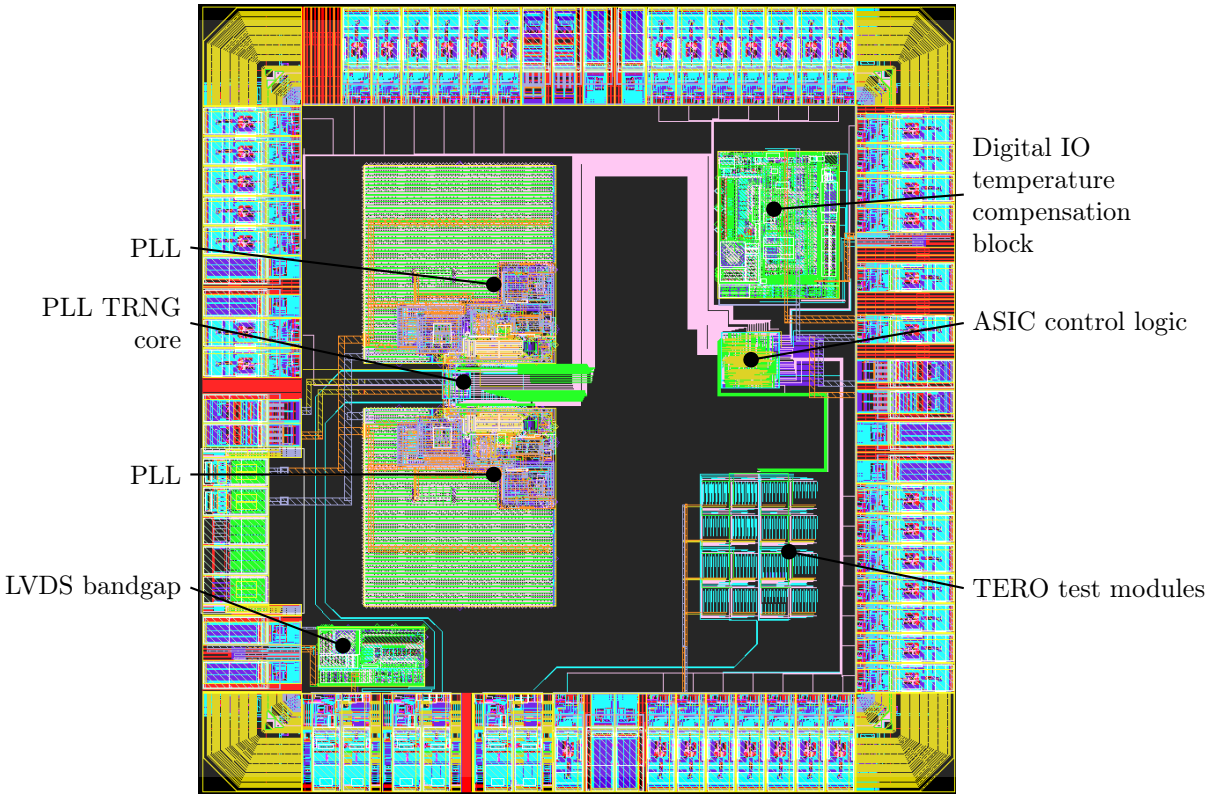


Figure 3.3: Physical layout of HECTOR ASIC v1

a physical unclonable function (PUF) but since PUFs are not in scope of this thesis, we will not discuss implementation details of this block. The PLL-TRNG, however, is much more interesting in scope of this thesis since we already discussed it in previous chapters.

3.2.2.1 PLL-TRNG in HECTOR ASIC v1

PLL-TRNG implemented in HECTOR ASIC v1 uses two PLLs connected to a TRNG core block. Implementation of the PLL-TRNG in ASIC brings more challenges than the implementation in the FPGA. In FPGA, one of the best features of the PLL-TRNG is that PLLs are physically isolated from the FPGA logic and hence there is only minimal (close to none) interference between the logic implemented in the FPGA and the noise source in PLLs. In ASIC, on the other hand, it is an engineering challenge to achieve a good level of isolation of the PLLs. PLLs must be isolated very well from the rest of the ASIC for several reasons:

— The PLL contains a high frequency oscillator, which radiates a lot and can interfere with other circuits.

— Analog parts of the PLL occupy large area, which increases the interference even more.

— The PLL requires several different power supplies in order to work properly. These supplies must be well distributed and isolated from each other.

— Input and output signals of the PLL are clock signals, which must be routed carefully in order to reduce the clock slack as much as possible.

— Analog and digital domains cross inside the PLL, which opens the door for cross talks.

Figure 3.4 shows the block diagram of the PLL-TRNG implemented in the HECTOR ASIC v1.

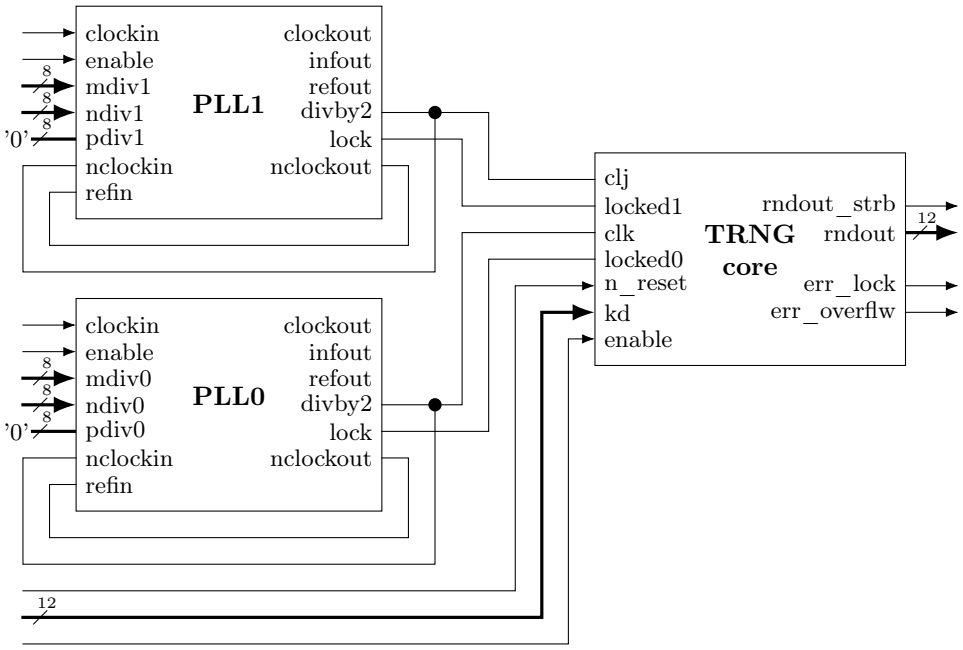


Figure 3.4: HECTOR ASIC v1 PLL-TRNG block diagram

The PLL IP was provided by ST Microelectronics and the documentation to it is confidential, hence we will provide only a brief description of the PLL's connections. The *clockin* port of the PLL is the input clock, *clockout* is the output clock and *divby2* output is the output clock with fixed frequency divider by two. The *divby2* output needs to be connected to *nclockin* when the PLL is used for frequency synthesis. So instead of routing two output signals, we decided to use only the *divby2* signal as the output of the PLL. Outputs *infout* and *refout* are used for skew measurements, which we do not do and hence these outputs are not connected. The feedback loop of the PLL closes through two connections: *divby2* needs to be connected to *nclockin* and *nclockout* needs to be connected to *refin*. Inputs *mdiv*, *ndiv* and *pdiv* are used to configure frequency dividers inside the PLL. We use only two of them because *pdiv* is used only at the *clockout* output, which we do not use. The *mdiv* divider is used before the phase frequency detector of the PLL to divide the input clock. The *ndiv* divider is then used in the feedback loop to multiply the output clock of the PLL.

All *enable* signals are tied together and they control whether the block operates or it is in a power-down mode.

The *mdiv, ndiv, kd* values have to be provided by the user and hence they are controlled via the control bus. The *mdiv* values represent multiplication factor of respective PLL. The *ndiv* values represent division factor of respective PLL. The value of *kd* has to be provided by the user even though it can be computed based on respective *mdiv* and *ndiv* values. This is to save space, which would have been occupied by a multiplier, in the ASIC.

The internal schematic of the PLL-TRNG core is depicted in Fig. 3.5.

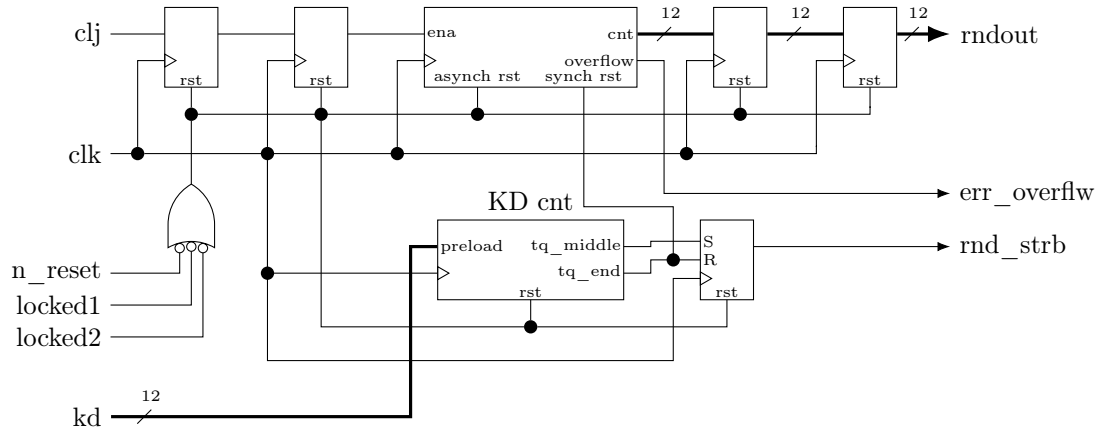


Figure 3.5: HECTOR ASIC v1 PLL-TRNG core schematic

We can see, that the heart of the TRNG core is the 12-bit counter as opposed to the XOR decimator seen in previous chapters. This counter counts the number of samples where *clj* is high during one T_q period. The LSB of the counter is, effectively, the random output bit. The full value of the counter can be used in embedded tests to compute T_q pattern parameters. This represents the main novelty of the new PLL-TRNG design, making its securing easier and more efficient.

3.2.3 HECTOR ASIC v2

The second HECTOR ASIC was manufactured for two main reasons: to conduct a more thorough study of the TERO element and to implement different TRNG cores, which could not have been implemented in the first ASIC due to the lack of space. From the TRNG perspective, we wanted to test the new approach of designing the STR-TRNG. But in order to do so, we needed two more TRNGs as a reference. We implemented the elementary ring oscillator based TRNG (ERO-TRNG) as a fundamental reference, because its straightforward design based on the free running oscillators. As a second reference, we also implemented an STR-TRNG according to the original principle proposed in [16].

The TERO test modules were also implemented in this second ASIC with some modifications. These modifications require some analog inputs. Since they were added to study PUFs, we will

not deal with them in this thesis.

The data input and output interface is the same as for the first HECTOR ASIC but we changed the internal configuration bus. Instead of a wide parallel bus we opted for a serial bus, which requires far less space and is much easier to route than a wide bus. Figure 3.6 shows the block diagram of HECTOR ASIC v2 and Figure 3.7 shows its physical layout.

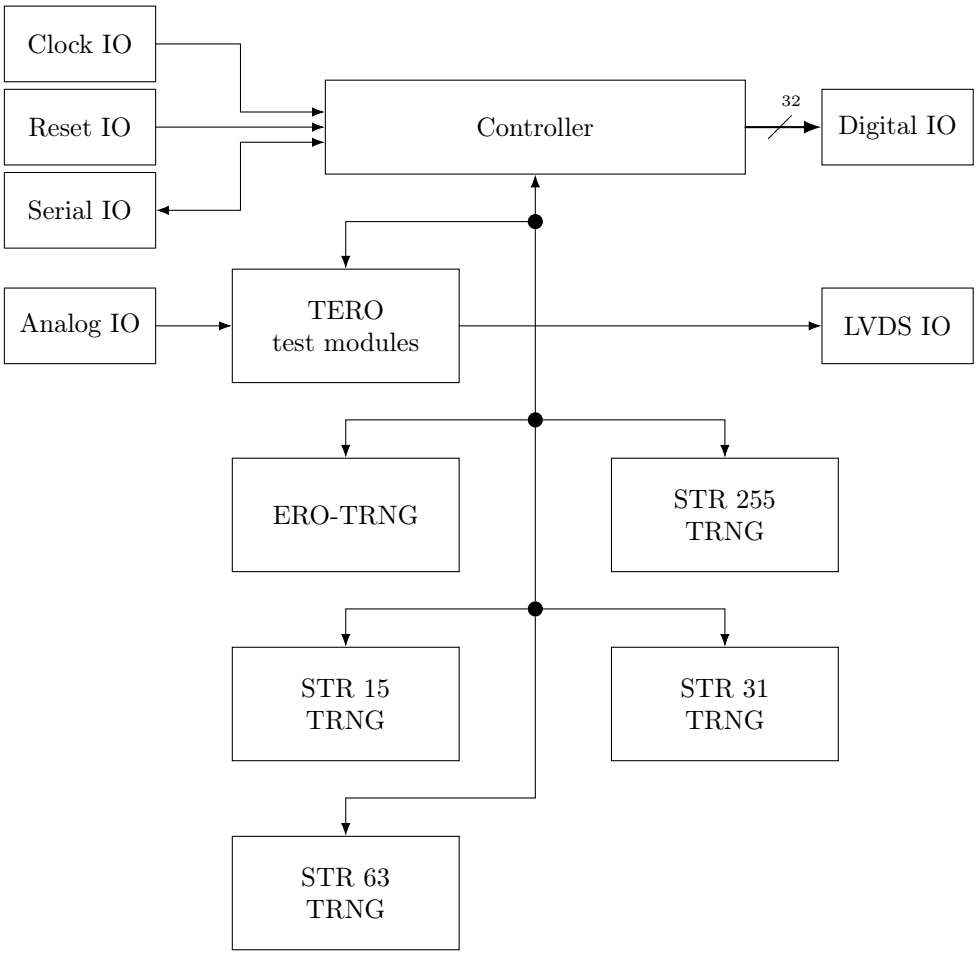


Figure 3.6: Block diagram of HECTOR ASIC v2

3.2.3.1 ERO-TRNG in HECTOR ASIC v2

We implemented ERO-TRNG in the ASIC v2 because of the simplicity of its design since simple design is more likely to work correctly in the final ASIC. We decided to include more than one ring oscillator configuration in the ASIC design in order to have more versions of this generator.

ERO-TRNG is very simple in its design. However, there are a few design details that might render it completely non functional when disregarded. The most important parameter to take into consideration is the oscillating frequency of the ring oscillators. When the frequency is not

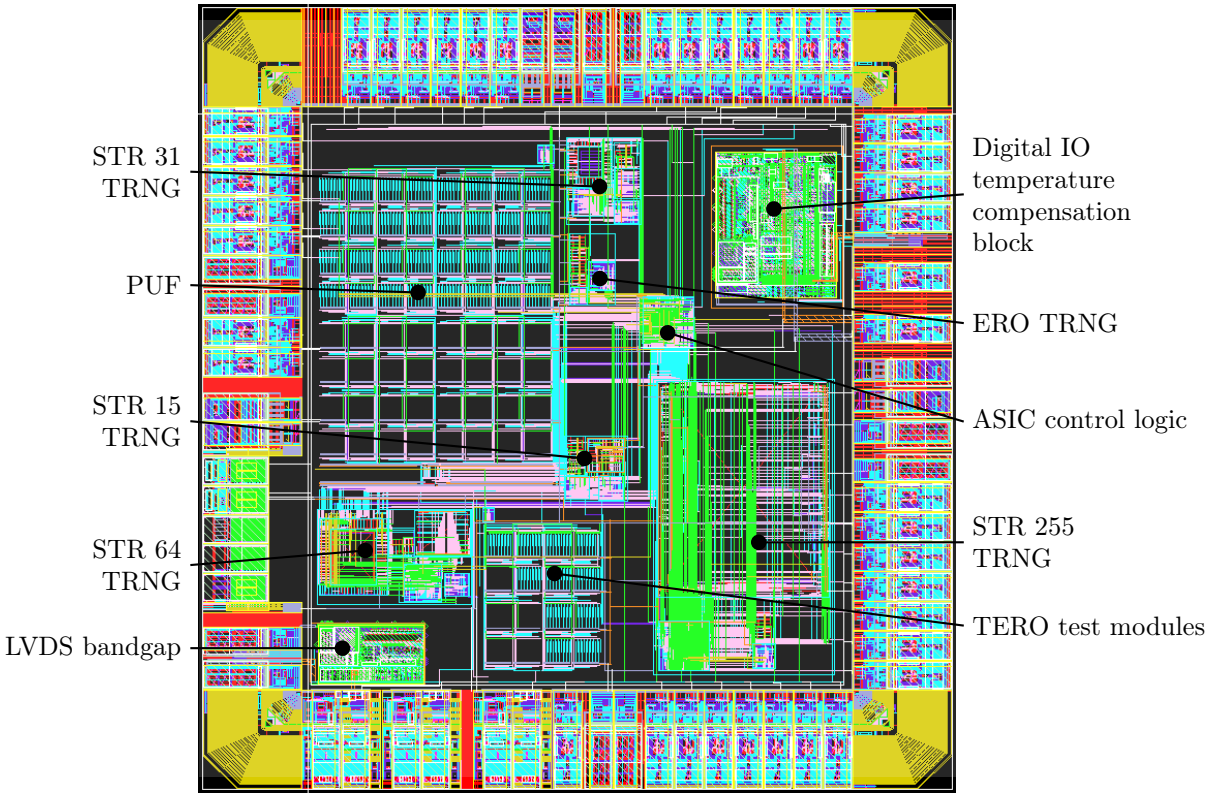


Figure 3.7: Physical layout of HECTOR ASIC v2

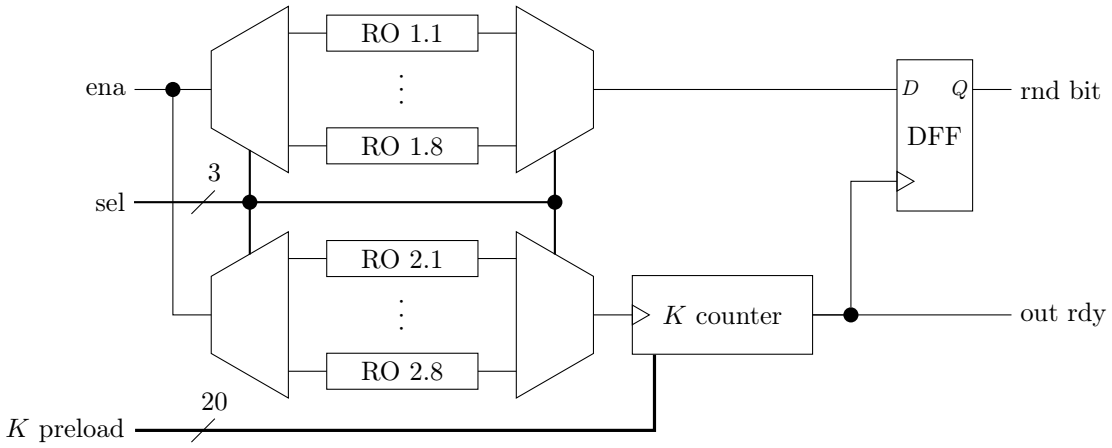


Figure 3.8: Architecture of the ERO-TRNG implemented in HECTOR ASIC v2

constrained in some way, the ring oscillator may achieve extremely high frequencies, which are impossible to follow by the logic. This is even more important in ASICs than in FPGAs, because in ASIC, the design cannot be reconfigured and ring oscillators are in general faster in ASIC than in FPGA. It is, however, very important to keep the balance because ring oscillators with low frequencies would also produce low output bit rate of a resulting TRNG. So on one hand we try to maximize the output bit rate but on the other hand we must keep the oscillating frequencies

within the operational limits of the logic. To minimize the risk, we decided to implement a TRNG using a bank of eight ring oscillators, from which it is possible to select one pair at a time. Next, we needed to design ring oscillators with different frequencies to put in this bank.

We performed a series of electrical simulations using Cadence Spectre simulator on a schematic level in order to determine the frequency of the ring oscillator. We found out, that the ring sizes we usually use in FPGAs (approx. 7 elements) oscillate at several GHz in the ASIC. So we used only the slowest inverter in the standard cell library in order to slow the entire ring down. This way, we achieved the frequency range of 300 – 500 MHz, which we set as a preliminary target frequency, using more than 128 inverters. 128 inverters is a huge number, which would greatly affect the area of an entire TRNG. So we experimented by alternating the inverter sizes in the ring. When a small inverter drives a bigger one, it has to charge big capacity at the input of the bigger inverter. But at the same time the small inverter can output only a small current. This way we slowed the ring oscillator down by charging a big capacity with a small current. Using alternate inverter sizes, we were able to achieve the frequency of 468 MHz using only 6 inverters. From this point, it took only a bit of parameter tweaking until we came up with the final ring frequencies, which we used in the ERO-TRNG. Table 3.1 summarizes these frequencies.

Ring oscillator	Oscillating frequency
RO1	350 MHz
RO2	429 MHz
RO3	500 MHz
RO4	559 MHz
RO5	634 MHz
RO6	672 MHz
RO7	724 MHz
RO8	916 MHz

Table 3.1: Ring oscillator frequencies used in ERO-TRNG in HECTOR ASIC v2

The jitter accumulation period of the TRNG can be set by a 20-bit K counter. The size of the counter was designed to be able to provide a frequency division factor of several hundreds of thousands, which is the value obtained by former studies (see Chapter 2).

3.2.3.2 STR-TRNG in HECTOR ASIC v2

We chose two different STR-TRNG principles to implement in the HECTOR ASIC v2. The principle proposed in [16] was implemented as a reference. One of the disadvantages of this original STR-TRNG principle is a large number of C-elements needed for sufficient entropy rate at the generator's output. To reduce the number of C-elements, we use jitter accumulation method in every stage of the STR chain. Figure 3.9 shows the principle of this modified STR-TRNG implementation.

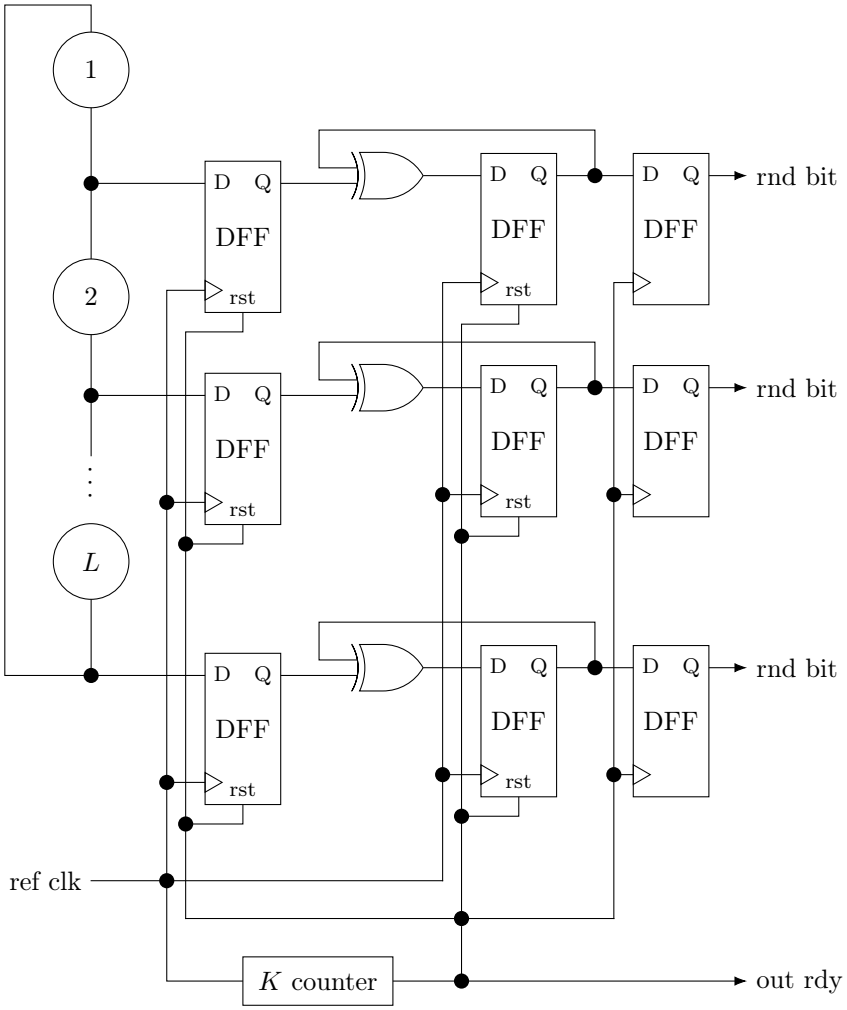


Figure 3.9: Schematic of an STR-TRNG using jitter accumulation implemented in HECTOR ASIC v2

The reference clock signal is generated in a bank of oscillators similar to the one used in ERO-TRNG design with only one difference. The fastest ring oscillator is replaced with a self-timed ring oscillator with L elements. We implemented three variants of this TRNG in order to assess its capacity to reduce the footprint of the conventional STR-TRNG design. STRs used in these three variants are of size $L = 15$, 31 and 63 elements. All of these configurations already greatly reduce the number of C-elements needed compared to 255 elements used in the reference implementation according to [16].

We used pre-chargeable C-elements in all the STR-TRNG implementations, which allows us to choose the initial population of tokens and bubbles in the ring. Figure 3.10 shows the schematic of such a C-element using standard cells.

Being able to initialize an STR in a free manner allows us to study the effect of various distributions of tokens and bubbles on the overall performance of the TRNG.

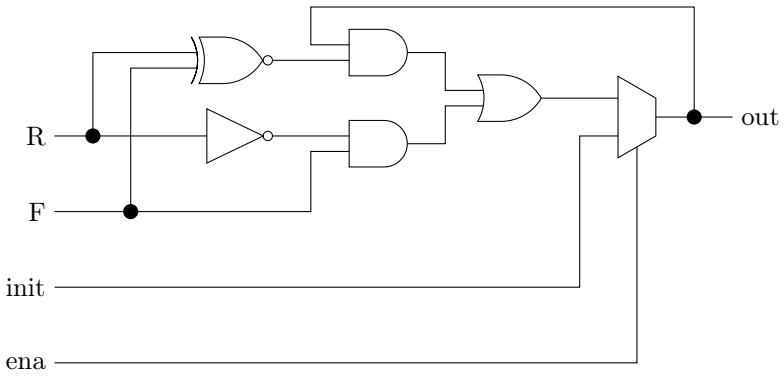


Figure 3.10: Schematic of a pre-chargeable C-element constructed from the standard cells

3.2.4 ASIC controller and control bus

Controllers for both HECTOR ASICs were designed in VHDL and the design was then synthesized into standard cells. Using the digital design flow, the controllers were placed and routed automatically. The role of the controller is to read the configuration through the serial configuration bus and then configure and activate the corresponding block in the ASIC. The controller then routes the digital output of the active block to the 32-bit digital output bus of the ASIC.

HECTOR ASIC v1 uses an internal configuration bus consisting of 88 bits. This bus is parallel and it connects all the ASIC elements. All of the 88 bits are shared between all the connected blocks and hence their meaning is different for every block:

— PLL-TRNG: 40 unused bits, 12 bits of K_D, 8 bits for ndiv1, 8 bits for mdiv1, 8 bits for ndiv0, 8 bits for mdiv0, 4 bits for active block identifier

— TERO test modules: 77 unused bits, 7 bits for TERO cell selection, 4 bits for active block identifier

The controller activates only the block identified by the 4-bit active block identifier.

HECTOR ASIC v2 contains blocks, which require more configuration bits and hence its configuration word is 280 bit wide. The parallel bus of this size would be tedious to route between all the blocks. So every block is equipped with its own configuration shift register and the configuration is passed using a serial bus with only two wires: data and clock. The only signal that has to be routed to all blocks independently then rests the enable signal. The configuration word contains the following information:

— ERO-TRNG: 4 bits for active block identifier, 20 bits for K preload value, padding 0, 3 bits for RO selection, 252 unused bits

— STR 15 TRNG: 4 bits for active block identifier, padding 0, 15 bits for STR initialization value, 16 bits for K preload, padding 0, 3 bits for sampling source selection, 240 unused bits

— STR 31 TRNG: 4 bits for active block identifier, padding 0, 31 bits for STR initialization value, 16 bits for K preload, padding 0, 3 bits for sampling source selection, 224 unused bits
— STR 63 TRNG: 4 bits for active block identifier, padding 0, 63 bits for STR initialization value, 16 bits for K preload, padding 0, 3 bits for sampling source selection, 192 unused bits
— STR 255 TRNG: 4 bits for active block identifier, padding 0, 255 bits for STR initialization value, 16 bits for K preload, padding 0, 3 bits for sampling source selection
— TERO test modules: 4 bits for active block identifier, padding 0, 7 bits for module selection, 268 unused bits

3.2.5 Interface to the outside world

Input and output pads and their corresponding IO blocks are necessary to get signals in and out of an ASIC. These IO blocks are arranged in a square or rectangle around the core of an ASIC and there are several types of IOs used in HECTOR ASICs:

— Core power supply – VDD and GND power supply connection of the ASIC core (1.2V). These power supply pins should be evenly spaced around the ASIC core in order to get even power distribution inside the core (low voltage loss).
— IO power supply – VDDE and GNDE power supply connections to supply the input/output circuitry. The documentation for IO blocks defines their power requirements. The IO power supply pads need to placed according to these requirements (only limited number of IOs may be between two IO supplies).
— Digital IOs – programmable digital IOs, which can be configured either as inputs or outputs. The digital IOs require digital IO power supply (3.3V) and they must be temperature compensated. For this purpose the temperature compensation block must be placed inside the ASIC core.
— Analog IOs – simple analog input/output pads. These pads require analog IO power supply (2.5V) and they do not require any temperature compensation.
— LVDS IOs – fast differential programmable IOs. They are similar to basic digital IOs but their maximum operating frequency is much higher. The LVDS IOs also require an additional compensation block, called the bandgap, to be implemented in the ASIC core.

Both HECTOR ASICs are equipped with 68 total IOs. HECTOR ASIC v1 has:

— Clock input
— Reset input
— 2 configuration inputs (config serial in and config ready)
— One configuration output, which confirms the reception of a configuration word

— One control input for TERO test modules
— 32-bit data output plus one data ready signal
— 3 LVDS outputs
— 5 pairs of 3.3V IO power supply pins (VDDE and GNDE)
— 3 pairs of 1.2V core power supply pins (VDD and GND)
— 3 pins for 2.5V analog IO supply (AVDDE, AGNDE and ESDSUB GND)
— One pair od 2.5V analog PLL power supply (PLL AVDD and PLL AGND)
— One PLL clock input
— One PLL lock error output

And HECTOR ASIC v2 has:

— Clock input
— Reset input
— 2 configuration inputs (config serial in and config ready)
— One configuration output, which confirms the reception of a configuration word
— One control input for TERO test modules
— 32-bit data output plus one data ready signal
— 3 LVDS outputs
— 5 pairs of 3.3V IO power supply pins (VDDE and GNDE)
— 3 pairs of 1.2V core power supply pins (VDD and GND)
— 3 pins for 2.5V analog IO supply (AVDDE, AGNDE and ESDSUB GND)
— 2 analog inputs
— 2 not connected pins in order to ensure pin compatibility with HECTOR ASIC v1

3.3 Testing and evaluation of TRNGs implemented in HECTOR ASICs

The first step towards testing the TRNGs implemented in HECTOR ASICs was to verify their correct functionality. We used HECTOR evaluation platform [3] to test the manufactured ASICs. First of all, we verified that both chips worked and that it was possible to communicate with them. This step revealed, that the LVDS outputs of both HECTOR ASICs were not functional, which reduced debugging possibilities for other components of the design. Further inspection of the design after consulting with the team of ST Microelectronics, who participated in HECTOR project, revealed that the LVDS outputs were not connected correctly. This error could have been avoided by the formal verification tools (DRC and LVS), but our version of the design kit did not support such a verification of LVDS outputs. LVDS outputs were intended to monitor high frequency signals at the output of PLLs and outputs of TERO test modules. Without these outputs, we could not effectively evaluate TERO test modules, but the PLL-TRNG functionality

was not impacted.

The core of the chip and the rest of the IOs, however, worked correctly and we were able to obtain the response of both chips at the data outputs. Next, we tested each TRNG individually. We acquired the output data for several reference configurations of each TRNG and we evaluated the statistical quality of the acquired data as well as the behavior of the TRNG compared to the simulations performed during the chip design.

3.3.1 Evaluation of PLL-TRNG in HECTOR ASIC v1

The PLL-TRNG implemented in HECTOR ASIC v1 was tested with PLL configurations found by optimized exhaustive search algorithm described in Chapter 4. We chose four representative configurations, which are presented in Table 3.2.

f_{ref} (MHz)	M0	N0	M1	N1	f_0 (MHz)	f_1 (MHz)	K_M	K_D	R (Mbits/s)	S (ps^{-1})	$R \cdot S$
24	183	11	83	5	399.273	398.4	913	915	0.4364	0.3645	0.1591
24	233	14	50	3	399.429	400	700	699	0.5714	0.2796	0.1598
24	131	8	50	3	393	400	400	393	1	0.1572	0.1572
24	33	2	149	9	396	397.333	298	297	1.3333	0.1180	0.1573

Table 3.2: PLL configurations of the PLL-TRNG tested in HECTOR ASIC v1

Every configuration was tested statistically using both AIS-20/31 and NIST 800-90B standard test suites. Table 3.3 shows the results of the statistical testing. The configurations are ordered by their output bit rate.

R	AIS-20/31		NIST 800-90B	
(Mbits/s)	result	Shannon entropy per bit	IID	min-entropy per bit
0.4364	pass	0.9998	IID	0.9817
0.5714	pass	0.9996	IID	0.9984
1	fail	0.9058	IID	0.9518
1.3333	pass	0.9999	IID	0.9979

Table 3.3: Results of statistical testing of the PLL-TRNG implemented in HECTOR ASIC v1

All selected configurations passed the IID track of NIST 800-90B statistical tests and only one of them did not pass AIS-20/31 statistical tests. Overall, we can conclude that the PLL-TRNG implemented in the HECTOR ASIC v1 worked properly and produced high quality random numbers. One configuration failing AIS-20/31 tests suggests, however, that the PLL-TRNG implemented in HECTOR ASIC v1 still requires more thorough testing, which we could not conduct because of time limitations.

3.3.2 Evaluation of ERO-TRNG in HECTOR ASIC v2

At first, we performed the functional verification of the TRNG. We acquired a short sequence of output data generated by each RO pair in the RO bank. The sequence was 10 000 bytes long and it was acquired using K=100 000. This verification revealed a flaw in the design of the ERO-TRNG. The output bit rate depended on the global ASIC clock rather than the sampling clock generated by an RO of the ERO-TRNG. This suggests that the sampled RO is sampled by the global ASIC clock and hence the sampled and sampling signals are not generated by identical oscillators.

Despite the flawed design of the TRNG, we decided to perform the statistical testing of a limited set of output data. We tested using only one RO out of 8 available in the RO bank. Table 3.4 shows the results of the statistical testing of ERO-TRNG.

K	AIS-20/31		NIST 800-90B	
	result	Shannon entropy per bit	IID	min-entropy per bit
100 000	fail	0.9856	IID	0.8090
200 000	fail	0.9861	IID	0.8105

Table 3.4: Results of statistical testing of the ERO-TRNG implemented in HECTOR ASIC v2

The results of statistical testing are not satisfactory, as we could expect based on the error in the design. None of the tested output sequences passed AIS-20/31 tests. But surprisingly, both of the sequences passed the IID track of NIST 800-90B tests. Their min-entropy is still quite low but there is a possibility that this TRNG would produce good quality random numbers if the design is corrected.

3.3.3 Evaluation of STR-TRNG in HECTOR ASIC v2

There are four STR-TRNGs implemented in HECTOR ASIC v2. The biggest one, using 255 C-elements, is the reference implementation of the STR-TRNG [16]. Other three TRNGs (featuring 15, 31 and 63 C-elements) were implemented using new architecture, which allows to output multiple bits at once.

The evaluation of STR-TRNGs began, similarly to ERO-TRNG, by their functional verification. This verification was performed by acquiring a short sequence of output data using K=10000 for each sampling oscillator available. The 15 element STR-TRNG was working only partially, because we were not able to acquire the output sequence for all of the sampling sources. Some of the sources did not produce any output at all. Unfortunately, the 15 element STR-TRNG was the only one that produced at least some output. We were unable to obtain any response from other STR-TRNGs.

When changing the sampling clock source of the STR15-TRNG we noticed that its output bit rate changed and hence the error of ERO-TRNG was not repeated in STR15-TRNG. This allowed us to proceed a little further with the functional evaluation and to compare the real frequencies of sampling sources with simulation results. Table 3.5 summarizes the expected and measured frequencies of sampling oscillators.

Sampling source	Expected frequency	Real frequency
1	350 MHz	44.91 MHz
2	429 MHz	not working
3	500 MHz	62.5 MHz
4	559 MHz	78.9 MHz
5	634 MHz	104 MHz
6	672 MHz	156 MHz
7	724 MHz	not working
8	STR	not working

Table 3.5: Comparison between expected and real frequencies of sampling sources of STR-TRNG in HECTOR ASIC v2

The real frequencies of sampling sources are much lower than the ones expected from simulations. This means that there are unforeseen parasitic elements that affect the resulting frequencies of oscillators.

To finalize the evaluation of STR-TRNG, we performed statistical testing of STR-TRNG outputs. We chose sampling sources 4 and 5 from Table 3.5 as a representative sample. Statistical testing was performed using K=10 000. Tables 3.6 and 3.7 show the results of statistical testing of the 15 element STR-TRNG.

Only one out of 30 tested output sequences did not pass the IID track of NIST 800-90B tests. It is only a slight anomaly since all of the tested sequences passed AIS-20/31 tests and all of the sequences achieve very high entropy rates.

The results of functional verification as well as statistical testing suggest that STR-TRNG is suitable for ASIC implementation and that it has the potential to produce good quality random numbers.

3.4 Conclusion

We chose three TRNG types to be implemented in ASIC:

— PLL based TRNG

— Elementary ring oscillator based TRNG

— Self timed ring based TRNG

All of these TRNGs were implemented in ST CMOS 65nm technology and two different ASICs were manufactured.

STR output	AIS-20/31		NIST 800-90B	
	result	Shannon entropy per bit	IID	min-entropy per bit
1	pass	0.9998	IID	0.9982
2	pass	0.9999	IID	0.9985
3	pass	0.9997	IID	0.9979
4	pass	0.9999	IID	0.9981
5	pass	0.9997	IID	0.9984
6	pass	0.9999	IID	0.9982
7	pass	0.9999	IID	0.9981
8	pass	0.9998	IID	0.9977
9	pass	0.9998	IID	0.9980
10	pass	0.9999	IID	0.9954
11	pass	0.9998	IID	0.9985
12	pass	0.9999	IID	0.9984
13	pass	0.9999	IID	0.9960
14	pass	0.9996	IID	0.9986
15	pass	0.9998	IID	0.9970

Table 3.6: Results of statistical testing of the 15 element STR-TRNG with the sampling source 4 from Table 3.5

STR output	AIS-20/31		NIST 800-90B	
	result	Shannon entropy per bit	IID	min-entropy per bit
1	pass	0.9998	IID	0.9980
2	pass	0.9999	IID	0.9982
3	pass	0.9999	IID	0.9982
4	pass	0.9999	IID	0.9971
5	pass	0.9999	IID	0.9984
6	pass	0.9996	IID	0.9980
7	pass	0.9999	IID	0.9986
8	pass	0.9996	IID	0.9980
9	pass	0.9999	IID	0.9979
10	pass	0.9999	non-IID	0.8688
11	pass	0.9998	IID	0.9972
12	pass	0.9996	IID	0.9978
13	pass	0.9998	IID	0.9962
14	pass	0.9999	IID	0.9977
15	pass	0.9999	IID	0.9986

Table 3.7: Results of statistical testing of the 15 element STR-TRNG with the sampling source 5 from Table 3.5

Electrical simulations performed at a schematic level during the design showed good preliminary results. However, manufactured chips did not always behave as expected based on these simulations. Some of the TRNGs were not working as expected and some of them were not working at all. These findings prove that relying solely on simulations during a TRNG design meant to be implemented in ASIC is not a good approach. In our case, some of the design errors could have been avoided by the verification tools, which we lacked. The two ASICs that we

produced are the first prototypes containing our design and their practical evaluation showed that another iterations of design, simulation and production would be needed to produce a fully working chip.

ERO-TRNG contains a design flaw, which would require another manufacturing round to correct: the output bit rate depends on the global ASIC clock rather than the sampling clock generated by an RO of the ERO-TRNG. Despite this flaw, the TRNG is working and we were able to acquire output data. Statistical analysis of output sequences showed that ERO-TRNG could produce good quality random numbers if the design flaw was fixed.

STR-TRNG was implemented in four different variants, out of which only one is working. Unlike the ERO-TRNG, we could not find the cause of the malfunction yet. However, the one STR-TRNG that is working produces random numbers of very high quality. Evaluation of STR-TRNG also revealed considerable differences between simulation and reality. The oscillators implemented in the TRNG are oscillating at frequencies several times lower than expected based on simulations. These findings highlight the fact that simulations are often insufficient in our context and a prototype should be produced in order to correctly verify the design.

The PLL-TRNG was the only TRNG implemented in the manufactured ASICs, which worked as expected. The suitable PLL configurations could be found using the optimized exhaustive search algorithm presented in Chapter 4 and the TRNG was producing random numbers of high quality.

Based on the results of evaluation of all the TRNGs implemented in HECTOR ASIC v1 and HECTOR ASIC v2, we can conclude that the PLL-TRNG is the most predictable TRNG to implement. Oscillatory structures do not always behave as expected based on simulations and they require a few design iterations to get to a working TRNG.

Because of the time limitations, we could not dive deeper into debugging the existing designs and hence it remains an open research topic.

In the next chapter, we will analyze the PLL-TRNG design in more detail.

Résumé

Nous avons choisi trois types de TRNG à implémenter sur ASIC :

— TRNG basé sur PLL (PLL-TRNG)

— TRNG basé sur des oscillateurs à anneau élémentaire (ERO-TRNG)

— TRNG basé sur des oscillateurs à anneau auto séquencé (STR-TRNG)

Tous ces TRNG ont été implémentés dans la technologie ST CMOS 65nm et deux ASICs différents ont été fabriqués.

Les simulations électriques effectuées à un niveau schématique lors de la conception ont donné de bons résultats préliminaires. Cependant, les puces fabriquées ne se sont pas toujours comportées comme prévu par ces simulations. Certains des TRNG ne fonctionnaient pas comme prévu et d'autres ne fonctionnaient pas du tout. Ces résultats prouvent que le fait de s'appuyer uniquement sur des simulations lors d'une conception TRNG destinée à l'implémentation dans un ASIC n'est pas une bonne approche. Dans notre cas, certaines des erreurs de conception auraient pu être évitées grâce aux outils de vérification, que nous n'avions pas. Les deux ASIC que nous avons produits sont les premiers prototypes contenant nos conceptions et leur évaluation pratique a montré qu'une autre itération de conception, de simulation et de production serait nécessaire pour produire des puces pleinement opérationnelles.

Le ERO-TRNG contient un défaut de conception qui nécessiterait un autre cycle de fabrication pour être corrigé : le débit à la sortie dépend de l'horloge d'ASIC globale plutôt que de l'horloge d'échantillonnage générée par un oscillateur à anneau de l'ERO-TRNG. Malgré cette lacune, le TRNG fonctionne et nous avons pu acquérir des données de sortie. L'analyse statistique des séquences de sortie a montré que le ERO-TRNG pourrait produire des nombres aléatoires de bonne qualité si le défaut de conception était corrigé.

Le STR-TRNG a été implémenté selon quatre variantes différentes, dont une seule fonctionne. Contrairement au ERO-TRNG, nous n'avons pas encore trouvé la cause du dysfonctionnement. Cependant, le STR-TRNG qui fonctionne produit des nombres aléatoires de très haute qualité. L'évaluation du STR-TRNG a également révélé des différences considérables entre la simulation et la réalité. Les oscillateurs implémentés dans le TRNG oscillent à des fréquences plusieurs fois inférieures aux prévisions basées sur simulations. Ces résultats mettent en évidence le fait que les simulations ne sont souvent pas suffisantes et qu'un prototype devrait être produit afin de vérifier correctement la conception.

Le PLL-TRNG est le seul TRNG implémenté dans les ASIC fabriqués, qui fonctionne comme prévu. Les configurations des PLL appropriées ont pu être trouvées en utilisant l'algorithme de recherche exhaustive optimisé présenté au chapitre 4 et le TRNG produit des nombres aléatoires de haute qualité.

Sur la base des résultats de l'évaluation de tous les TRNGs mis en œuvre dans HECTOR ASIC v1 et HECTOR ASIC v2, nous pouvons en conclure que le PLL-TRNG est le TRNG le plus prévisible à mettre en œuvre. Les structures oscillatoires ne se comportent pas toujours comme prévu par les simulations et elles nécessitent quelques itérations de conception pour parvenir à un TRNG fonctionnel.

En raison des contraintes de temps, nous n'avons pas pu approfondir le débogage des conceptions existantes et le sujet reste donc un sujet de recherche ouvert.

Dans le chapitre suivant, nous analyserons plus en détail la conception du PLL-TRNG.

Chapter 4

Design of a secure PLL-TRNG

The PLL-TRNG, proposed in [20] and modeled in [21], is especially well suited for implementation in FPGAs. Its implementation in ASICs is quite costly since PLLs require large silicon area. In most FPGAs, however, several PLLs are available, which greatly reduces the implementation cost. What is more, PLLs are physically and electrically isolated from the rest of the FPGA since they are implemented as hardwired blocks. The PLL-TRNG is also well repeatable because the implementation depends only on the configuration of digital components of the PLL: multiplication and division factors. In this chapter, we will study the PLL-TRNG design in depth. We will present the advantages and caveats of the PLL-TRNG design, propose some automated ways of the PLL-TRNG design optimization, and some dedicated embedded tests to secure the generator.

4.1 Overview of the PLL-TRNG design

The basic PLL-TRNG design based on one PLL is described in Section 1.2 with its block diagram showed in Figure 1.15. In this chapter, we will focus on the PLL-TRNG based on two PLLs since this configuration provides better control over the overall PLL-TRNG design parameters and it was selected by HECTOR partners depending on our results presented in Chapter 2. Figure 4.1 shows the two PLL variant of the PLL-TRNG.

The reference clock signal clk_{ref} is synthesized in the PLL 1, while in the single PLL variant of the TRNG the clk_{ref} is represented by the input of the PLL. This seemingly simple change in the design introduces new relationships between different design components and their parameters.

First of all, the sampled (also called jittered) clock clk_{jit} as well as reference clock clk_{ref} are both synthesized inside PLLs according to Eq. (4.1) and (4.2). This introduces a new

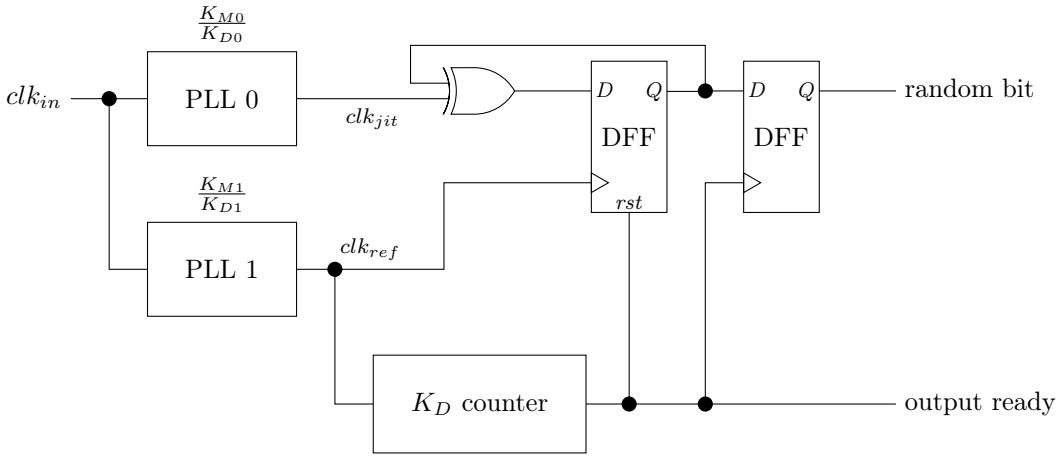


Figure 4.1: Two PLL variant of the PLL-TRNG

relation between clk_{jit} and clk_{ref}, which defines the behavior of the whole TRNG. Equation 4.3 demonstrates this relationship and defines the overall multiplication and division factors.

$$clk_{ref} = clk_{in} \cdot \frac{K_{M1}}{K_{D1}} \tag{4.1}$$

$$clk_{jit} = clk_{in} \cdot \frac{K_{M0}}{K_{D0}} \tag{4.2}$$

$$\begin{aligned} clk_{jit} &= clk_{ref} \cdot \frac{K_M}{K_D} \\ clk_{in} \cdot \frac{K_{M0}}{K_{D0}} &= clk_{in} \cdot \frac{K_{M1}}{K_{D1}} \cdot \frac{K_M}{K_D} \\ \frac{K_{M0}}{K_{D0}} \cdot \frac{K_{D1}}{K_{M1}} &= \frac{K_M}{K_D} \end{aligned} \tag{4.3}$$

$$\begin{aligned} K_M &= K_{M0} \cdot K_{D1} \\ K_D &= K_{D0} \cdot K_{M1} \end{aligned}$$

$$R = \frac{f_{ref}}{K_D} \tag{4.4}$$

$$S = f_{jit} \cdot K_D \tag{4.5}$$

Figure 4.2 shows the internal structure of the PLL as it is implemented in all major FPGA families (Intel Cyclone V, Xilinx Spartan-6, Microsemi SmartFusion2).

The frequency of the input signal clk_{in} is first divided by N, giving the input frequency of the phase frequency detector (PFD) – f_{pfdin}:

$$f_{pfdin} = \frac{clk_{in}}{N}. \tag{4.6}$$

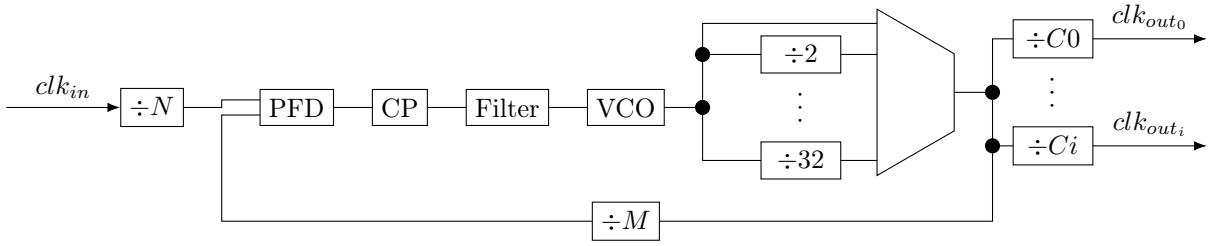


Figure 4.2: Internal structure of PLLs as implemented in all major FPGA families (PFD – phase frequency detector, CP – charge pump, VCO – voltage controlled oscillator, N, C_i – division factors of the PLL, M – multiplication factor of the PLL

The f_{pfdin} is compared with the feedback signal frequency in the PFD. Output current of the PFD depends on the phase difference between its inputs. This current feeds the charge pump (CP), which produces a control voltage for the voltage controlled oscillator (VCO). Before the control voltage is fed to the VCO itself, it must be filtered by a low pass loop filter, which removes unwanted spikes from the VCO control voltage. VCO generates the output signal with frequency proportional to this control voltage. This frequency is usually quite high and hence must be divided by the post-VCO divider (P_{VCOd}) and then by output dividers (C) for each individual output of the PLL. In order to synthesize frequencies higher than the input frequency, we must divide the VCO output frequency by the divider M, which is then fed back to the PFD. The output frequency of the VCO can be thus defined:

$$f_{VCO} = f_{pfdin} \cdot M \cdot P_{VCOd} \tag{4.7}$$

And the output frequency of the PLL is:

$$f_{out} = \frac{f_{pfdin} \cdot M}{C} \tag{4.8}$$

The PLL contains multiple programmable components, which allow us to set the desired output frequencies. However, every component of the PLL has its physical limits such as maximum input frequency, VCO frequency, etc., within which it must operate. Table 4.1 summarizes these limits for PLLs implemented in three major FPGA families.

Bearing all the physical limits in mind there is one additional constraint specific for the PLL-TRNG: the overall multiplication and division factors (K_M, K_D) must be coprime. From Figure 4.2 and Eq. (4.3), we can construct the overall K_M and K_D using individual dividers inside the PLL as stated in Eq. (4.9) and (4.10).

$$K_M = K_{M0} \cdot K_{D1} = M0 \cdot N1 \cdot C1 \tag{4.9}$$

$$K_D = K_{M1} \cdot K_{D0} = M1 \cdot N0 \cdot C0 \tag{4.10}$$

Table 4.1: PLL specifications for Intel Cyclone V, Xilinx Spartan-6, and Microsemi SmartFusion2 FPGA families

Parameter	Cyclone V		Spartan-6		SmartFusion2		Description
	Min	Max	Min	Max	Min	Max	
f_{in} [MHz]	5	500	19	540	1	200	Input frequency of the PLL
M	1	512	1	552	1	256	Multiplication factor (M-counter)
N	1	512	1	21	1	16384	Division factor of the input clock (N-counter)
$PVCOd$	1	2	1	1	1	32	Post-VCO division factor ($PVCOd$)
C	1	512	1	128	1	255	Division factor of the output clock (C-counter)
f_{pfdin} [MHz]	5	325	19	500	1	200	Input frequency of the phase frequency detector (PFD)
f_{VCO} [MHz]	600	1300	400	1000	500	1000	Operating range of the voltage-controlled oscillator (VCO)
f_{out} [MHz]	0	460	3.125	400	20	1000	Output frequency for internal global or regional clock

Given the number of constraints affecting the PLL-TRNG design, it is a tedious task to find a suitable configuration manually from millions of possible configurations. So we were first working on some methods of automatic design optimization in order to find suitable PLL configurations for PLL-TRNG easier.

4.2 PLL-TRNG design optimization

Finding a suitable PLL-TRNG configuration is a hard task and it requires a lot of experience in a PLL-TRNG design. The results of comparison presented in Chapter 2 show a clear example of this difficulty as the PLL-TRNG configuration, found manually, achieved only average results among all the compared TRNG cores. In order to improve the PLL-TRNG design we searched for an optimization method that would help us find suitable PLL-TRNG configurations faster and more reliably.

Since the relationships between all the PLL-TRNG parameters are well defined and the number of combinations is finite, the exhaustive search for the optimal configuration seems like a natural choice. However, the exploration space is so large that it would take years to find all the configurations of a PLL-TRNG for one single FPGA family on an average PC.

So as a second candidate, we evaluated suitability of meta heuristic algorithms, namely genetic algorithm, for PLL-TRNG design optimization.

4.2.1 Genetic algorithm explored

Genetic algorithms (GAs) belong to the group of evolutionary algorithms (EAs), which are widely used to solve searching or optimization problems. The GA mimics the behavior of natural evolution by survival of the fittest. GA operates over a population of individuals, called a generation. Each individual represents a solution to a problem expressed as a vector of values.

The initial generation of individuals is generated randomly. The GA then performs the following list of tasks for each generation of individuals:

1. Compute the fitness function for each individual.
2. Compute survival probability and reduce the generation size (eliminate the least fit individuals).
3. Repopulate the generation: perform crossover with some individuals and copy the rest.
4. Perform mutation on some individuals.

These steps are repeated until either the fitness of best individual is below set threshold or the generation limit is reached.

A fitness function represents a "desirability" of a particular individual. The survival probability of an individual is computed based on its fitness and population rejection rate. The rejection rate determines which portion of the generation is rejected.

After rejecting the least fit individuals, the generation is repopulated to its original size. This is done, mimicking the natural process, by performing crossovers on the most fit individuals. A crossover is an operation of combining the parameters of two parents and creating two children. Figure 4.3 shows single point crossover in more detail. Crossover may be performed with multiple crossover lines, thus creating multi-point crossover.

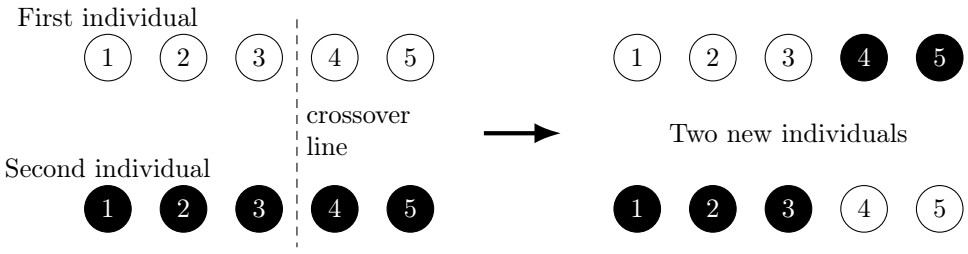


Figure 4.3: Crossover operation

After the population size is restored, the mutation probabilities are computed and selected individuals are mutated. A mutation is a process of randomly changing one or several parameters of an individual. This process allows the GA to create new individuals with unique parameters. Without mutation, the GA would be limited only to parameters contained in the initial random generation.

The whole process is repeated several times (usually a few hundred or thousand times). The average fitness of the entire population tends to increase with each new generation since only the fittest survive and are allowed to reproduce. The GA may stop sooner than the generation limit is reached if the fitness of at least one individual reaches the desired value.

We will now focus on particular implementations of genetic algorithms suitable to solve PLL-TRNG related problems. Interested readers can find further explanation of different genetic operations and algorithms in [38].

4.2.1.1 Generic open-source GA implementation

At first, we used an open-source GA optimization utility provided by [39]. This utility is suitable for the GA optimization with up to 5 design variables and 5 constraints, which was sufficient for a single PLL variant of the PLL-TRNG. We used 16 chromosomes in population, one-point crossover with a probability of 0.9. The mutation probability was 0.1 and random selection probability was also set to 0.1. We used 10 generations per run with 4 preliminary runs.

Since GA relies greatly on random initial selection, we ran the algorithm several times to find several PLL-TRNG configurations. We then chose three best candidates for each of the three FPGA families: Intel Cyclone V, Xilinx Spartan-6, and Microsemi SmartFusion2. The GA was configured to generate only the configurations within physical limits of PLLs used in tested families. Since we wanted to maximize the entropy rate at generator's output, we used the sensitivity to jitter S as the fitness function.

Table 4.2 presents three selected TRNG configurations for the three tested families. Even though all the configurations respected physical constraints of PLLs and were implementable in said families, we selected the best candidate for each family based on our previous experience with the PLL-TRNG design.

Compared to manual configurations used in Chapter 2, the GA was able to find better configurations for every tested FPGA family. This result is promising but the generic implementation we used had multiple disadvantages for our particular case. Limited number of design variables and constraints did not allow for optimization of the two PLL version of the PLL-TRNG and the generic implementation of the GA is not optimal for the PLL-TRNG. To overcome these caveats, we decided to implement our own GA in C.

4.2.1.2 Custom GA implementation

Our custom GA implementation allows to optimize a single PLL or two PLL variant of the PLL-TRNG. It is implemented in C for speed and efficiency and it supports mutation and single point crossover. In addition to more design variables and constraints, this implementation offers

f_{osc} [MHz]	M	D	$PVCOd$	C	f_{pfdin} [MHz]	f_{VCO} [MHz]	f_{out} [MHz]	K_M	K_D	R [Mbits/s]	S [ps^{-1}]	S^{-1} [ps]
Intel Cyclone V												
330	34	33	2	1	10.00	680	340	34	33	10.00	0.011	89.13
350	**131**	**37**	**1**	**3**	**9.46**	**1239**	**413**	**131**	**111**	**3.15**	**0.045**	**21.81**
338	198	64	1	2	5.28	1046	522	198	128	2.64	0.067	14.94
Xilinx Spartan-6												
430	**47**	**21**	**1**	**5**	**20.476**	**962**	**192**	**47**	**105**	**4.095**	**0.020**	**49.48**
400	52	21	1	15	19.047	990	66	52	315	1.269	0.021	48.08
430	48	12	1	27	20.476	982	36	48	567	0.758	0.021	48.45
Microsemi SmartFusion2												
59	149	29	2	1	2.034	606	303	149	29	2.034	0.008	113.75
200	**216**	**127**	**2**	**1**	**1.574**	**680**	**340**	**216**	**127**	**1.574**	**0.043**	**23.15**
200	291	199	2	1	1.005	584	292	291	199	1.005	0.058	17.18

Table 4.2: Three PLL-TRNG configurations found by the GA for each tested FPGA family. Best candidate is highlighted in bold.

also more flexibility in terms of data representation. The generic implementation used Microsoft Excel as a presentation tool. Our custom implementation uses comma separated values (CSV) file format and we can output any internal state of the algorithm.

At first, we used the same setup of the GA as in generic implementation in order to compare the results. The results of the custom implementation were comparable to the results of the generic one. This confirmed that our custom implementation works as expected.

In the next step, we performed several test runs of the genetic algorithm for the single and two PLL variants of the PLL-TRNG. We tried to tweak different algorithm parameters in order to obtain the best results but changing mutation and crossover probabilities did not work very well. So we decided to change the fitness penalty of solutions, which did not satisfy all the design constraints. The fitness penalty in our fitness function was the multiplier, by which the final fitness value was multiplied. Originally, we wanted to set this penalty to zero in order to eliminate every configuration that does not satisfy all the constraints.

Figure 4.4 shows the results of several runs of the GA with this setting. One out of 10 runs failed to find any suitable configuration of the PLL-TRNG. Out of the 9 successful runs, only two reached bit rates higher than 0.3 Mbit/s and the maximal bit rate was only slightly more than 0.6 Mbit/s. These results were not satisfying, so we increased the fitness penalty to 0.1 meaning that even if the particular PLL-TRNG configuration does not satisfy all the constraints, it still has a 10% chance to survive. In GA, even the configurations not meeting all the criteria might

produce better configurations in future generations by the means of crossover and mutation.

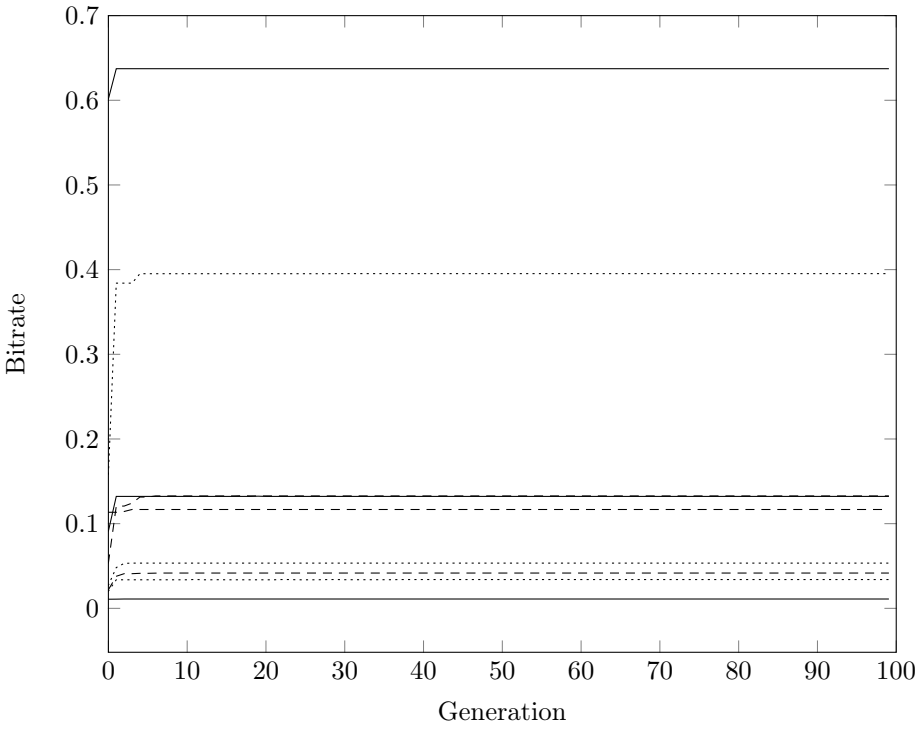


Figure 4.4: Best configurations of the single PLL variant of the PLL-TRNG found by several runs of the genetic algorithm. Fitness penalty set to 0.

Figure 4.5 shows 10 runs of the GA with fitness penalty set to 0.1. With this setup of the GA, we could consistently obtain PLL-TRNG configurations reaching bit rates around 1 Mbit/s and all the runs succeeded to deliver some results. Three runs achieved bit rates of more than 2 Mbit/s and the highest bit rate is 18 Mbit/s. This setup proved to be more successful than the previous one, so we used fitness penalty 0.1 even for the two PLL variant of the PLL-TRNG.

Figure 4.6 shows 10 runs of the GA for the two PLL variant of the PLL-TRNG. No matter how many times we repeated the experiment and tweaked the algorithm parameters, the results did not get any better. With all the bit rates being below 100 kbit/s we concluded, that the GA is not suitable to solve the problem of searching PLL-TRNG configurations for the two PLL variant.

4.2.2 Optimized exhaustive search

Since the genetic algorithm failed to reliably find suitable configurations for the two PLL variant of the PLL-TRNG, we searched further for the best method of parameter search. We analyzed the relationships between different design parameters and created an optimized exhaustive search algorithm, which finds all feasible PLL configurations for any given hardware. This

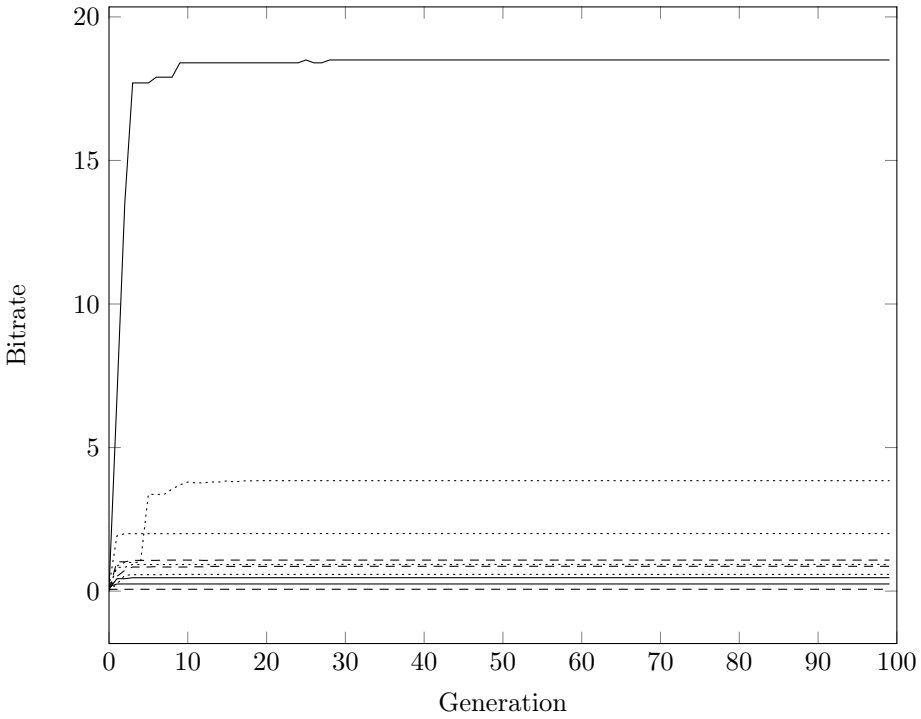


Figure 4.5: Best configurations of the single PLL variant of the PLL-TRNG found by several runs of the genetic algorithm. Fitness penalty set to 0.1.

algorithm provides a globally optimal solution, while the GA can find only a locally optimal one depending on initial conditions.

The optimized exhaustive search algorithm reduces the search space by reducing intervals of valid values for individual parameters of the PLL. This reduction is possible thanks to well defined relations between some of the parameters, which we obtained by a detailed study of PLLs.

Parameter search begins by choosing a valid f_{ref}. Then, for every PLL (denoted i) and every value of $PVCOd_i$:

1. We determine new minimal and maximal values of N_i. Based on Eq. (4.6), $N_i = \frac{f_{ref}}{f_{pfdin_i}}$. So new limits of N_i depend on the chosen f_{ref} and the limits of f_{pfdin}:

$$N_{min_i} = \max\left(N_{min}, \left\lceil \frac{f_{ref}}{f_{pfdin_{max}}} \right\rceil\right) \tag{4.11}$$

and

$$N_{max_i} = \min\left(N_{max}, \left\lfloor \frac{f_{ref}}{f_{pfdin_{min}}} \right\rfloor\right). \tag{4.12}$$

Note that $N_{max_i} - N_{min_i} \leq N_{max} - N_{min}$, showing that the search range of N_i is reduced.

2. For every value of N_i chosen from the new range, we find a new range of M_i. Based on

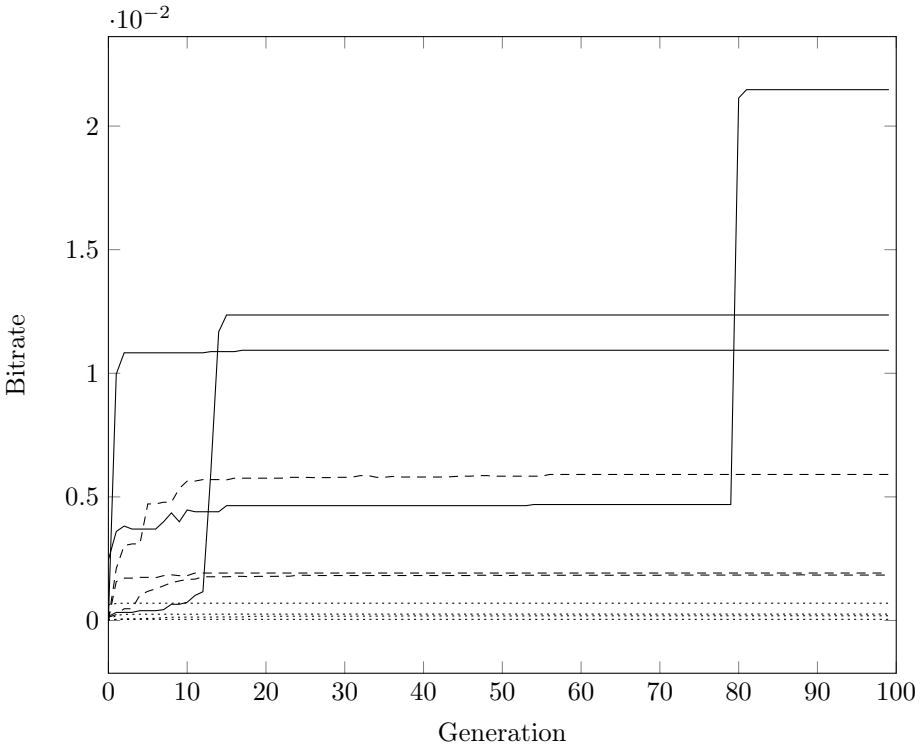


Figure 4.6: Best configurations of the two PLL variant of the PLL-TRNG found by several runs of the genetic algorithm. Fitness penalty set to 0.1.

Eq. (4.6) and (4.7), we can write:

$$M_i = \frac{N_i \cdot f_{VCO_i}}{f_{ref} \cdot P_{VCOd_i}}. \tag{4.13}$$

Since N_i, f_{ref} and P_{VCOd_i} are all fixed at this point, f_{VCO_i} is the only variable. Hence the new limits of M_i are:

$$M_{min_i} = \max\left(M_{min}, \left\lceil \frac{N_i \cdot f_{VCO_{min}}}{f_{ref} \cdot P_{VCOd_i}} \right\rceil\right) \tag{4.14}$$

and

$$M_{max_i} = \min\left(M_{max}, \left\lfloor \frac{N_i \cdot f_{VCO_{max}}}{f_{ref} \cdot P_{VCOd_i}} \right\rfloor\right). \tag{4.15}$$

3. Finally, for every value of M_i we determine a new range of C_i. Equations (4.6) and (4.8) give $C_i = \frac{f_{ref} \cdot M_i}{N_i \cdot f_i}$. At this point, all values except f_{out_i} are fixed. So we can define new limits of C_i:

$$C_{min_i} = \max\left(C_{min}, \left\lceil \frac{f_{ref} \cdot M_i}{N_i \cdot f_{out_{max}}} \right\rceil\right) \tag{4.16}$$

and

$$C_{max_i} = \min\left(C_{max}, \left\lfloor \frac{f_{ref} \cdot M_i}{N_i \cdot f_{out_{min}}} \right\rfloor\right). \tag{4.17}$$

Using this procedure, we can significantly decrease the search space, and hence search time, of all physically feasible configurations of PLLs in the PLL-TRNG. The search space for a single value of reference frequency was reduced from 72 057 594 037 927 936 possible configurations to 389 853 feasible configurations for Intel Cyclone V FPGA, from 2 201 591 218 176 to 89 025 configurations for Xilinx Spartan-6 and from 18 302 910 360 610 406 400 to 2 339 412 configurations for Microsemi SmartFusion2 FPGA.

We sped up the search of all the feasible configurations of PLLs, but not all of them are suitable for use in PLL-TRNG design. PLL-TRNG puts additional requirements on the PLL configuration:

— overall K_M and K_D must be coprimes,
— K_D must be odd,
— based on the stochastic model from [21], sensitivity to jitter S must higher than 0.09 ps^{-1} in order to reach the minimum Shannon entropy rate of 0.997 per bit as required by the AIS-20/31 standard [25],
— output frequency of both PLLs must not exceed the maximum frequency supported by the logic, which can be estimated by the timing analysis,
— K_M and K_D should be bounded.

The coprimality condition must be checked by the Euclidean algorithm and hence it slows the search process down rather than speeds it up. A considerable speed up can be achieved, though, by the oddity requirement of K_D. This requirement imposes that all M_0, N_1 and C_1 must be odd so they can be searched in steps of two, which effectively halves the search time for these values.

The security requirement on S is imposed by the stochastic model and is not questionable. It reduces the final number of configurations found but it does not reduce the search space itself.

Limiting the output frequency of the PLLs reduces the search space for parameters C_i, so it speeds the algorithm up.

Finally, in order to bind K_M and K_D we introduce s_M and s_D as their upper bounds. Based on Eq. (4.9) and (4.10), these bounds reduce the search space for C_0 and C_1. New maximal values are defined by Eq. (4.18) and (4.19).

$$C'_{max_0} = \left\lfloor \min\left(C_{max}, \frac{f_{ref} \cdot M_0}{N_0 \cdot f_{out_{min}}}, \frac{s_M}{M_1 \cdot N_0} \right) \right\rfloor \tag{4.18}$$

$$C'_{max_1} = \left\lfloor \min\left(C_{max}, \frac{f_{ref} \cdot M_1}{N_1 \cdot f_{out_{max}}}, \frac{s_D}{M_0 \cdot N_1} \right) \right\rfloor \tag{4.19}$$

Algorithm 1 summarizes all the constraints presented above and presents an easy to implement way to generate all the suitable configurations for PLL-TRNG.

Algorithm 1 Optimized exhaustive search algorithm for PLL-TRNG parameters

compute N_{min_0} from Eq. (4.11)
compute N_{max_0} from Eq. (4.12)
$N_{min_1} \leftarrow round_up_to_odd(N_{min_0})$
$N_{max_1} \leftarrow N_{max_0}$
`configs` $\leftarrow MAKEEMPTYLIST()$
for all P_{VCO_0} in Pvco_vals **do**
  for all P_{VCO_1} in Pvco_vals **do**
    for $N_1 = N_{min_1}$ **to** N_{max_1} by 2 **do**
      compute M_{min_1} from Eq. (4.14)
      compute M_{max_1} from Eq. (4.15)
      for $N_0 = N_{min_0}$ **to** N_{max_0} **do**
        compute M_{min_0} from Eq. (4.14)
        $M_{min_0} \leftarrow round_up_to_odd(M_{min_0})$
        compute M_{max_0} from Eq. (4.15)
        for $M_0 = M_{min_0}$ **to** M_{max_0} by 2 **do**
          compute C_{min_0} from Eq. (4.16)
          for $M_1 = M_{min_1}$ **to** M_{max_1} **do**
            compute C_{min_1} from Eq. (4.16)
            $C_{min_1} \leftarrow round_up_to_odd(C_{min_1})$
            compute C'_{max_0} from Eq. (4.18)
            compute C'_{max_1} from Eq. (4.19)
            for $C_1 = C_{min_1}$ **to** C'_{max_1} by 2 **do**
              compute K_M from Eq. (4.9)
              for $C_0 = C_{min_0}$ **to** C'_{max_0} **do**
                compute K_D from Eq. (4.10)
                if $gcd(K_M, K_D) = 1$ **then**
                  compute f_0 and f_1 from Eq. (4.8)
                  compute R from Eq. (4.4)
                  compute S from Eq. (4.5)
                  save into `configs`, values of $f_{ref}, P_{VCO_0}, P_{VCO_1}, M_0, N_0, C_0,$
                  $M_1, N_1, C_1, f_0, f_1, K_M, K_D, R, S$
                end if
              end for
            end for
          end for
        end for
      end for
    end for
  end for
end for

f_{ref} (MHz)	P_{VCO_0}	P_{VCO_1}	M_0	N_0	C_0	M_1	N_1	C_1	f_0 (MHz)	f_1 (MHz)	K_M	K_D	R (Mbit/s)	S (ps^{-1})	$R \cdot S$
Intel Cyclone V															
125	1	1	7	1	4	113	19	3	218.750	247.807	452	399	**0.548**	0.0988	0.0542
125	2	1	43	11	2	17	3	3	244.318	236.111	374	387	0.631	**0.0913**	0.0577
125	1	1	19	2	3	41	7	3	237.500	244.047	410	399	0.595	0.0973	**0.0579**
Xilinx Spartan-6															
125	1	1	43	5	5	17	7	3	215.000	236.110	410	399	**0.555**	**0.0913**	**0.0507**
Microsemi SmartFusion2															
125	1	1	7	1	4	113	19	3	218.750	247.807	452	399	**0.548**	0.0988	0.0542
125	1	4	29	5	3	25	13	1	241.660	240.384	375	377	0.641	**0.0906**	0.0580
125	1	4	23	3	4	33	17	1	239.580	242.647	396	391	0.612	0.0948	**0.0580**

Table 4.3: Best PLL configurations for the two PLL variant of the PLL-TRNG with jitter sensitivity $S > 0.09ps^{-1}$

We used this algorithm to search for PLL configurations on the three FPGA families: Intel Cyclone V, Xilinx Spartan-6 and Microsemi SmartFusion2. We limited the PLL output clock frequency to 250 MHz on the three families since the timing analysis of the PLL-TRNG design showed 250 MHz as its maximum frequency. The maximum frequency is the highest frequency, on which the logic of the TRNG core is guaranteed to work correctly.

To guarantee the security of the TRNG, it must achieve Shannon entropy of 0.997 per bit according to AIS-20/31 [25]. The stochastic model [21] links the sensitivity to jitter S with the entropy at generator's output. Based on this stochastic model, the sensitivity to jitter S must be at least 0.09 ps^{-1} in order to obtain the entropy $H_1 = 0.997$ per bit.

With all the physical and design constraints we found 188 suitable configurations out of 389 853 suitable ones for Intel Cyclone V (0.048%), 8 out of 89 025 for Xilinx Spartan-6 (0.0089%) and 9 976 out of 2 339 412 for Microsemi SmartFusion2 (0.426%). The number of suitable configurations is less than 1% of the feasible ones for all three tested FPGA families, which only underlines the fact that manual search is nearly impossible.

From all the suitable configurations, we selected three representative ones for each FPGA family: the one with the highest output bit rate R, best jitter sensitivity S and best product $R \cdot S$. Table 4.3 shows the selected configurations.

Properties marked in bold in Table 4.3 are those, for which the given configuration is the best among all the suitable configurations for the given family. For Intel Cyclone V and Microsemi SmartFusion2, there are three different configurations with the best bit rate, jitter sensitivity and $S \cdot R$ product respectively. But for Xilinx Spartan-6, there is only one configuration, which is the best among all the 8 suitable ones in all the regards.

4.2.3 Modifying the PLL-TRNG design to overcome its limitations

As we can see from the previous sections, most notably Table 4.3, the PLL-TRNG has a bit rate limited at approximately 0.6 Mbit/s when security constraints are kept (minimal jitter sensitivity S). However, HECTOR project required a TRNG implementation achieving an output bit rate of at least 1 Mbit/s [40] while satisfying the security requirements for a PTG.3 class according to AIS-20/31 [25].

Luckily, the PLL-TRNG is scalable enough to achieve higher bit rate without sacrificing the security. Figure 4.1 shows the basic design of a two PLL variant of the PLL-TRNG, where each PLL generates one output clock. These clocks are then used in the TRNG. In modern circuits, however, PLLs often have multiple outputs with controllable mutual phase shifts. And this feature is a key to achieve higher bit rates. Figure 4.7 shows a modified PLL-TRNG design, which takes advantage of multiple phase shifted clocks.

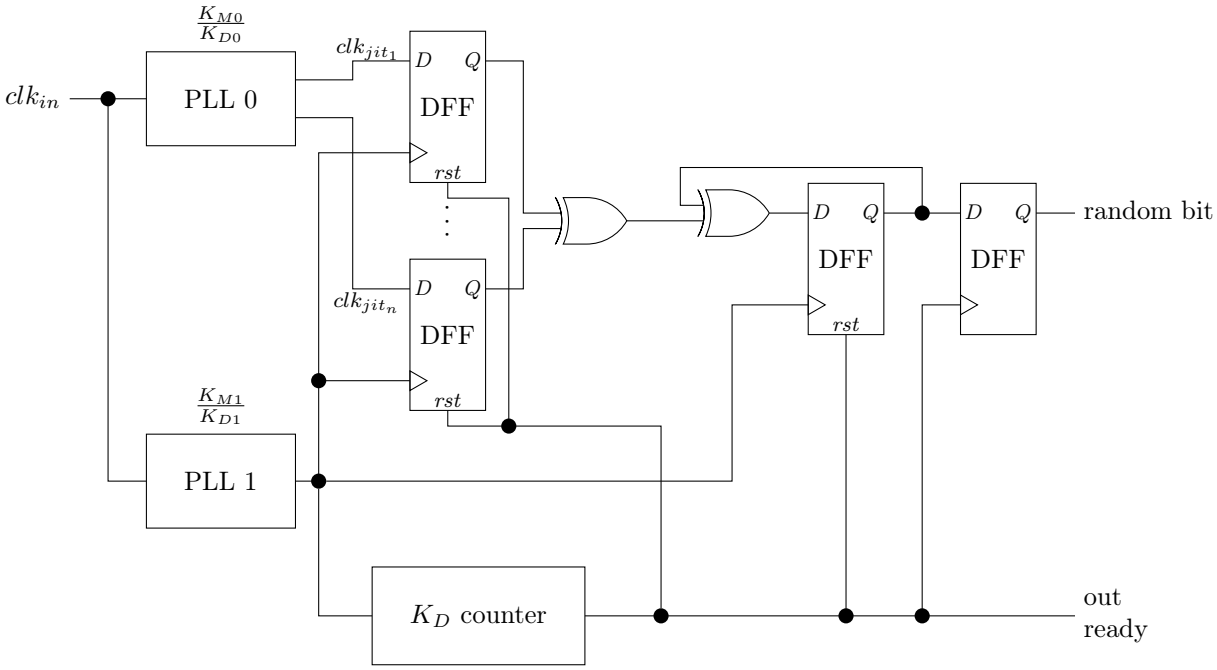


Figure 4.7: PLL-TRNG design using multiple phase shifted clocks to increase the output bit rate

Multiple clocks are sampled and XOR-ed, which multiplies the frequency by a number of clocks used. To avoid overlapping clock edges, the clocks must be shifted by $180/N$ degrees, where N is the number of clocks. Since the effective sampled frequency f_{jit} is multiplied by N, we can sample it at higher rates and hence achieve higher output bit rate of the TRNG.

Using this implementation of the PLL-TRNG we were able to find a suitable configuration, which satisfies the requirements of the HECTOR project. Table 4.4 shows the parameters of the HECTOR PLL-TRNG. The target FPGA for HECTOR TRNG was Intel Cyclone V FPGA, so

we present only one PLL-TRNG configuration for this particular FPGA. This configuration uses two PLLs, one of which has four clock outputs shifted by 45°.

Parameter	Value
f_{in}	125 MHz
M_0	37
N_0	6
C_0	4
$PVCOd_0$	1
f_{ref}	192.708 MHz
M_1	19
N_1	5
C_1	1
$PVCOd_1$	2
f_{jit}	475.000 MHz
Number of f_{jit} clocks	4
f_{jit} phase shift	45°
K_M	456
K_D	185
S	0.08788 ps^{-1}
effective S	0.35152 ps^{-1}
R	1.0417 Mbit/s

Table 4.4: HECTOR PLL-TRNG parameters

Using four clk_{jit} clocks, we achieved an output bit rate of 1.04117 Mbit/s, which satisfies the HECTOR requirement of at least 1 Mbit/s. The effective jitter sensitivity S is multiplied by the number of clk_{jit} clocks, which means that the base S according to Eq. (4.5) can be N times lower and still guarantee a sufficient entropy at the output of the TRNG.

4.3 Embedded tests

Embedded tests are a crucial part of modern TRNG design according to AIS-20/31. There are two fundamental types of these tests according to their functionality: total failure test and online tests.

The purpose of the total failure test is to detect when the entropy source breaks down. This test must react extremely fast in order to prevent any output from the failed entropy source. That is why the total failure test must be tailored to the entropy source used and it must be simple in its design. The total failure test in PLL-TRNG is the lock signal of both PLLs since both PLLs must be locked in order to generate jittered clock signals, which are the source of randomness in the PLL-TRNG.

Online tests, on the other hand, are meant to detect non tolerable statistical defects of the generator's output. The online tests must also be tailored to the particular TRNG design so that they can detect the particular defects that affect the particular design. But while the total failure test can be purely technical since its sole purpose is to quickly detect the total breakdown of the entropy source, the online tests must be based on the stochastic model of the particular TRNG design. A model from [21] can be used for the PLL-TRNG because it uses the principle of coherent sampling. Based on this stochastic model, measuring of parameters P_1 and P_2 was proposed in order to construct embedded tests for the PLL-TRNG [41].

Parameter P_1 is defined by Eq. (4.20) and corresponds to the number of samples within one period T_Q with probability $P(X_j = 1) \in (P_{min}; P_{max})$.

$$P_1 = \#\{(X_j)_j | P(X_j = 1) \in (P_{min}; P_{max})\} \tag{4.20}$$

Probability bounds P_{min} and P_{max} depend on the number N of periods T_Q, which are accumulated to compute $P(X_j = 1)$.

Parameter P_2 is defined by Eq. (4.21). This parameter does not depend only on the number of the unstable bits in the period T_Q but also on their position. Its maximal value is reached when the probability of one of the samples is equal to 0.5.

$$P_2 = \sum_{j=0}^{K_D-1} P(X_j = 1)(1 - P(X_j = 1)) \tag{4.21}$$

For the given number N of periods T_Q taken into account, the parameter P_2 can be estimated by Eq. (4.22) where A_j represents the number of cases in N periods T_Q, in which the random sample X_j was equal to one.

$$P_2 = \frac{1}{N^2} \sum_{j=0}^{K_D-1} A_j(N - A_j) \tag{4.22}$$

4.4 Stability of the PLL-TRNG

Some TRNGs are very sensitive to environmental conditions, supply voltage fluctuation and all other kinds of interference. By design, these TRNGs are fine tuned circuits working only when the conditions are right. But in real life applications, the TRNGs must work reliably under all kinds of conditions because they provide the root of trust. So we tested the stability of the PLL-TRNG and its sensitivity to the temperature and supply voltage.

4.4.1 Testing methodology

The PLL-TRNG was implemented in the Intel Cyclone V FPGA on a daughter board of the HECTOR evaluation platform [3]. A continuous stream of 2 MB of raw output data was taken for every measurement. The daughter board was placed inside a climatic chamber, where we can control the ambient temperature, and we used an external bench power supply to set the core voltage for the FPGA.

The FPGA used is a commercial grade circuit, which is guaranteed by the manufacturer to work within 0°C and 85°C. In order to stress test the TRNG design, we selected these five temperatures for our measurements: -20°C, 0°C, 40°C, 85°C, 100°C. We decreased the temperature from the highest selected towards the lowest one to minimize the risk of water condensing in the climatic chamber.

The nominal core voltage of the FPGA used is 1.1V and absolute maximum rating according to the datasheet [42] is from 1.07V to 1.13V. As for the temperature, we selected five test points: 1.04V, 1.07V, 1.1V, 1.13V, 1.17V.

We tested raw TRNG output data with the AIS-20/31 [25], NIST800-90B [32] and embedded [41] tests. Even though only the Procedure B is required to test raw data according to AIS-20/31, we decided to test also with the Procedure A to verify the statistical quality of the data. For NIST800-90B tests, we first determined whether the data is IID or non-IID and we ran the corresponding test suite afterwards. We recorded the values of the embedded tests P1 and P2 and verified if they are within the thresholds defined by the stochastic model: $P1 > 4$ and $P2 > 139$ [43].

4.4.2 Test results

Table 4.5 shows that the embedded test values P1 and P2 are always within the limits provided by the stochastic model ($P1 > 4$ and $P2 > 139$). What is more, the values are much higher than required even in extreme conditions outside of manufacturer's specified range. Successful results of embedded tests are further underlined by the statistical tests of the Procedure B of AIS-20/31 suite. All the required tests passed under all the tested conditions and the Shannon entropy per output bit is always much higher than required (at least 0.997 per bit). There were two cases, when the raw output of the TRNG did not pass the tests according to the procedure A, which is meant to test the statistical quality of the TRNG and compares it to the ideal TRNG. We repeat that this test procedure is meant for the post processed output of the TRNG, so if one or two out of 257 tests fail it may still be considered a success when raw random output is tested.

Table 4.6 shows the results of testing according to the NIST800-90B standard. The results show that the output of the TRNG is IID under all the tested conditions and the min-entropy

Temp. [°C]	Voltage [V]	Embedded tests P1	Embedded tests P2	AIS-20/31 Procedure A	AIS-20/31 Procedure B	AIS-20/31 Entropy
-20	1.16	28	842	Passed	Passed	1.0000
	1.13	26	841	Passed	Passed	1.0000
	1.10	28	814	Passed	Passed	1.0000
	1.07	29	821	Passed	Passed	0.9994
	1.04	25	820	Passed	Passed	1.0000
0	1.16	25	779	Passed	Passed	1.0000
	1.13	26	784	Passed	Passed	0.9996
	1.10	24	767	Passed	Passed	0.9997
	1.07	24	763	Passed	Passed	1.0000
	1.04	26	763	Passed	Passed	0.9999
40	1.16	23	750	Passed	Passed	0.9995
	1.13	24	749	Passed	Passed	1.0000
	1.10	26	744	Passed	Passed	0.9999
	1.07	23	733	Passed	Passed	1.0000
	1.04	24	735	Passed	Passed	1.0000
85	1.16	23	717	Passed	Passed	0.9996
	1.13	22	713	Passed	Passed	0.9998
	1.10	23	691	1/257 failed	Passed	0.9993
	1.07	22	716	Passed	Passed	1.0000
	1.04	21	702	Passed	Passed	1.0000
100	1.16	24	727	Passed	Passed	0.9999
	1.13	25	718	Passed	Passed	1.0000
	1.10	24	708	Passed	Passed	0.9999
	1.07	23	728	2/257 failed	Passed	0.9996
	1.04	24	712	Passed	Passed	1.0000

Table 4.5: PLL-TRNG temperature and voltage sensitivity tests according to AIS-20/31 and embedded tests

per bit is quite high. We ran also the non-IID test branch since its entropy estimation is more conservative but even non-IID min-entropy per bit is still high. The results of NIST800-90B tests are thus coherent with the results of the AIS-20/31 testing presented in Table 4.5.

4.5 Conclusion

The PLL-TRNG did not come out on a winning end from the evaluation in the Chapter 2. However, it is promising in terms of repeatability since once it is configured properly, it will work on all the devices. Also, since PLLs are already available in modern FPGAs, this TRNG is cheap to implement because it does not require a lot of logic elements. Physical isolation of the

Temp. [°C]	Voltage [V]	NIST800-90B IID branch	Min-entropy	Non-IID branch	Min-entropy
-20	1.16	Passed	0.9956	Passed	0.9279
	1.13	Passed	0.9953	Passed	0.9159
	1.10	Passed	0.9847	Passed	0.9269
	1.07	Passed	0.9835	Passed	0.9202
	1.04	Passed	0.9941	Passed	0.8934
0	1.16	Passed	0.9964	Passed	0.9179
	1.13	Passed	0.9904	Passed	0.9312
	1.10	Passed	0.9929	Passed	0.9056
	1.07	Passed	0.9868	Passed	0.8993
	1.04	Passed	0.9937	Passed	0.9072
40	1.16	Passed	0.9885	Passed	0.9072
	1.13	Passed	0.9927	Passed	0.9072
	1.10	Passed	0.9953	Passed	0.9020
	1.07	Passed	0.9938	Passed	0.9101
	1.04	Passed	0.9836	Passed	0.9159
85	1.16	Passed	0.9801	Passed	0.9020
	1.13	Passed	0.9913	Passed	0.9092
	1.10	Passed	0.9802	Passed	0.8967
	1.07	Passed	0.9912	Passed	0.9092
	1.04	Passed	0.9869	Passed	0.9020
100	1.16	Passed	0.9831	Passed	0.9119
	1.13	Passed	0.9951	Passed	0.8811
	1.10	Passed	0.9945	Passed	0.9265
	1.07	Passed	0.9806	Passed	0.9205
	1.04	Passed	0.9792	Passed	0.9102

Table 4.6: PLL-TRNG temperature and voltage sensitivity tests according to NIST800-90B

PLLs from the rest of the logic promises low sensitivity of the TRNG to crosstalks, variations in environmental conditions and fluctuation of the power supply. One of the biggest caveats, however, is the search for the suitable PLL configuration because there are a lot of possibilities and it takes considerable design experience to find a working configuration manually.

In this chapter, we first described the PLL-TRNG design and its constraints. Then, we explored the possibilities for automatic PLL parameter search. In this regard, we first looked at the genetic algorithm. A genetic algorithm is an evolutionary algorithm, which searches for the best possible solution for a given problem using an approach of the natural evolution. Firstly, it generates the first generation of solutions randomly. Then, it simulates a process of natural selection by deleting the least performant solutions and by recombination of the rest it produces

new solutions. This process is repeated until the best possible solution is found. The genetic algorithm proved to be a suitable way of searching for the PLL configurations but it provided only locally optimal solutions. By definition, the genetic algorithm cannot escape from the original random selection so it cannot search in the whole search space. Also, the two PLL variant of the PLL-TRNG proved to be very complex for the genetic algorithm. For the two PLL variant, the genetic algorithm could not provide consistent results.

So we moved on with our search and tried to find an algorithm that could provide a globally optimal solution. We studied closely the relationship between all the PLL parameters in the PLL-TRNG and we found some dependencies. Based on these dependencies, we designed an algorithm that is able to reduce the search space from all the possible PLL configurations to all the feasible ones. A feasible PLL configuration is a configuration, which is technically implementable respecting all the technical constraints of the PLL design such as maximal VCO frequency, frequency divisor ranges, etc. After limiting the search space, we applied also some additional constraints coming from the PLL-TRNG design and the resulting algorithm was able to find all the suitable PLL configurations. We consider a configuration suitable for the PLL-TRNG when it satisfies not only the physical PLL constraints but also the additional security constraints of the PLL-TRNG design. The optimized exhaustive search algorithm is able to find all the suitable configurations for a given device family (FPGA or ASIC). This algorithm was developed in close collaboration with Elie Noumon Allini and published in a common paper [44].

And lastly, we tested the sensitivity of the PLL-TRNG to the temperature and voltage variations. The device, we used for testing was a commercial grade Intel Cyclone V FPGA, which means that its operational temperature range is from 0°C to 85°C and the core voltage range is between 1.07V and 1.13V with a nominal value of 1.1V. We extended the testing temperature and voltage ranges beyond the operational range in order to stress test the PLL-TRNG in extreme conditions. For the temperature we selected -20°C, 0°C, 40°C, 85°C and 100°C. For the core voltage 1.04V, 1.07V, 1.10V, 1.13V and 1.16V. We recorded the raw random output of the TRNG and embedded tests values for each test point. We then tested the acquired data using AIS-20/31 Procedure A and B and NIST800-90B tests. The procedure A of the AIS-20/31 is not required for the raw random data but our acquired data succeeded even this test in most cases. Only in two cases it failed but only by a small margin where one or two out of 257 tests failed. All the tested data succeeded AIS-20/31 procedure B testing as well as NIST800-90B tests. The embedded test values were always within the range specified by the stochastic model.

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

[44] E. N. Allini, O. Petura, V. Fischer, and F. Bernard, "Optimization of the PLL configuration in a pll-based TRNG design," in *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018*, pp. 1265–1270, 2018

[46] J. Balasch, F. Bernard, V. Fischer, M. Grujić, M. Laban, O. Petura, V. Rožić, G. van Battum, I. Verbauwhede, M. Wakker, and B. Yang, "Design and testing methodologies for true random number generators towards industry certification," in *2018 IEEE 23rd European Test Symposium (ETS)*, May 2018

[43] G. Battum, S. Lattacher, M. Deutschmann, B. Kasser, M. Agoyan, J. Nicolai, M. Madau, R. Sussella, J. Balasch, M. Grujic, V. Fischer, O. Petura, M. Laban, J. Luhman, M. Wakker, and R. Malafre, "HECTOR deliverable D2.4 – robustness tests on TRNGs and PUFs," July 2018

Résumé

Le PLL-TRNG n'est pas sorti gagnant de l'évaluation du chapitre 2. Cependant, il est prometteur en termes de répétabilité, car une fois configuré correctement, il fonctionnera sur tous les circuits de la même famille. De plus, étant donné que les PLL sont déjà disponibles dans les FPGA modernes, ce TRNG est peu coûteux à mettre en œuvre car il ne nécessite pas beaucoup d'éléments logiques. L'isolation physique des PLLs par rapport au reste de la logique promet une faible sensibilité du TRNG à la diaphonie, les variations de conditions environnementales et aux fluctuations de l'alimentation. Cependant, la recherche de la configuration PLL appropriée constitue l'un des principaux inconvénients, car il existe de nombreuses possibilités et qu'il faut une longue expérience en conception pour trouver manuellement une configuration opérationnelle.

Dans ce chapitre, nous avons d'abord décrit la conception du PLL-TRNG et ses contraintes. Nous avons ensuite exploré les possibilités de recherche automatique des paramètres des PLL. À cet égard, nous avons d'abord examiné l'algorithme génétique. Un algorithme génétique est un algorithme évolutif, qui recherche la meilleure solution possible pour un problème donné en utilisant une approche de l'évolution naturelle. Tout d'abord, il génère la première génération de solutions de manière aléatoire. Ensuite, il simule un processus de sélection naturelle en supprimant les solutions les moins performantes et en recombinant le reste pour produire de nouvelles solutions. Ce processus est répété jusqu'à ce que la meilleure solution possible soit trouvée. L'algorithme génétique s'est avéré être un moyen approprié de rechercher les configurations de PLL, mais il n'a fourni que des solutions localement optimales. Par définition, l'algorithme génétique ne peut pas échapper à la sélection aléatoire d'origine et ne peut donc pas effectuer de recherche dans tout l'espace de recherche. De plus, la variante du PLL-TRNG avec deux PLLs s'est révélée très complexe pour l'algorithme génétique. Pour cette variante du PLL-TRNG, l'algorithme génétique n'a pas été en mesure de fournir des résultats cohérents.

Nous avons donc poursuivi notre recherche en essayant de trouver un algorithme pouvant fournir une solution globalement optimale. Nous avons étudié de près la relation entre tous les paramètres des PLL dans le PLL-TRNG et nous avons trouvé des dépendances. Sur la base de ces dépendances, nous avons conçu un algorithme capable de réduire l'espace de recherche de toutes les configurations de la PLL possible à toutes les configurations réalisable. Une configuration de PLL réalisable est une configuration qui peut être mise en œuvre techniquement en respectant toutes les contraintes techniques de la conception de la PLL, telles que la fréquence VCO maximale, les plages des diviseurs de fréquence, etc. Après avoir limité l'espace de recherche, nous avons également appliqué certaines contraintes supplémentaires issues de la conception du PLL-TRNG et l'algorithme résultant a pu trouver toutes les configurations de PLL appropriées. Nous considérons une configuration appropriée pour le système PLL-TRNG quand elle satisfait

non seulement aux contraintes physiques de la PLL, mais également aux contraintes de sécurité supplémentaires de conception du PLL-TRNG. L'algorithme de recherche exhaustive optimisée permet de trouver toutes les configurations appropriées pour une famille de circuits donnée (FPGA ou ASIC). Cet algorithme a été développé en étroite collaboration avec Elie Noumon Allini et publié dans un papier commun [44].

Enfin, nous avons testé la sensibilité du PLL-TRNG aux variations de température et de tension d'alimentation. Le circuit que nous avons utilisé pour les tests était un FPGA Intel Cyclone V de grade commerciale, ce qui signifie que sa plage de températures de fonctionnement est comprise entre 0°C et 85°C et que la plage de tension de cœur est comprise entre 1,07V et 1,13V avec une valeur nominale de 1,1V. Nous avons étendu les plages de température et de tension de test au-delà de la plage opérationnelle afin de soumettre le PLL-TRNG à un test de contrainte dans des conditions extrêmes. Pour la température, nous avons sélectionné -20°C, 0°C, 40°C, 85°C et 100°C. Pour la tension de cœur 1.04V, 1.07V, 1.10V, 1.13V et 1.16V. Nous avons enregistré la sortie aléatoire brute du TRNG et les valeurs des tests intégrés pour chaque point de test. Nous avons ensuite testé les données acquises à l'aide des tests AIS-20/31, procédures A et B et NIST800-90B. La procédure A de l'AIS-20/31 n'est pas requise pour les données aléatoires brutes, mais nos données acquises ont réussi même ce test dans la plupart des cas. Dans deux cas seulement, il a échoué, mais d'une faible marge : un ou deux des 257 tests ont échoué. Toutes les données testées ont réussi les tests de la procédure B de l'AIS-20/31 ainsi que les tests NIST800-90B. Les valeurs des tests embarqués se situaient toujours dans la plage spécifiée par le modèle stochastique.

Les travaux présentés dans ce chapitre ont été publiés dans:

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

[44] E. N. Allini, O. Petura, V. Fischer, and F. Bernard, "Optimization of the PLL configuration in a pll-based TRNG design," in *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018*, pp. 1265–1270, 2018

[46] J. Balasch, F. Bernard, V. Fischer, M. Grujić, M. Laban, O. Petura, V. Rožić, G. van Battum, I. Verbauwhede, M. Wakker, and B. Yang, "Design and testing methodologies for true random number generators towards industry certification," in *2018 IEEE 23rd European Test Symposium (ETS)*, May 2018

[43] G. Battum, S. Lattacher, M. Deutschmann, B. Kasser, M. Agoyan, J. Nicolai, M. Madau, R. Sussella, J. Balasch, M. Grujic, V. Fischer, O. Petura, M. Laban, J. Luhman, M. Wakker, and R. Malafre, "HECTOR deliverable D2.4 – robustness tests on TRNGs and PUFs," July 2018

Chapter 5

Randomness extraction and embedded testing of oscillator based TRNGs

Jitter of the clock generated by free running oscillators is the most commonly used source of randomness in modern digital devices. These free running oscillators are mostly ring oscillators [1, 14, 15], or self-timed rings [16]. The statistical quality and unpredictability of generated numbers depend on the jitter quality (composition) and size. So the jitter must be studied and characterized in order to correctly estimate the entropy at the output of the generator. The jitter must also be monitored continuously to guarantee a stable entropy rate.

Usually, many sources of randomness contribute to the overall entropy rate at the output of the RNG based on free running oscillators [47]:

1. *Secure sources* – random sources such as thermal noise, which are considered to be the best sources of randomness, because of their large and almost uniform signal spectrum similar to the white noise, they are mutually independent, and non manipulable (i.e. they cannot be manipulated by the attacker);
2. *Security critical sources* – random sources such as low frequency noises that feature some autocorrelation, which reduces the entropy rate at the generator output, while making entropy estimation very complex because of long term dependencies;
3. *Dangerous sources* – environmental, data dependent and correlated sources, which can be random or deterministic. Their contribution to random number generation must be avoided by the design, since they can be manipulated. If the manipulation cannot be avoided, it must at least be detectable through dedicated embedded tests.

These different sources of randomness are almost impossible to isolate in practice, which means that more, and sometimes all, of the noise sources are present in a device and hence in the

generator. This would not be a big security issue if: 1) only the contribution of secure sources was taken into account in entropy rate estimation; 2) the generated numbers were impossible to manipulate.

In [1], Sunar *et al.* use an urn stochastic model to estimate the entropy rate at the output of the generator using a huge number of ring oscillators, which the authors claimed were independent. However, the model does not account for possible dependencies between the outputs of the ring oscillators, which can even cause the rings to lock [48].

In [14], Baudet *et al.* proposed a comprehensive stochastic model for an elementary oscillator based random number generator sampling the jittered clock signal. In their model, the entropy rate at the generator output is estimated from the variance of the random jitter component that originates from the thermal noise.

The output numbers generated by both generators may be biased depending on the duty cycle of the sampled signal(s). Although both generators use the clock signal generated in the rings as a source of randomness, only the model proposed by Baudet *et al.* estimates entropy rate from the jitter component originating from the thermal noise and hence avoids overestimating entropy. Distinguishing between the contribution of thermal and low frequency noise is, however, very difficult. In [49], Haddad *et al.* computed the variance of the jitter for various accumulation times and then computed the jitter component originating from the thermal noise by curve fitting. This method has two disadvantages: 1) its precision depends to a great extent on the precision of the curve fitting algorithm; 2) it is not suitable for monitoring the jitter inside the device.

In [22], Fischer and Lubicz proposed a method of evaluation of the variance of the random jitter originating from the thermal noise that is suitable to be embedded in logic devices and hence used for online evaluation of the entropy rate at the output of the generator. However, depending on the initial phase of the two clock signals and the jitter accumulation time, the method can give incorrect results. This error can be corrected by using different accumulation times, but it turns out that it is not easy to make this correction automatic.

There are several problems to solve and we decided to tackle them one by one. So we studied how different methods of randomness extraction affect the quality of generated random numbers. We looked into the most common method of randomness extraction from the jittered clock by its sampling after a jitter accumulation period and compared it to counting the jittered clock periods during the jitter accumulation period. Then we studied the use of the Allan variance instead of commonly used statistical variance to monitor the jitter quality since Allan variance is less sensitive to low frequency noises. We also examined the use of two identical oscillators versus one ring and one quartz oscillator in a TRNG design and the effects of this simple design change on generated numbers. And last, but not least, we looked into statistical methods being

able to detect dependencies in the output of the RNG.

5.1 Comparison of different randomness extraction methods

Random variations of the clock signal generated by free running oscillators can be transformed into 1-bit or l-bit random numbers by two fundamental methods:

1. Sampling the jittered clock signal after a sufficiently long entropy accumulation period. Figure 5.1 (a) shows such a method.
2. Counting the periods of the jittered clock signal during the entropy accumulation period. Figure 5.1 (b) shows this method of randomness extraction.

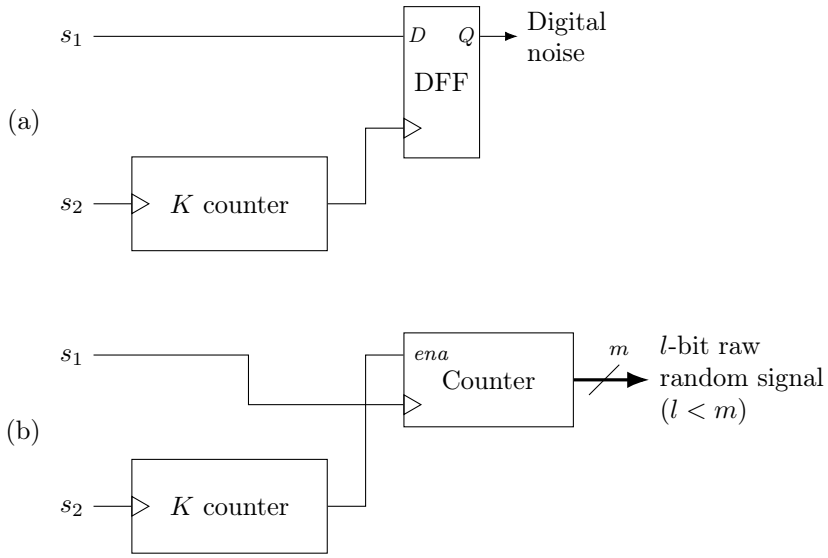


Figure 5.1: Randomness extraction using (a) sampling of jittered clock and (b) counting the jittered periods

The first method of randomness extraction consisting in the sampling of the jittered clock signal is the most commonly used method of randomness extraction in TRNG design based on free running oscillators. This is mostly because of its simplicity. But this method of extraction is very sensitive to dependencies between clock signals and to the duty cycle, which may introduce bias into generated random numbers. The bias gets introduced because the output bit of the RNG depends on the logic state of the sampled signal at the time of sampling, which depends on the duty cycle.

The second method consisting in counting the jittered clock periods removes the dependence of the generated random numbers on the duty cycle of the clock signal. The counter counts the number of rising edges of the signal s_1 during K periods of signal s_2. This number of rising edges does not depend on the duty cycle of any of the two signals. Counting the jittered clock

periods also removes the dependencies of the two clock signals because it transforms the time domain events into the frequency domain. On top of that, the counter values can be used as a basis of embedded tests since they contain more information than 1-bit random values output by a simple sampler.

In order to compare the two randomness extraction methods, we performed a series of statistical tests using standard statistical test suites defined by AIS-20/31 and NIST 800-90B standards. We generated random bit streams with K ranging from 10 000 to 100 000 periods of s_2 while counting the jittered periods and with K ranging from 2 000 to 100 000 periods of s_2 while sampling the jittered clock. We extracted the least significant bit of counter values and their first order differences as output random bits. Sources of randomness used were:

— two identical ROs oscillating at frequencies between 125 and 127 MHz

— two identical STRs generating a signal at frequencies between 128 and 130 MHz

Tables 5.1 and 5.2 show results of formal testing of the TRNG outputs using sampling method of randomness extraction.

K	AIS Test procedure B	AIS T8 Shannon entropy	NIST 800-90B IID	NIST 800-90B min-entropy
100000	failed	0.9935	non-IID	0.7491
50000	failed	0.9992	non-IID	0.7495
30000	failed	0.9847	non-IID	0.6376
25000	failed	0.9902	non-IID	0.6335
20000	failed	0.9851	non-IID	0.6498
15000	failed	0.9848	non-IID	0.6462
10000	failed	0.9844	non-IID	0.6461

Table 5.1: Entropy estimation using two internal ROs and the sampling method of extraction.

K	AIS Test procedure B	AIS T8 Shannon entropy	NIST 800-90B IID	NIST 800-90B min-entropy
100000	failed	0.9076	non-IID	0.3595
50000	failed	0.9030	non-IID	0.3596
30000	failed	0.9021	non-IID	0.3606
20000	failed	0.9074	non-IID	0.3596
10000	failed	0.9072	non-IID	0.3611

Table 5.2: Entropy estimation using two internal STRs and the sampling method of extraction.

None of the configurations using the sampling method of extraction passed any of the statistical tests. The entropy estimates from all estimators were very similar for all values of K, which may suggest that all of these TRNGs generate numbers of similar quality so we would need to wait much more than 100 000 periods of s_2 to produce good quality random numbers. It is also noticeable that STRs generated random numbers of lower quality than ROs.

Tables 5.3 and 5.5 show the results of formal testing of the TRNG based on counting the jittered periods when the least significant bit of the counter values was taken as random bit. Tables 5.4 and 5.5 show the results of formal testing when the least significant bit of counter value difference was taken as random bit.

K	AIS Test procedure B	AIS T8 Shannon entropy	NIST 800-90B IID	NIST 800-90B min-entropy
100000	passed	0.9999	IID	0.9945
25000	passed	0.9999	IID	0.9947
20000	passed	0.9999	IID	0.9947
15000	passed	0.9998	IID	0.9954
10000	failed	0.9966	non-IID	0.7086
2000	failed	0.0910	non-IID	0.1876

Table 5.3: Entropy estimation using two internal ROs and extracting the least significant bits of counter values.

K	AIS Test procedure B	AIS T8 Shannon entropy	NIST 800-90B IID	NIST 800-90B min-entropy
100000	passed	0.9999	IID	0.9937
25000	passed	0.9999	IID	0.9954
20000	passed	0.9999	IID	0.9947
15000	passed	0.9998	non-IID	0.8262
10000	failed	0.9865	non-IID	0.6565
2000	failed	0.0981	non-IID	0.2093

Table 5.4: Entropy estimation using two internal ROs and extracting the least significant bits of the first differences of counter values.

K	AIS Test procedure B	AIS T8 Shannon entropy	NIST 800-90B IID	NIST 800-90B min-entropy
100000	passed	0.9999	IID	0.9948
30000	passed	0.9999	IID	0.9946
25000	passed	0.9996	IID	0.9957
20000	passed	0.9999	IID	0.9962
15000	passed	0.9999	IID	0.9960
10000	failed	0.9966	non-IID	0.7208
2000	failed	0.0999	non-IID	0.1907

Table 5.5: Entropy estimation using two internal STRs and extracting the least significant bits of counter values.

The entropy estimates are much more consistent when using counter values than when the sampling method of randomness extraction was used. The statistical tests were not able to distinguish any difference between random numbers produced from raw counter values and produced from counter differences. The oscillator type had no effect on the statistical test results either.

K	AIS Test procedure B	AIS T8 Shannon entropy	NIST 800-90B IID	NIST 800-90B min-entropy
100000	passed	0.9999	IID	0.9937
30000	passed	0.9999	IID	0.9959
25000	passed	0.9998	IID	0.9959
20000	passed	0.9999	IID	0.9954
15000	passed	0.9999	IID	0.9948
10000	failed	0.9979	non-IID	0.6607
2000	failed	0.0997	non-IID	0.2033

Table 5.6: Entropy estimation using two internal STRs and extracting the least significant bits of the first differences of counter values.

We can see that when entropy is extracted by sampling the jittered clock signal, the generator output does not pass any standard statistical test even when the entropy accumulation period K was set to 100 000 periods of s_2. On the other hand, counting the jittered periods produced random numbers of sufficient quality to pass both AIS-20/31 procedure B and NIST 800-90B IID track already with $K = 20\ 000$ periods of s_2. Moreover, the counter values obtained by the second method can be used to characterize the jitter, as we will show in the following sections.

5.2 Variance measurement as a basis for embedded testing

Random fluctuations of clock signals are characterized by the power law spectrum, which also corresponds to the probability distribution function of the random process, defined by the Eq. (5.1) [50].

$$S_y(f) = h_\alpha f^\alpha \tag{5.1}$$

where

— y is the fractional frequency,

— α is a constant characterizing the noise process,

— h_α is the intensity of the noise.

The power law spectrum depends on the noise type causing the random fluctuations. There are several noise types affecting the clock signal:

— $\alpha = -2$ Random walk frequency noise (RWF)

— $\alpha = -1$ Flicker frequency noise (FF)

— $\alpha = 0$ White frequency noise (RWF) or random walk phase noise (RWP)

— $\alpha = 1$ Flicker phase noise (FP)

— $\alpha = 2$ White phase noise (WP)

If y is a zero mean stationary process, it can be characterized by its variance. Variance of such a process, based on its power law spectrum, is defined by Eq. (5.2).

$$\sigma_y^2(\tau) = \int_0^{+\infty} S_y(f) \times |H_\tau(f)|^2 df, \tag{5.2}$$

where H_τ is the transfer function of the variance operator, which depends on the type of variance used.

We will focus on two variance types: statistical variance and Allan variance.

5.2.1 Statistical variance

Statistical variance is the most widely used type of variance. Variance can be computed as a convolution of the signal and the variance computation window of size τ. Figure 5.2 shows the computation window of the statistical variance.

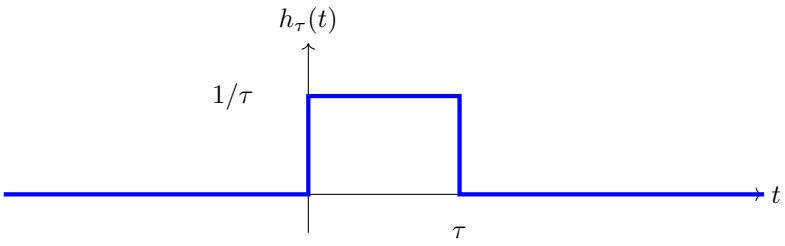


Figure 5.2: Convolution computation window of the statistical variance

Counter variances at the output of the TRNG represent frequency samples of the signal, so we need to use frequency representation of the convolution operation. The frequency representation of this variance computation is then defined by Eq. (5.3).

$$|H_\tau(f)|^2 = \left(\frac{\sin(\pi\tau f)}{\pi\tau f}\right)^2 \tag{5.3}$$

The variance of random fluctuations based on Eq. (5.2) is defined in Eq. (5.4) for the statistical variance.

$$\sigma_y^2(\tau) = \sum_{\alpha=-2}^{2} \frac{h_\alpha}{(\pi\tau)^2} \int_0^{f_h} f^{\alpha-2} \sin^2(\pi\tau f) df. \tag{5.4}$$

The statistical variance does not converge for $\alpha \leq -1$ as f tends to 0. This means that the statistical variance provides inaccurate estimates in the presence of flicker frequency noise and other low frequency noises.

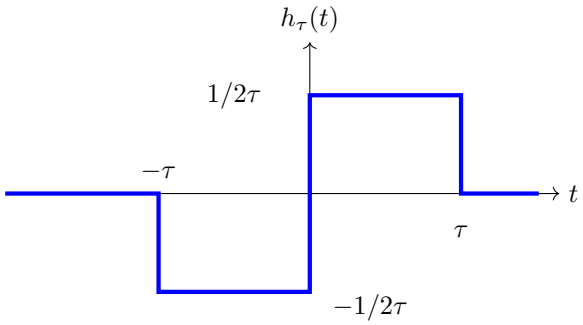


Figure 5.3: Allan variance convolution window

5.2.2 Allan variance

Figure 5.3 shows the convolution window of the Allan variance.

Its frequency representation is defined by Eq. (5.5).

$$|H_\tau(f)|^2 = \left(\frac{\sin(\pi\tau f)}{\pi\tau f}\right)^2 \sin^2(\pi\tau f) \tag{5.5}$$

So the Allan variance of random fluctuations is defined by Eq. (5.6).

$$\sigma_y^2(\tau) \;=\; \sum_{\alpha=-2}^{2} \frac{2h_\alpha}{(\pi\tau)^2} \int_0^{f_h} f^{\alpha-2} \sin^4(\pi\tau f) df \tag{5.6}$$

Allan variance converges for $\alpha > -3$ as f tends to 0, which means that Allan variance provides correct jitter size estimate even in presence of low frequency noises.

If we want to use Allan variance to monitor jitter continuously inside the logic device, we need to estimate Allan variance from a limited set of data instead of computing it from the infinite random process. Such an estimate is defined in Eq. (5.7).

$$\sigma_y^2(\tau) \;=\; \frac{1}{2(M-1)} \sum_{i=1}^{M-1} \left(\overline{y}_{i+1} - \overline{y}_i\right)^2. \tag{5.7}$$

where

— y_i is the average frequency fluctuation over a limited time interval τ. This value corresponds to the counter values when counting the jittered periods over the time τ.

— M is the total number of counter values from which the variance is estimated.

5.2.3 Hardware implementation of variance measurements

We implemented the circuit from Figure 5.1(b) in order to practically study the differences between statistical and Allan variance. We compared four different hardware configurations implemented in Intel Cyclone V FPGA:

— **Configuration 1**: Signal s_1 of 127 MHz was generated in an RO and signal s_2 came from a low jitter quartz oscillator generating a stable 125 MHz clock.

— **Configuration 2**: Both signals (s_1 and s_2) were generated in two ROs with the same number of elements, oscillating at a frequency of 125 and 127 MHz, respectively.

— **Configuration 3**: Signal s_1 of 128 MHz was generated in an STR and signal s_2 came from a low jitter quartz oscillator generating a stable 125 MHz clock.

— **Configuration 4**: Both signals (s_1 and s_2) were generated in two STRs with the same number of elements and oscillating at a frequency of 130 and 128 MHz, respectively.

The counter values were sent to a PC and evaluated in software. We implemented the whole project using HECTOR evaluation platform [3]. The variance measurement will only give meaningful results when set up correctly. The two crucial parameters are:

— K – the number of periods of s_2 during which we will count the periods of s_1. This number defines the accumulation period τ.

— M – the number of samples from which the variance will be computed.

We started by finding the right M. For this study, we fixed $K = 30000$ and performed several variance measurements. Figure 5.4 shows the measurement results.

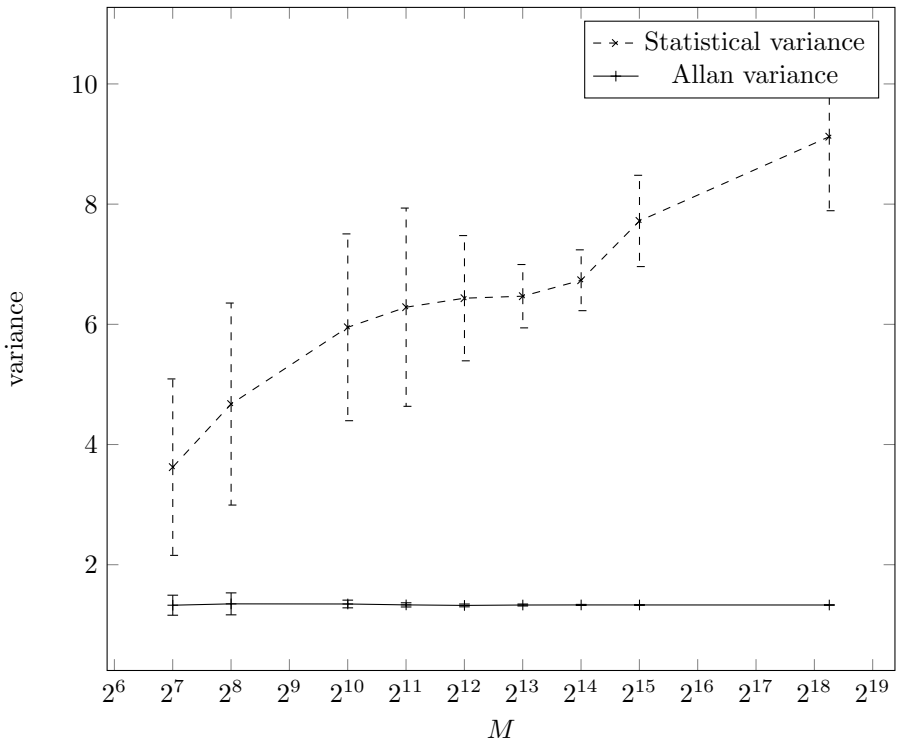


Figure 5.4: Variance measurement results dependence on the parameter M

Statistical variance increases greatly with increasing M while Allan variance changes only slightly. This effect occurs because the contribution of low frequency noises increases with in-

creasing M. We chose 4096 (2^{12}) to use as M for our implementation of embedded variance measurement and for all subsequent experiments. This value is a compromise between the number of statistical data to compute variance from and the computation speed. In order to do a fair comparison of the two variance computation methods we set M to the same value for statistical and Allan variance.

The second parameter to set is the accumulation time τ, which is set by the constant K. This constant represents the number of periods of s_2 during which we count the periods of s_1. Figures 5.5 and 5.6 show the results of variance study with K ranging from 300 to several million. Two ring oscillators were used as clock sources in the case shown in Figure 5.5 and two self timed rings were used in the case shown in Figure 5.6.

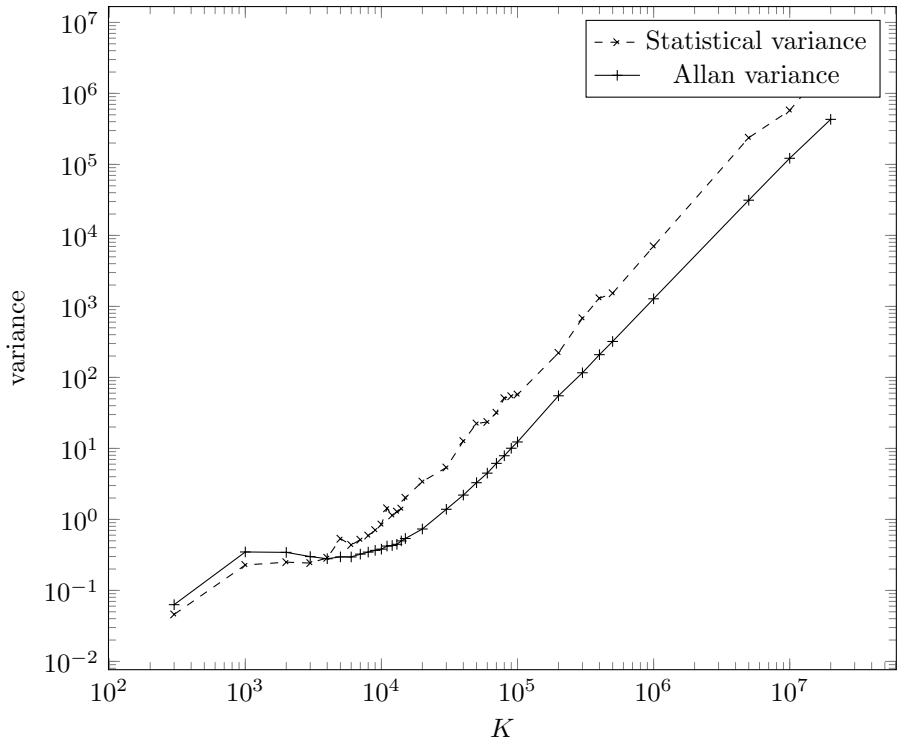


Figure 5.5: Variance of counter values depending on K with two ROs as a source

We can see that the variance increases with the accumulation time τ in very similar way for both ROs and STRs. This means that jitter accumulates similarly in both oscillatory structures.

It is also noticeable that for low values of K (below 1000), the variance is impacted by the quantization noise since for low accumulation periods the quantization noise is stronger. So in order to obtain meaningful results, k should be larger than 10 000, which is the value where both curves begin to be more distinct.

Last but not least, we can observe that Allan variance is almost always lower than the statistical variance. This fact proves that the statistical variance overestimates jitter size compared

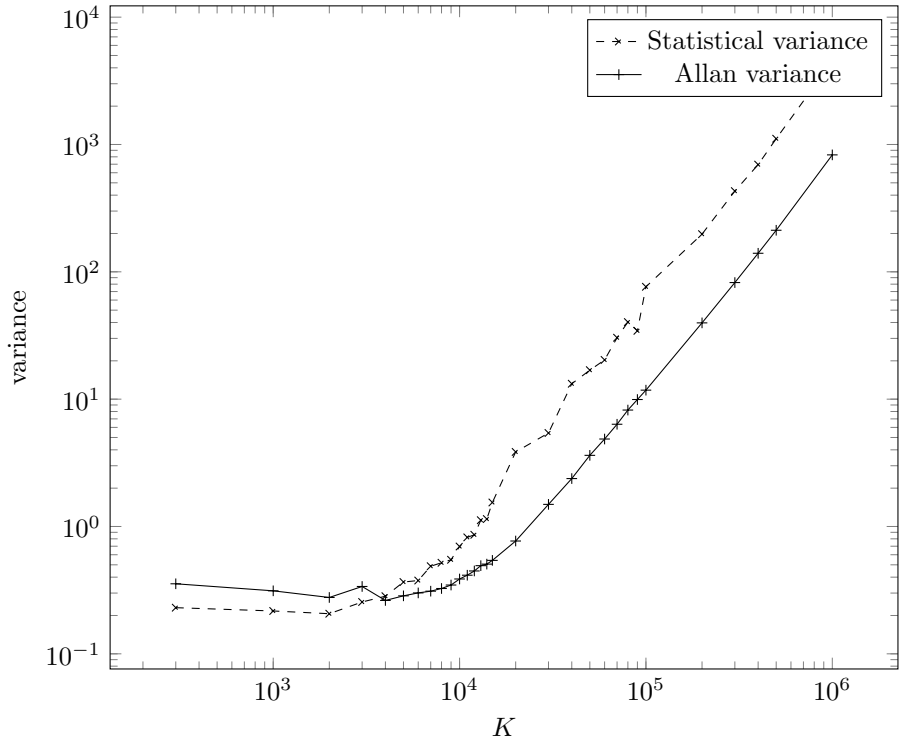


Figure 5.6: Variance of counter values depending on K with two STRs as a source

to Allan variance.

Findings from this study can help us design embedded variance measurement since the design parameters depend on the signal behavior studied. We implemented variance measurement based on works of Haddad et al. [49] and Fischer et al. [22] as well as Allan variance measurement based on Eq. (5.7). Figure 5.7 shows the variance measurement implementation based on the paper of Haddad et al. [49]. Figure 5.8 shows Allan variance measurement implementation. All circuits were implemented in Intel Cyclone V FPGA and they use fixed point arithmetic.

In Figure 5.7 we can see that to compute statistical variance we need one accumulator, one multiplier, one subtractor and one multiply and accumulate operation (MAC). It is also noticeable that this circuitry needs to process rather large numbers because there are 12-bit counter values at the input. These 12-bit values are accumulated in the accumulator, which makes them 24 bit wide. Then they are squared, which again doubles their size to 48 bits.

Allan variance, on the other hand, requires only one subtractor and one MAC. Since Allan variance processes differences of counter values, it needs to process much smaller numbers. Even after squaring and accumulation these numbers are only 16-bit wide. This drastically reduces the area footprint of the measurement circuitry and it should also allow the circuit to run at higher frequencies.

To put these assumptions to the test we implemented the variance measurement methods and

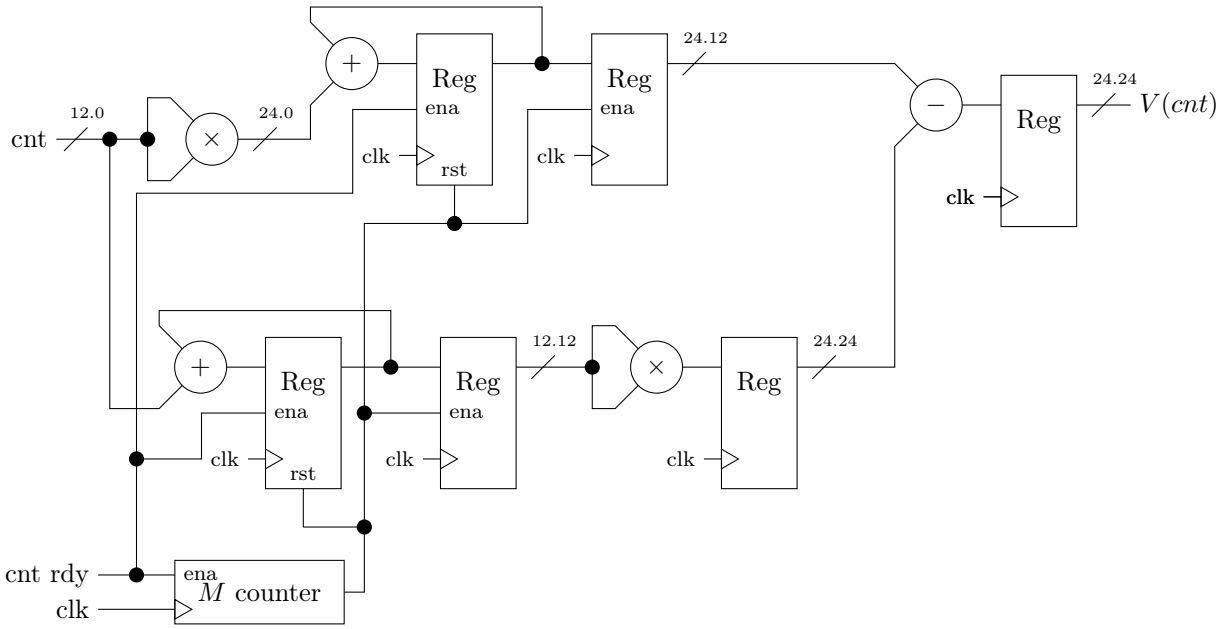


Figure 5.7: Statistical variance measurement circuitry

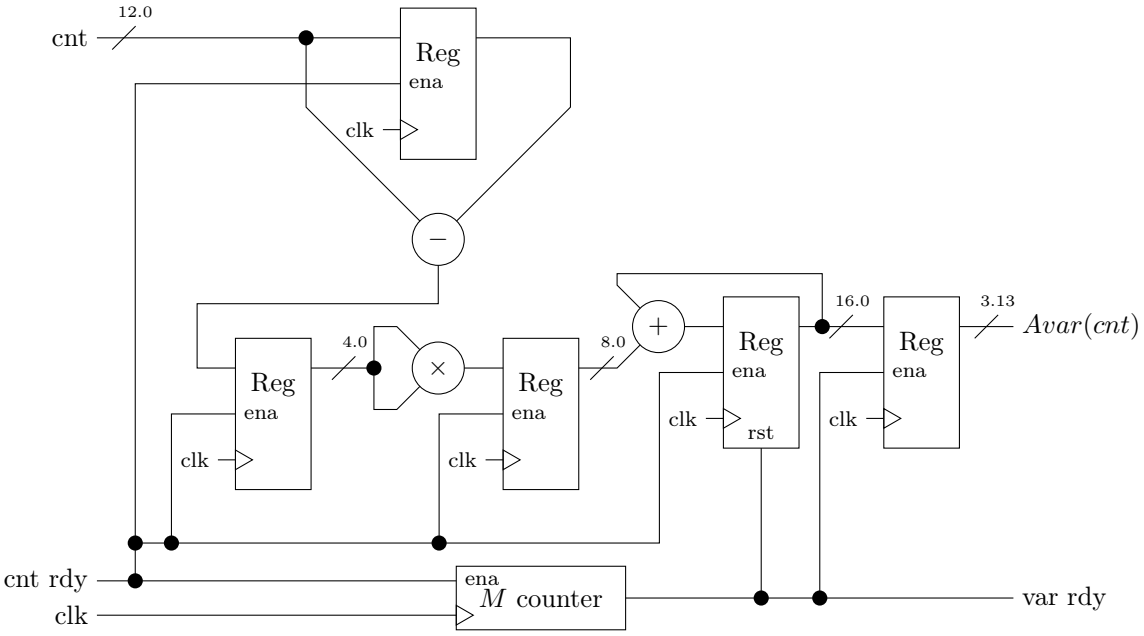


Figure 5.8: Allan variance measurement circuitry

compared their size, estimated maximum frequency and power consumption. Area requirements and maximum estimated operating frequency are taken from the compilation report of the Quartus software. Power consumption was measured physically on HECTOR evaluation platform [3]. Table 5.7 summarizes the implementation results.

We can confirm that Allan variance measurement is the smallest and fastest among the

Method	Area		f_{max}	Power
	ALM/Regs	DSPs	[MHz]	[mW]
Statistical variance [49]	119/160	2	178.3	6-7
Embedded tests from [22]	169/200	4	187.7	7-8
Allan variance based on Eq. (5.7)	49/117	1	238.5	4-5

Table 5.7: Implementation results of different variance measurement methods in Intel Cyclone V FPGA device 5CEBA4F17C8N

compared methods. It also consumes slightly less power than the other two methods.

In a real cryptographic system, there are many blocks implemented in a single FPGA. These blocks contain not only TRNG and its tests but also ciphers and various communication peripherals. We wished to evaluate how these other circuits affect the randomness source and variance measurement. In order to do that, we prepared three projects to rigorously study the impact of the embedded measurement itself on the source of randomness and the impact of the surrounding circuitry on the source as well as measurement.

— **Project 1** – Only two oscillators serving as a source of randomness were implemented. Outputs of both oscillators were observed using LVDS outputs of the device and differential oscilloscope probes. Figure 5.9 shows the block diagram of this project.

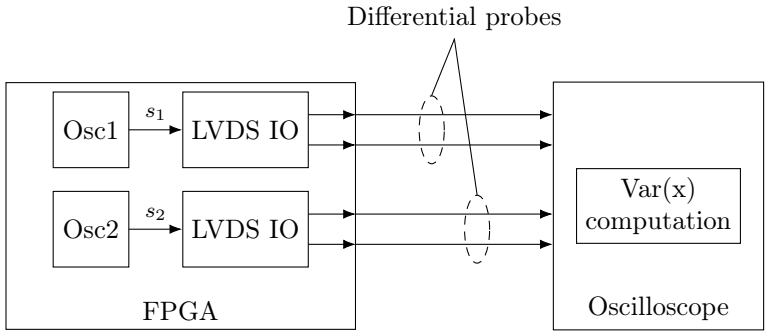


Figure 5.9: External jitter measurement using oscilloscope and differential probes

— **Project 2** – Besides the two oscillators, we implemented the embedded variance measurement, AES cipher and oscillator based TRNG to mimic the behavior of a real crypto SoC. Data from all the blocks were collected in the PC. Figure 5.10 presents this project.

— **Project 3** – Project 2 with only one internal oscillator and quartz oscillator used as the second clock source (see Figure 5.11).

Since STRs and ROs seem to generate jitter in very similar way (see Figures 5.5 and 5.6), we decided to implement only ROs for this study because they are easier to implement than STRs. To prevent changes in timing between different project compilations we used exported post-fit netlists in Exported Partition file (.qxp) of the Quartus software. This way, the ROs

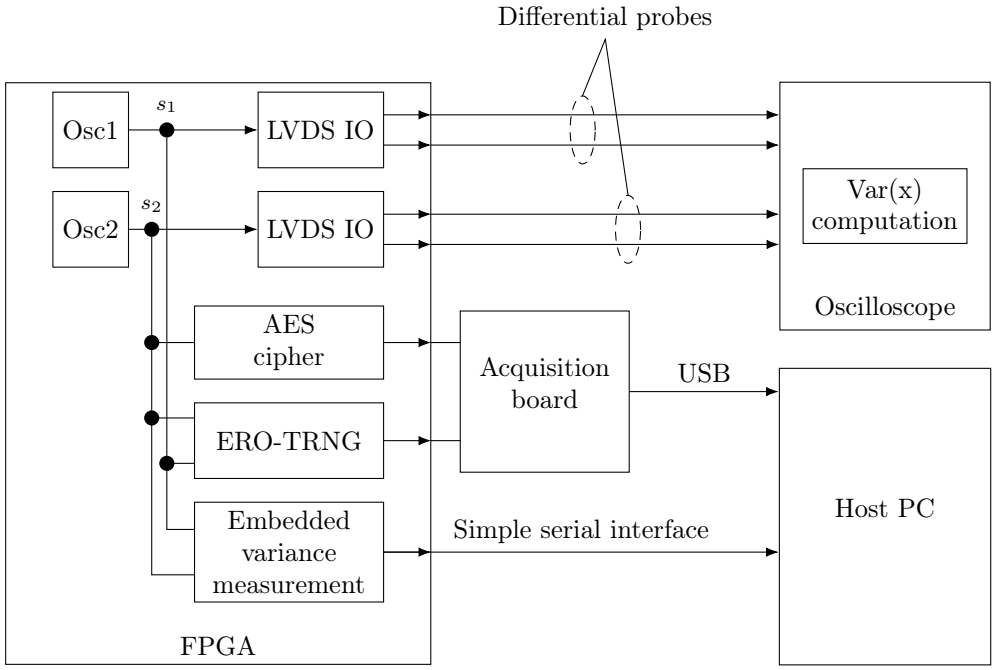


Figure 5.10: External jitter measurement using oscilloscope and differential probes together with internal variance measurement and other components of the cryptographic SoC

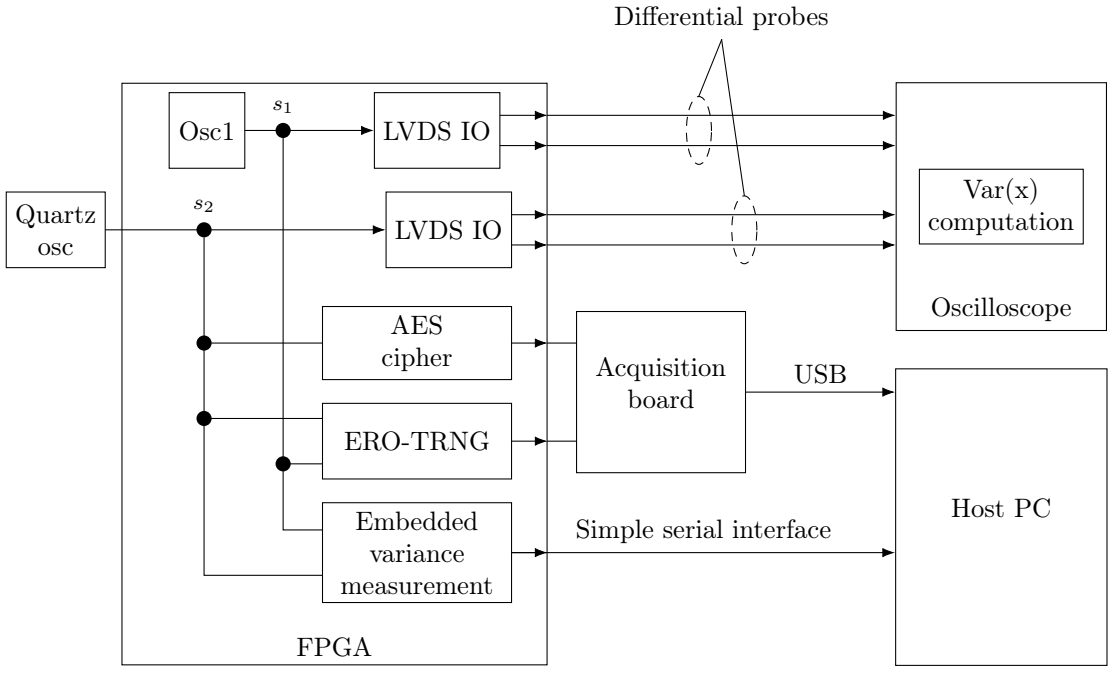


Figure 5.11: Crypto SoC with one internal and one external oscillator as source of randomness

were physically identical in all the projects. The output frequencies of the oscillators were 124.5 ± 0.3 MHz and 126.3 ± 0.2 MHz. So the overall difference between the output frequencies in the three projects was less than 1 %.

External measurements were performed using LeCroy wavePro 735i oscilloscope with 4 GHz bandwidth and sampling rate of 40 GS/s together with two D420 WaveLink 4 GHz differential probes. Counter values cannot be obtained directly from an oscilloscope since the value of k cannot be set up like in hardware – it can only be deduced from the oscilloscope time base, which, in our case, was set to 5 μs per division. We measured the number of periods of both clocks in this time interval. Finally, to make the comparison of values obtained by the external and embedded measurements more consistent, we measured the number of cycles of both clocks at the same time base interval and normalized the resulting data according to Eq. (5.8). n_1 represents the number of clock periods of s_1 and n_2 the number of clock periods of s_2 that appear during the same time interval determined by oscilloscope's time base.

$$cnt = \frac{n_1}{n_2} \cdot k, \tag{5.8}$$

We used $k = 30000$ to normalize oscilloscope measurements and to set the accumulation interval τ for the embedded variance measurement. Table 5.8 shows the results of this study.

Project	Osc1 jitter [ps]	Osc2 jitter [ps]	Normalized counter variance	Embedded variance	
				statistical	Allan
Project 1	3.4	4.3	15.92	N/A	N/A
Project 2	10.13	10.61	15.6	6.24	1.32
Project 3	6.57	9.84	125.44	7.66	1.58

Table 5.8: Impact of surrounding logic on the randomness source as well as on the embedded variance measurement

Embedded measurements used $M = 4096$ samples to compute variance from.

We can see that putting the whole cryptosystem into the FPGA more than doubles the measured jitter but it does not affect the normalized counter variance. It is also clearly visible that the jitter of the quartz oscillator is significantly smaller than the jitter of the RO but the normalized variance is increased drastically when external quartz oscillator is used.

The variance measured internally does not change much even when using quartz oscillator. We can also notice that internal measurement always gives smaller variance than the external measurement using an oscilloscope. This effect is expected since the external measurement is affected by the transmission properties of the FPGA IOs and oscilloscope probes.

Drastic change in normalized counter variance when using quartz oscillator is alarming. So we decided to have a closer look at what exactly is causing it.

We acquired raw counter values with accumulation period $k = 30000$ using quartz oscillator to generate s_2 and then using two identical ROs. The measurement took approximately 30 minutes. Figure 5.12 shows raw counter values acquired using quartz oscillator and Figure 5.13 shows counter values acquired using two internal ROs.

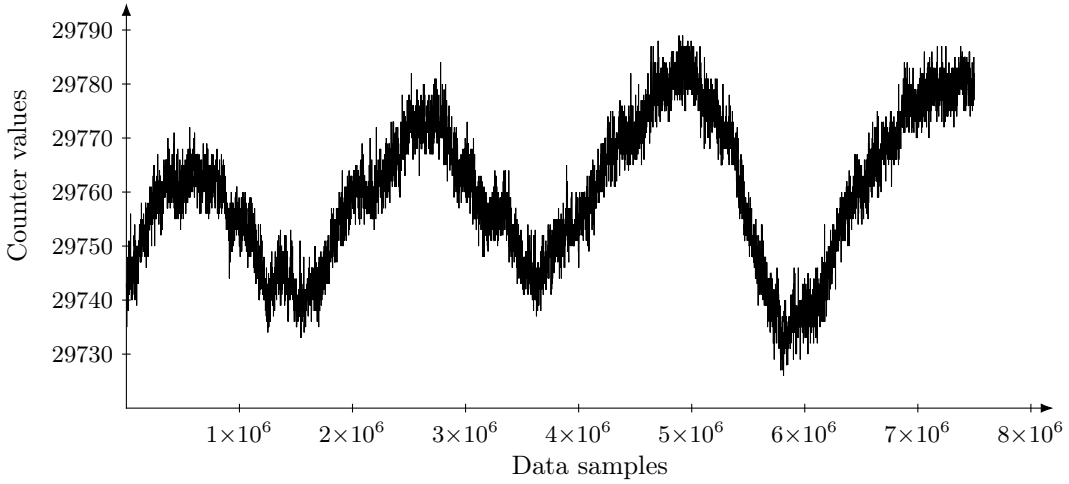


Figure 5.12: Counter values acquired using a quartz oscillator for s_2

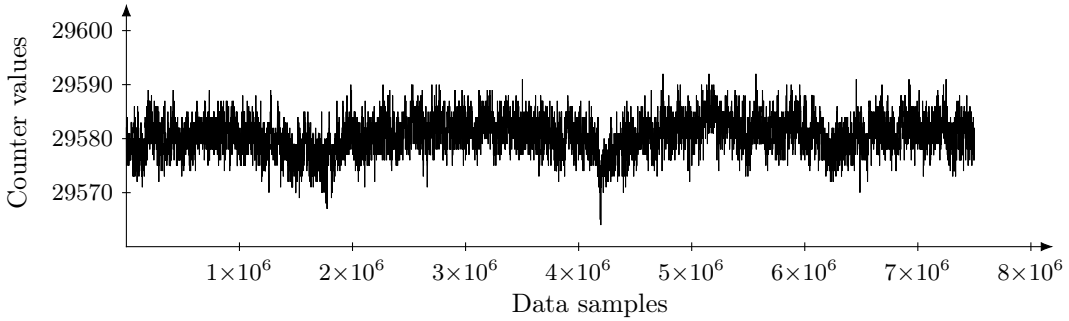


Figure 5.13: Counter values acquired using two identical ROs for s_1 and s_2

We can clearly see a low frequency signal affecting the counter values when using quartz oscillator. This signal has a frequency of approximately 1.5 mHz. When two internal ROs were used, this low frequency signal is still noticeable in the counter values but the amplitude of this signal is considerably reduced.

We found out that this low frequency signal comes from the power supply and it is observable in counter values even though the evaluation board was using low noise linear power supplies. It is possible that the low frequency signal we observed is specific to the power supply network used in the laboratory, where we conducted experiments. It may originate from some other equipment connected to the same supply grid. But this study confirms that such phenomena may occur and they are beyond the control of the TRNG designer. Furthermore, using two identical ring oscillators greatly reduces the amplitude of the unwanted signal.

A signal, such as one visible in Figure 5.12, is extremely hard to detect because of its low frequency. So the use of external clock sources should be completely avoided in TRNG design. Moreover, even when using internal clocks we should always use identical clock sources in order

to produce a differential design. This way, both oscillators are affected by unwanted global phenomena, such as ambient temperature or power supply noises, in the same way and their relative jitter should stay unaffected.

Dependence of the generated random numbers on the global deterministic signals should be hence effectively reduced by using two identical internal oscillators. But there still might be the dependence between the consecutive output random numbers. We performed an autocorrelation analysis of counter values and their first order differences in order to find out if applying first order difference to counter values can break their dependencies. Figures 5.14 and 5.15 show the autocorrelation of counter values and their differences when generated using one quartz and one ring oscillator and one quartz and one self timed ring.

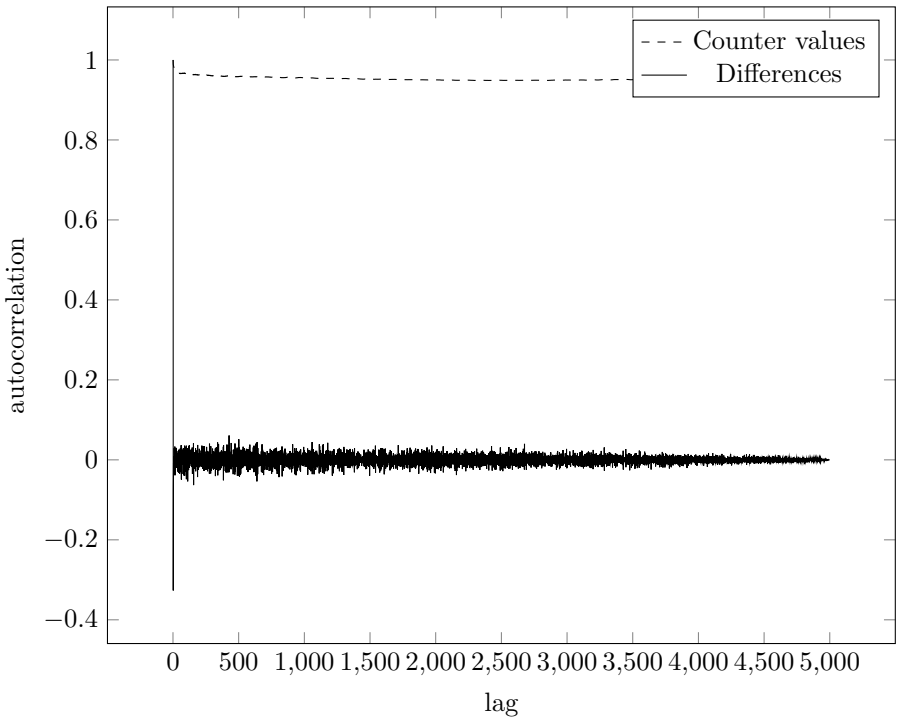


Figure 5.14: Autocorrelation function of counter values and their first order differences when generated by one RO and one external quartz oscillator

We can see that applying the first order difference to counter values can effectively break dependencies even when the signal is affected by a strong global deterministic noise. Since using quartz oscillator should be avoided, as we already established, it is much more interesting to analyze counter values generated using two identical internal oscillators. Figures 5.16 and 5.17 show the autocorrelation of counter values and their first order differences when generated using two identical internal oscillators.

The autocorrelation of raw counter values is much lower in this case than the autocorrelation of raw counter values generated by one quartz oscillator and one internal oscillator. This

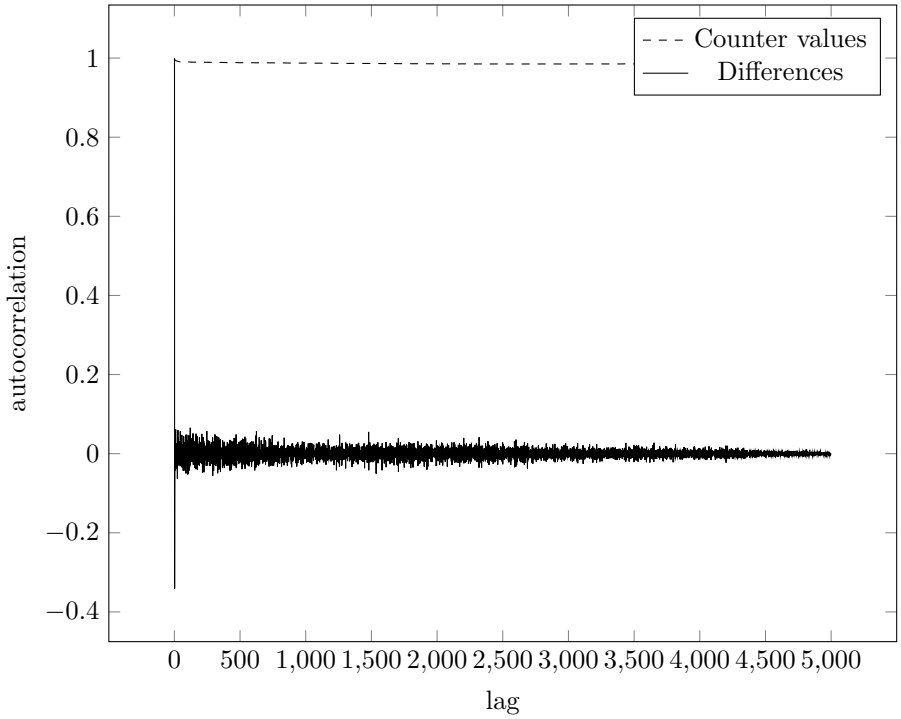


Figure 5.15: Autocorrelation function of counter values and their first order differences when generated by one STR and one external quartz oscillator

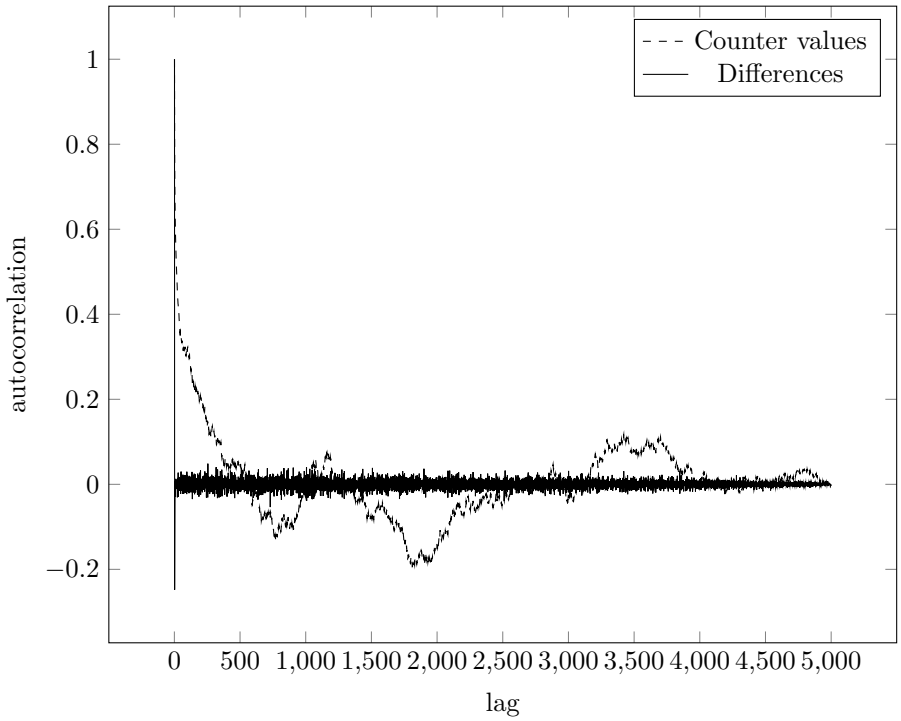


Figure 5.16: Autocorrelation function of counter values and their first order differences when generated by two identical ROs

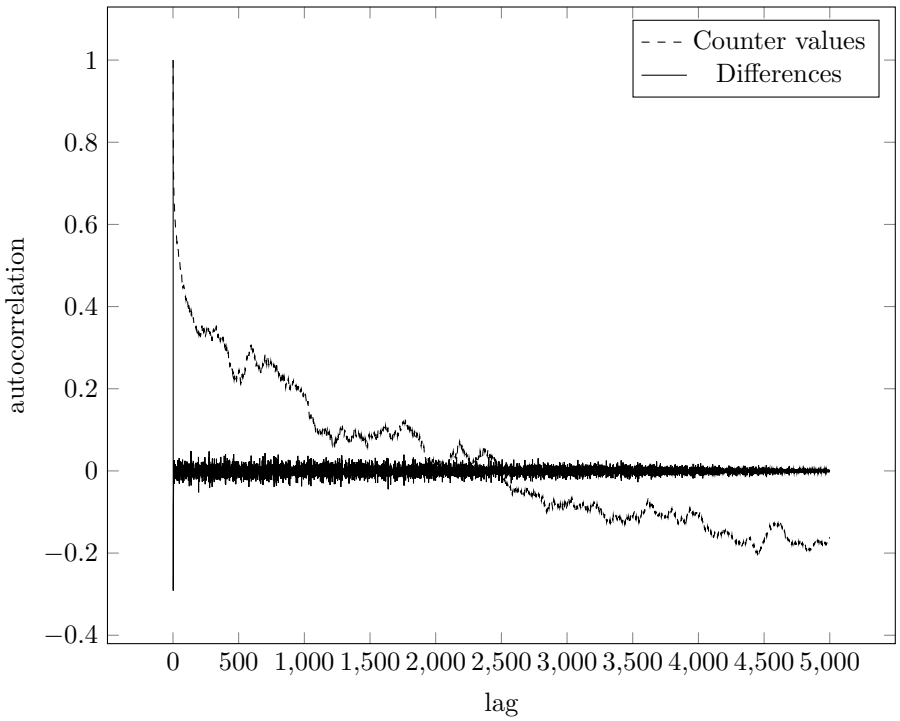


Figure 5.17: Autocorrelation function of counter values and their first order differences when generated by two identical STRs

suggests that the low frequency deterministic signal seen in Figure 5.12 introduces not only dependencies between counter values and global noises but also dependencies between counter values themselves. Using two internal oscillators minimizes the effect of global deterministic signals on generated counter values but it also helps to break dependencies between generated counter values as can be seen in Figures 5.16 and 5.17. Autocorrelation of first order differences clearly shows that the dependencies between generated counter values can be broken by applying the first order difference.

It is also noticeable that the autocorrelation of counter values and their first order differences is very similar when generated using ROs and STRs. This even further confirms the claims from preceding sections that jitter behaves similarly in both oscillatory structures.

All of the studies, we conducted, suggest that the best random numbers should be produced from a TRNG based on counting of jittered periods as the randomness extraction method, two identical internal oscillators as a source of randomness and the least significant bit of counter value differences should be taken as random bit. The type of the oscillator used should not affect the statistical quality of generated random numbers.

5.3 Conclusion

In this chapter we evaluated free running oscillators as sources of randomness. We studied how randomness is produced in oscillatory structures, how we can extract it in order to produce random numbers and what embedded test principles can be used to monitor such source of randomness.

Two methods of randomness extraction from the clock jitter were evaluated: sampling of jittered clock and counting of jittered clock periods. Sampling the jittered clock is the most commonly used randomness extraction method in TRNG design. But random numbers extracted this way are biased based on the duty cycle of the sampled clock. Also, the jitter must accumulate for a long time (hundreds of thousands of periods of sampling clock), which does not allow for high output bit rates.

Counting the jittered clock periods, on the other hand, mitigates the effect of biased duty cycle on the generated random numbers and it requires much lower jitter accumulation times. So this extraction method produces random numbers of higher quality at higher output bit rate.

Randomness produced in the source was evaluated and characterized using the variance of counter values. We compared two kinds of variance in terms of their suitability to characterize the clock jitter produced in free running oscillators: statistical variance and Allan variance. Our studies show that the statistical variance is not suitable for jitter characterization since it overestimates the jitter especially when the source of randomness is affected by low frequency noises. Allan variance estimates the jitter size accurately even in the presence of low frequency noises and hence it is the recommended method of embedded jitter monitoring. We also compared the two variance measurement methods in terms of the implementation parameters such as their area footprint, maximum operating frequency and power consumption. Allan variance measurement is superior to the statistical variance measurement in all of the tested categories.

Then, we used the variance measurement to characterize the jitter produced in two oscillator types: ring oscillators and self-timed rings. Both oscillatory structures are suitable for implementation in digital logic devices (FPGAs and ASICs). The results of our studies show that the jitter accumulates in a similar manner in both oscillators. So the oscillator type does not affect the quality of produced random numbers.

We also tried to use low noise external quartz oscillator to generate one of the clocks for the TRNG. But strong deterministic signals with long period were observed when an external oscillator was used. Further study of this phenomenon showed that it originates in the power supply and it propagated to the generated random numbers even when the evaluation board was using low noise linear power supplies. These findings confirm that global deterministic noises are unavoidable and unpredictable. The only thing a TRNG designer can do to protect the design

from such negative effects is to use two identical internal oscillators to generate both clocks.

Work presented in this chapter was published in:

[51] E. Noumon Allini, M. Skórski, O. Petura, F. Bernard, M. Laban, and V. Fischer, "Evaluation and monitoring of free running oscillators serving as source of randomness," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2018, pp. 214–242, Aug. 2018

Résumé

Dans ce chapitre, nous avons évalué les oscillateurs libres en tant que sources d'aléa. Nous avons étudié la manière dont l'aléa est produite dans les structures oscillatoires, comment l'extraire pour produire des nombres aléatoires et quels principes de tests intégrés peuvent être utilisés pour suivre l'évolution de cette source d'aléa.

Deux méthodes d'extraction d'aléa du jitter d'horloge ont été évaluées : l'échantillonnage de l'horloge affectée par le jitter et le comptage des périodes d'horloge affectée par le jitter. L'échantillonnage de l'horloge affectée par le jitter est la méthode d'extraction d'aléa la plus couramment utilisée dans la conception des TRNGs. Mais les nombres aléatoires extraits de cette manière sont biaisés en fonction du rapport cyclique de l'horloge échantillonnée. En outre, le jitter doit s'accumuler pendant longtemps (des centaines de milliers de périodes d'horloge d'échantillonnage), ce qui ne permet pas des débits élevés à la sortie du TRNG.

En revanche, le comptage des périodes d'horloge affectée par le jitter supprime l'effet du rapport cyclique biaisé sur les nombres aléatoires générés et nécessite des temps d'accumulation du jitter beaucoup plus faibles. Cette méthode d'extraction produit donc des nombres aléatoires de qualité supérieure à un débit plus élevé.

L'aléa produit dans la source a été évalué et caractérisé à l'aide de la variance des valeurs de compteur. Nous avons comparé deux types de variances en termes d'aptitude à caractériser le jitter d'horloge produit par les oscillateurs libres : la variance statistique et la variance d'Allan. Nos études montrent que la variance statistique ne convient pas à la caractérisation du jitter, car elle surestime celui-ci, en particulier lorsque la source d'aléa est affectée par des bruits de basse fréquence. La variance d'Allan estime la taille du jitter avec précision, même en présence de bruits de basse fréquence. C'est donc la méthode recommandée pour la surveillance du jitter. Nous avons également comparé les deux méthodes de mesure de la variance en termes de paramètres d'implémentation, tels que la surface requise, la fréquence de fonctionnement maximale et la consommation électrique. La méthode utilisant la variance d'Allan est plus efficace que celle utilisant la variance statistique dans toutes les catégories testées.

Nous avons ensuite utilisé la mesure de la variance pour caractériser le jitter produit par deux types d'oscillateurs : les oscillateurs en anneau et les oscillateurs à anneaux auto séquencées. Les deux structures oscillatoires sont adaptées à une mise en œuvre dans des circuits logiques numériques (FPGAs et ASICs). Les résultats de nos études montrent que le jitter s'accumule de manière similaire dans les deux types oscillateurs. Le type d'oscillateur n'affecte donc pas la qualité des nombres aléatoires produits.

Nous avons également essayé d'utiliser un oscillateur à quartz externe à faible bruit pour générer l'une des horloges du TRNG. Mais des signaux déterministes forts avec une longue

période ont été observés lorsqu'un oscillateur externe était utilisé. Une étude plus approfondie de ce phénomène a montré qu'il provenait de l'alimentation et s'était propagé aux nombres aléatoires générés même lorsque la carte d'évaluation utilisait des alimentations linéaires à faible bruit. Ces résultats confirment que les bruits déterministes globaux sont inévitables et imprévisibles. La seule chose qu'un concepteur de TRNG puisse faire pour protéger le design contre de tels effets négatifs est d'utiliser deux oscillateurs internes identiques pour générer les deux horloges.

Les travaux présentés dans ce chapitre ont été publiés dans :

[51] E. Noumon Allini, M. Skórski, O. Petura, F. Bernard, M. Laban, and V. Fischer, "Evaluation and monitoring of free running oscillators serving as source of randomness," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2018, pp. 214–242, Aug. 2018

Conclusion

In this thesis, we presented fundamental methods of generation of true random numbers in logic devices (FPGAs and ASICs): clock jitter, metastability, chaos and analog signals. We concluded that clock jitter is the most commonly used and most suitable to use for TRNGs implemented in logic devices. We then discussed various types of TRNGs using clock jitter as a source of randomness, fundamentals of embedded tests of TRNGs and standards for TRNG design and evaluation. Based on HECTOR project requirements, we chose to focus on PTG.2 class of TRNGs according to the AIS-20/31 standard [25].

We then proceeded with the rigorous evaluation of different AIS-20/31 compliant TRNG cores implemented in three different FPGA families: Intel Cyclone V, Xilinx Spartan-6 and Microsemi SmartFusion2. The TRNG cores were evaluated in terms of the required area (logic blocks and flip flops), power consumption, output bit rate, power/energy efficiency and entropy of generated random numbers.

On top of these parameters, we proposed two new metrics, which are crucial for TRNG design: entropy * bit rate product and feasibility and repeatability of the design. Since all of the compared TRNGs were implemented in the three FPGA families, the evaluation could be done objectively and the most versatile TRNGs could be selected for further study. Based on the results of the evaluation, we selected ERO-TRNG, PLL-TRNG and STR-TRNG for further testing on ASICs.

Two HECTOR ASICs were designed using ST CMOS 65nm manufacturing technology. The first ASIC contained only one TRNG: PLL-TRNG. This TRNG took already half of the circuit area because PLLs are huge and the TRNG with two PLLs occupied 0.25 mm^2. The other half of the circuit contained ASIC control logic, temperature compensation blocks and TERO test modules aimed at PUF research. The second HECTOR ASIC contained four STR-TRNGs and one ERO-TRNG.

We could not perform a formal verification of the complete circuits since our tools did not allow it. Consequently, the ASICs were only partially functional: PLL-TRNG was working but we were unable to observe PLL outputs using an oscilloscope, only one STR-TRNG was working and ERO-TRNG revealed a design flaw during its evaluation. Despite the problems created by

lacking tools, we were able to confirm that the TRNGs selected for ASIC implementation are indeed feasible and they can produce good quality random numbers.

In the next step, we analyzed the PLL-TRNG design in detail. Even though the PLL-TRNG is not the easiest to implement based on the evaluation made in Chapter 2, it is very well repeatable since once a suitable configuration of the PLLs is found, it will work on all the devices with the same PLLs. Evaluation of HECTOR ASICs underlines the potential of the PLL-TRNG as a robust and versatile TRNG when it comes to implementation on different technologies because it is the only one, which operates correctly and without any issues both on FPGAs and on HECTOR ASICs.

As we already mentioned, the PLL-TRNG works very well once a suitable PLL configuration is found. But PLLs are complex circuits with many configuration possibilities and on top of that, all of the PLL parameters have their physical limits, which we must respect in order to produce a functional PLL-TRNG design.

The search for a suitable PLL configuration for the given technology (FPGA family or PLL IP in selected ASIC technology) is a tedious task and it is almost impossible to do this manually. So we explored several methods of automatic search for suitable PLL configurations.

We began with genetic algorithms, which mimic the natural selection process in order to find a locally optimal solution for a given problem. This approach worked quite well for a single PLL variant of the PLL-TRNG but the genetic algorithm was unable to find suitable configuration for the two PLL variant of the PLL-TRNG. This led us to explore a way to search for a globally optimal solution.

Based on the physical limits of the PLL parameters and their specific mutual relations within a PLL-TRNG design, we were able to design an optimized search algorithm, which limits the search space only to those PLL configurations, which are physically feasible for the PLL-TRNG implementation. The algorithm then explores this limited space and searches for configurations, which provide high entropy random numbers. We designed a PLL-TRNG using this algorithm and we tested its output using AIS-20/31 and NIST 800-90B standard statistical tests.

We also performed an extensive testing of the PLL-TRNG implemented in Intel Cyclone V FPGA in wide temperature and supply voltage range. This testing confirmed that the PLL-TRNG is capable of producing high quality random numbers in wide range of operating conditions. It also confirmed that the optimized exhaustive search algorithm is able to find a PLL configuration, which withstands all kinds of conditions.

Finally, we studied oscillator based TRNGs. At first, we compared two methods of randomness extraction from the clock jitter in terms of their efficiency: sampling the jittered clock signal and counting the jittered clock periods. Sampling of the jittered clock signal is the most common method of randomness extraction from the clock jitter in literature, but its biggest drawback

is that the bias of generated random numbers depends on the duty cycle of the sampled clock signal.

Counting the jittered clock periods effectively reduces this dependency and it allows to reduce the jitter accumulation time by an order of magnitude compared to the sampling method of randomness extraction.

What is more, using a counter provides a good basis for embedded testing of generated numbers since counter values hold more information than just 1-bit values at the output of the sampler. The variance of counter values is directly related to the size of the jitter of the clock signal produced by an oscillator.

By measuring the variance of counter values we then evaluated the behavior of the jitter produced by two types of oscillators: ring oscillators and self timed rings. We measured accumulated jitter in ring oscillators and self timed rings with different jitter accumulation periods and compared how the jitter accumulates in each of the two types of oscillators. The comparative study showed that the jitter accumulates in a similar manner in both oscillator types.

During this study we also compared two kinds of variance estimation: statistical variance and Allan variance. The study experimentally confirmed that statistical variance overestimates the jitter and further evaluation of embedded variance measurements showed that Allan variance is even more suitable for hardware implementation as it requires smaller area and its computation is faster than computation of statistical variance.

To finalize the study of oscillator based TRNGs, we evaluated their susceptibility to internal and external interference and we looked for methods of mitigating such interference. We compared the results of embedded variance measurement when nothing but this measurement and a TRNG was implemented in the Intel Cyclone V FPGA to the results of the same measurement when a more complex circuit with an AES cipher was implemented in the same FPGA.

The comparison showed that even though the jitter of both oscillators in the TRNG increased when an AES cipher was running alongside it, the relative jitter between the two oscillators represented by the variance of counter values did not change. This means that the variance of counter values does not overestimate the jitter in the complex system, where internal sources of interference are present.

The external interference was evaluated by replacing one of the oscillators in the TRNG with an external low noise quartz oscillator. The analysis of the counter values obtained from such a modified TRNG revealed the presence of strong external low frequency signals. We managed to match these signals to power supply variations, which were present despite the low noise linear power supplies used on the evaluation boards. We concluded that such an external interference is inevitable and the only way to avoid it is to use two identically implemented internal oscillators.

List of publications

Work presented in this thesis was published in scientific journal, international conferences and deliverables of the HECTOR project.

Scientific journal publication:

[51] E. Noumon Allini, M. Skórski, O. Petura, F. Bernard, M. Laban, and V. Fischer, "Evaluation and monitoring of free running oscillators serving as source of randomness," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2018, pp. 214–242, Aug. 2018

International conference publications:

[33] O. Petura, U. Mureddu, N. Bochard, V. Fischer, and L. Bossuet, "A survey of AIS-20/31 compliant TRNG cores suitable for FPGA devices," in *26th International Conference on Field-Programmable Logic and Applications, FPL '16, Lausanne, Switzerland*, Aug. 2016

[36] O. Petura, U. Mureddu, N. Bochard, V. Fischer, and L. Bossuet, "A survey of AIS-20/31 compliant TRNG cores implemented on FPGAs," in *TRUDEVICE – 6th Conference on Trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE 2016), Barcelona, Spain*, Nov. 2016

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

[44] E. N. Allini, O. Petura, V. Fischer, and F. Bernard, "Optimization of the PLL configuration in a pll-based TRNG design," in *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018*, pp. 1265–1270, 2018

[46] J. Balasch, F. Bernard, V. Fischer, M. Grujić, M. Laban, O. Petura, V. Rožić, G. van Battum, I. Verbauwhede, M. Wakker, and B. Yang, "Design and testing methodologies for true random number generators towards industry certification," in *2018 IEEE 23rd European Test Symposium (ETS)*, May 2018

HECTOR deliverables:

[37] M. Deutschmann, S. Lattacher, J. Delvaux, V. Rozic, B. Yang, D. Singelee, L. Bossuet, V. Fischer, U. Mureddu, O. Petura, A. A. Yamajako, B. Kasser, and G. BATTUM, "HECTOR deliverable D2.1 – report on selected TRNG and PUF principles," Feb. 2016

[43] G. Battum, S. Lattacher, M. Deutschmann, B. Kasser, M. Agoyan, J. Nicolai, M. Madau, R. Sussella, J. Balasch, M. Grujic, V. Fischer, O. Petura, M. Laban, J. Luhman, M. Wakker, and R. Malafre, "HECTOR deliverable D2.4 – robustness tests on TRNGs and PUFs," July 2018

Contributions

— We proposed a methodology for fair evaluation of different TRNG cores and demonstrated its efficiency on several TRNG designs implemented in different FPGA families. Thanks to this approach, we could select suitable TRNG cores to implement in ASICs.

— We implemented selected TRNG cores in ASICs and evaluated their feasibility and performance.

— We closely studied PLL-TRNG design and we proposed automated tools for the PLL-TRNG design, which enable rapid development of high quality TRNGs within different technological constraints.

— We clearly demonstrated that oscillator based TRNGs must be designed using a differential principle: using two identically implemented internal oscillators.

— We showed that extracting randomness from the clock jitter by counting the jittered clock periods is much more effective than sampling the jittered clock edges. This method of randomness extraction allows to design TRNGs with higher bit rate and it provides a solid base for embedded tests, which can directly use counter values to characterize the jitter.

— We proposed to use Allan variance computed from counter values as a method of embedded testing of oscillator based TRNGs.

Conclusion

Dans cette thèse, nous avons présenté les méthodes fondamentales de génération de nombres véritablement aléatoires dans des circuits logiques (FPGAs et ASICs) : jitter d'horloge, métastabilité, chaos et signaux analogiques. Nous avons conclu que le jitter d'horloge est le plus couramment utilisé et le plus adapté aux TRNGs implémentés dans des circuits logiques. Nous avons ensuite discuté de divers types de TRNGs utilisant le jitter d'horloge comme source d'aléa, des principes de base des tests embarqués des TRNGs et des normes de conception et d'évaluation des TRNGs. Sur la base des exigences du projet HECTOR, nous avons choisi de nous concentrer sur les TRNGs de la classe PTG.2 conformément à la norme AIS-20/31 [25].

Nous avons ensuite procédé à l'évaluation rigoureuse de différents noyaux TRNG conformes à la norme AIS-20/31 implémentés dans trois familles de FPGA différentes : Intel Cyclone V, Xilinx Spartan-6 et Microsemi SmartFusion2. Les noyaux TRNG ont été évalués en fonction de la surface requise (blocs logiques et bascules), de la consommation d'énergie, du débit à la sortie, de l'efficacité de puissance/énergie et de l'entropie des nombres aléatoires générés.

En plus de ces paramètres, nous avons proposé deux nouvelles métriques, qui sont essentielles pour la conception de TRNG : le produit entropie $\times$ débit et la faisabilité et la répétabilité de la conception. Étant donné que tous les TRNGs comparées ont été mis en œuvre dans les trois familles de FPGA, l'évaluation peut être réalisée de manière objective et les TRNGs les plus polyvalents peuvent être sélectionnés pour une étude ultérieure. Sur la base des résultats de l'évaluation, nous avons sélectionné ERO-TRNG, PLL-TRNG et STR-TRNG pour des tests supplémentaires dans les ASICs.

Deux ASICs HECTOR ont été conçus à l'aide de la technologie de fabrication ST CMOS 65nm. Le premier ASIC ne contenait qu'un seul TRNG : PLL-TRNG. Ce TRNG occupait déjà la moitié de la surface du circuit car les PLL étaient énormes et le TRNG avec deux PLL prenait 0,25 mm^2. L'autre moitié du circuit contenait une logique de contrôle de l'ASIC, des blocs de compensation de température et des modules de test TERO pour la recherche sur les PUF. Le deuxième ASIC HECTOR contenait quatre STR-TRNG et un ERO-TRNG.

Nous n'avons pas pu effectuer de vérification formelle du circuit complet car nos outils ne le permettaient pas. Par conséquence, les ASICs n'étaient que partiellement fonctionnels: PLL-

TRNG fonctionnait mais nous ne pouvions pas observer les sorties de PLL à l'aide d'un oscilloscope, un seul STR-TRNG fonctionnait et ERO-TRNG révélait un défaut de conception lors de son évaluation. Malgré les problèmes posés par le manque d'outils, nous avons pu confirmer que les TRNGs sélectionnées pour l'implémentation sur un ASIC sont effectivement réalisables et qu'ils peuvent produire des nombres aléatoires de bonne qualité.

Ensuite, nous avons analysé en détail la conception du PLL-TRNG. Même si le PLL-TRNG n'est pas le plus simple à implémenter sur la base de l'évaluation faite au chapitre 2, il est très bien reproductible puisqu'une fois trouvée une configuration appropriée des PLL, il fonctionnera sur tous les appareils avec les mêmes PLLs. L'évaluation des ASIC HECTOR souligne le potentiel de la technologie PLL-TRNG en tant que solution TRNG robuste et polyvalente pour la mise en œuvre sur différentes technologies, car c'est la seule qui fonctionne correctement et sans problèmes, tant sur les FPGAs que sur les HECTOR ASICs.

Comme nous l'avons déjà mentionné, le PLL-TRNG fonctionne très bien une fois qu'une configuration PLL appropriée est trouvée. Mais les PLLs sont des circuits complexes avec de nombreuses possibilités de configuration. De plus, tous les paramètres de la PLL ont leurs limites physiques, qu'il faut respecter pour produire une conception PLL-TRNG fonctionnelle.

La recherche d'une configuration PLL appropriée sur une technologie donnée (famille FPGA ou type PLL sur ASIC) est une tâche fastidieuse et il est presque impossible de le faire manuellement. Nous avons donc exploré plusieurs méthodes de recherche automatique des configurations PLL appropriées.

Nous avons commencé par les algorithmes génétiques, qui imitent le processus de sélection naturelle afin de trouver une solution localement optimale pour un problème donné. Cette approche a plutôt bien fonctionné pour une variante de PLL-TRNG avec une seule PLL mais l'algorithme génétique n'a pas permis de trouver la configuration appropriée pour la variante du PLL-TRNG avec deux PLLs. Cela nous a amenés à rechercher un moyen de rechercher une solution globalement optimale.

Sur la base des limites physiques des paramètres de la PLL et de leurs relations mutuelles spécifiques au sein d'une conception PLL-TRNG, nous avons pu concevoir un algorithme de recherche optimisé, qui limite l'espace de recherche uniquement aux configurations de la PLL physiquement réalisables pour l'implémentation du PLL-TRNG. L'algorithme explore ensuite cet espace limité et recherche des configurations qui fournissent des nombres aléatoires à entropie élevée. Nous avons conçu un PLL-TRNG utilisant cet algorithme et avons testé sa sortie à l'aide de tests statistiques standards AIS-20/31 et NIST 800-90B.

Nous avons également effectué des tests approfondis sur le PLL-TRNG implémenté dans les circuits FPGA Intel Cyclone V dans une large plage de températures et de tensions d'alimentation. Ces tests ont confirmé que le PLL-TRNG est capable de produire des nombres aléatoires de

haute qualité dans une large gamme de conditions de fonctionnement. Il a également confirmé que l'algorithme de recherche exhaustive optimisée est capable de trouver une configuration PLL, qui résiste à toutes sortes de conditions.

Enfin, nous avons étudié les TRNG basés sur les oscillateurs. Au début, nous avons comparé deux méthodes d'extraction d'aléa du jitter d'horloge en termes d'efficacité : l'échantillonnage du signal d'horloge instable et le comptage des périodes de l'horloge instable. L'échantillonnage du signal d'horloge instable est la méthode la plus courante d'extraction d'aléa de l'instabilité d'horloge dans la littérature, mais son inconvénient majeur est que le biais des nombres aléatoires générés dépend du rapport cyclique du signal d'horloge échantillonné.

Le comptage des périodes d'horloge du jitter réduit efficacement cette dépendance et permet de réduire le temps d'accumulation de jitter d'un ordre de grandeur par rapport à la méthode d'échantillonnage pour l'extraction d'aléa.

De plus, l'utilisation d'un compteur constitue une bonne base pour le test embarqué des nombres générés, car les valeurs des compteurs contiennent davantage d'information que des valeurs à 1 bit à la sortie de l'échantillonneur. La variance des valeurs de compteur est directement liée au jitter du signal produit par un oscillateur.

En mesurant la variance des valeurs de compteur, nous avons évalué le comportement du jitter produit par deux types d'oscillateurs : les oscillateurs en anneau et les oscillateur en anneau auto séquencés. Nous avons mesuré le jitter accumulé dans l'oscillateur en anneau et dans l'oscillateur en anneau auto séquencé avec différentes périodes d'accumulation du jitter, et nous avons comparé l'accumulation du jitter dans chacun des deux oscillateurs. L'étude comparative a montré que le jitter s'accumule de manière similaire dans les deux oscillateurs.

Au cours de cette étude, nous avons également comparé deux types d'estimation de la variance : la variance statistique et la variance d'Allan. L'étude a confirmé expérimentalement que la variance statistique surestime le jitter et une évaluation plus poussée des mesures de variance intégrées a montré que la variance d'Allan est encore plus adaptée à l'implémentation matérielle, car elle nécessite une surface plus petite et son calcul est plus rapide que le calcul de la variance statistique.

Pour finaliser l'étude des TRNGs basés sur des oscillateurs, nous avons évalué leur sensibilité aux interférences internes et externes et nous avons recherché des méthodes permettant d'atténuer ces interférences. Nous avons comparé les résultats de la mesure embarquée de la variance dans le FPGA Intel Cyclone V ne contenant rien d'autre que cette mesure et un TRNG, avec les résultats de la même mesure lorsqu'un circuit plus complexe avec un chiffrement AES était implémenté dans le même FPGA.

La comparaison a montré que même si le jitter des deux oscillateurs du TRNG augmentait lorsqu'un chiffreur AES passait à côté, le jitter relatif entre les deux oscillateurs représenté par

la variance des valeurs de compteur ne changeait pas. Cela signifie que la variance des valeurs de compteur ne surestime pas le jitter dans le système complexe, où des sources de bruit internes sont présentes.

L'interférence externe a été évaluée en remplaçant l'un des oscillateurs du TRNG par un oscillateur externe à quartz à faible bruit. L'analyse des valeurs de compteur obtenues à partir d'un tel TRNG modifié a révélé la présence de forts signaux externes basse fréquence. Nous avons trouvé que ces signaux viennent des variations d'alimentation, qui étaient présentes malgré les alimentations linéaires à faible bruit utilisées sur les cartes d'évaluation. Une telle interférence externe est donc inévitable et le seul moyen de l'éviter consiste à utiliser deux oscillateurs internes mis en oeuvre de manière identique.

Liste des publications

Les travaux présentés dans cette thèse ont été publiés dans des journaux scientifiques, des conférences internationales et les livrables du projet HECTOR.

Publication du journal scientifique:

[51] E. Noumon Allini, M. Skórski, O. Petura, F. Bernard, M. Laban, and V. Fischer, "Evaluation and monitoring of free running oscillators serving as source of randomness," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2018, pp. 214–242, Aug. 2018

Publications de la conférence internationale:

[33] O. Petura, U. Mureddu, N. Bochard, V. Fischer, and L. Bossuet, "A survey of AIS-20/31 compliant TRNG cores suitable for FPGA devices," in *26th International Conference on Field-Programmable Logic and Applications, FPL '16, Lausanne, Switzerland*, Aug. 2016

[36] O. Petura, U. Mureddu, N. Bochard, V. Fischer, and L. Bossuet, "A survey of AIS-20/31 compliant TRNG cores implemented on FPGAs," in *TRUDEVICE – 6th Conference on Trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE 2016), Barcelona, Spain*, Nov. 2016

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

[44] E. N. Allini, O. Petura, V. Fischer, and F. Bernard, "Optimization of the PLL configuration in a pll-based TRNG design," in *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018*, pp. 1265–1270, 2018

[46] J. Balasch, F. Bernard, V. Fischer, M. Grujić, M. Laban, O. Petura, V. Rožić, G. van Battum, I. Verbauwhede, M. Wakker, and B. Yang, "Design and testing methodologies for true

random number generators towards industry certification," in *2018 IEEE 23rd European Test Symposium (ETS)*, May 2018

Livrables HECTOR:

[37] M. Deutschmann, S. Lattacher, J. Delvaux, V. Rozic, B. Yang, D. Singelee, L. Bossuet, V. Fischer, U. Mureddu, O. Petura, A. A. Yamajako, B. Kasser, and G. BATTUM, "HECTOR deliverable D2.1 – report on selected TRNG and PUF principles," Feb. 2016

[43] G. Battum, S. Lattacher, M. Deutschmann, B. Kasser, M. Agoyan, J. Nicolai, M. Madau, R. Sussella, J. Balasch, M. Grujic, V. Fischer, O. Petura, M. Laban, J. Luhman, M. Wakker, and R. Malafre, "HECTOR deliverable D2.4 – robustness tests on TRNGs and PUFs," July 2018

Contributions

- Nous avons proposé une méthodologie pour une évaluation équitable de différents noyaux TRNG et démontré son efficacité sur plusieurs TRNG implantés dans différentes familles de FPGA. Grace à cette approche, nous avons pu sélectionner des noyaux TRNG à implanter dans les ASIC.
- Nous avons implémenté les noyaux TRNG sélectionnés dans des ASIC et évalué leur faisabilité et leurs performances.
- Nous avons étudié de près la conception du PLL-TRNG et avons proposé des outils automatisés pour la conception de PLL-TRNG, qui permettent le développement rapide de TRNG de haute qualité avec différentes contraintes technologiques.
- Nous avons clairement démontré que les TRNG basés sur des oscillateurs doivent être conçus selon un principe différentiel : à l'aide de deux oscillateurs internes implémentés identiquement.
- Nous avons montré que l'extraction d'aléa du jitter d'horloge en comptant les périodes d'horloge affectée par le jitter est beaucoup plus efficace que l'échantillonnage des fronts d'horloge affectée par le jitter. Cette méthode d'extraction d'aléa permet de concevoir des TRNGs avec un débit plus élevé et constitue une base solide pour les tests embarqués, qui peuvent directement utiliser les valeurs de compteur pour calculer la variance.
- Nous avons proposé d'utiliser la variance d'Allan calculée à partir des valeurs de compteurs comme méthode de test embarqué des TRNGs basés sur des oscillateurs.

Bibliography

[1] B. Sunar, W. Martin, and D. Stinson, "A provably secure true random number generator with built-in tolerance to active attacks," *IEEE Transactions on Computers*, vol. 56, pp. 109–119, Jan. 2007.

[2] K. Wold and C. H. Tan, "Analysis and enhancement of random number generator in FPGA based on oscillator rings," in *ReConFig'08: 2008 International Conference on Reconfigurable Computing and FPGAs, 3-5 December 2008, Cancun, Mexico, Proceedings*, pp. 385–390, 2008.

[3] M. Laban, M. Drutarovsky, V. Fischer, and M. Varchola, "Platform for testing and evaluation of PUF and TRNG implementations in FPGAs," in *TRUDEVICE – 6th Conference on Trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE 2016), Barcelona, Spain*, Nov. 2016.

[4] M. Drutarovsky and P. Galajda, "Chaos-based true random number generator embedded in a mixed-signal reconfigurable hardware," *Journal Of Electrical Engineering Bratislava*, vol. 57, no. 4, p. 218, 2006.

[5] M. Drutarovsky and P. Galajda, "A robust chaos-based true random number generator embedded in reconfigurable switched-capacitor hardware," in *17th International Conference Radioelektronika, 2007.*, pp. 1–6, 2007.

[6] A. McWhorter, "1/f noise and germanium surface properties," in *Semiconductor surface physics*, pp. 207–228, Philadelphia, PA: Univ. Pennsylvania Press, 1957.

[7] F. Hooge, "1/f noise is no surface effect," *Physics letters A*, vol. 29, no. 3, pp. 139–140, 1969.

[8] N. Kasdin, "Discrete simulation of colored noise and stochastic processes and 1/fα; power law noise generation," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 802–827, 1995.

[9] C. Liu, *Jitter in Oscillators with 1/f Noise Sources and Application to True RNG for Cryptography*. PhD thesis, WORCESTER POLYTECHNIC INSTITUTE, Jan. 2006.

[10] Intel, "Reliability report," tech. rep., 2017.

[11] Altera, "Understanding metastability in FPGAs," tech. rep., 1999.

[12] L. M. Reyneri, D. Del Corso, and B. Sacco, "Oscillatory metastability in homogeneous and inhomogeneous flip-flops," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 1, pp. 254–264, 1990.

[13] M. Varchola and M. Drutarovský, "New high entropy element for FPGA based true random number generators," in *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, pp. 351–365, 2010.

[14] M. Baudet, D. Lubicz, J. Micolod, and A. Tassiaux, "On the security of oscillator-based random number generators," *Journal of Cryptology*, vol. 24, pp. 398–425, Apr. 2011.

[15] V. Rozic, B. Yang, W. Dehaene, and I. Verbauwhede, "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*, pp. 116:1–116:6, ACM, 2015.

[16] A. Cherkaoui, V. Fischer, A. Aubert, and L. Fesquet, "A self-timed ring based true random number generator," in *IEEE 19th International Symposium on Asynchronous Circuits and Systems (ASYNC), 2013*, pp. 99–106, IEEE, 2013.

[17] N. Bochard, F. Bernard, V. Fischer, and B. Valtchanov, "True-randomness and pseudo-randomness in ring oscillator-based true random number generators," *International Journal of Reconfigurable Computing*, vol. 2010, pp. 879281:1–879281:13, Feb. 2010.

[18] U. Mureddu, N. Bochard, L. Bossuet, and V. Fischer, "Experimental study of locking phenomena on oscillating rings implemented in logic devices," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. PP, pp. 1–12, 03 2019.

[19] P. Kohlbrenner and K. Gaj, "An embedded true random number generator for FPGAs," in *Proceedings of the ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays, FPGA 2004, Monterey, California, USA, February 22-24, 2004*, pp. 71–78, 2004.

[20] V. Fischer and M. Drutarovsky, "True random number generator embedded in reconfigurable hardware," in *Cryptographic Hardware and Embedded Systems - CHES 2002*, vol. 2523 of *LNCS*, pp. 415–430, Redwood Shores, CA, USA, Springer Verlag, 2002.

[21] F. Bernard, V. Fischer, and B. Valtchanov, "Mathematical model of physical rngs based on coherent sampling," *Tatra Mountains Mathematical Publications*, vol. 45, no. 1, pp. 1–14, 2010.

[22] V. Fischer and D. Lubicz, "Embedded evaluation of randomness in oscillator based elementary trng," in *Cryptographic Hardware and Embedded Systems – CHES 2014: 16th International Workshop, Busan, South Korea, September 23-26, 2014, Proceedings* (L. Batina and M. Robshaw, eds.), vol. 8731 of *Lecture Notes in Computer Science*, pp. 527–543, Springer Berlin Heidelberg, 2014.

[23] R. B. Davies, "Exclusive OR (XOR) and hardware random number generators." Online. Available at: http://www.robertnz.net/pdf/xor2.pdf, Feb. 2002.

[24] J. Von Neumann, "13. various techniques used in connection with random digits," *National Bureau of Standards Applied Math Series*, vol. 12, pp. 36–38, 1951.

[25] W. Killmann and W. Schindler, "A proposal for: Functionality classes for random number generators," 2011.

[26] U.S. DEPARTMENT OF COMMERCE / National Institute of Standards and Technology, "Announcing the advanced encryption standard (AES)," Nov. 2001. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[27] D. Jablon, "IEEE P1363 standard specifications for public-key cryptography," Nov. 2001.

[28] U.S. DEPARTMENT OF COMMERCE / National Institute of Standards and Technology, "FIPS 180-4 secure hash standard (SHS)," Aug. 2012. https://csrc.nist.gov/publications/detail/fips/180/4/final.

[29] U.S. DEPARTMENT OF COMMERCE / National Institute of Standards and Technology, "Security requirements for cryptographic modules FIPS 140-1," Jan. 1994.

[30] G. Marsaglia, "Diehard: Battery of tests of randomness." Online. Available at: http://stat.fsu.edu/pub/diehard/, 1996.

[31] A. Rukhin, J. Soto, J. Nechvatal, J. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo, "A statistical test suite for random and pseudorandom number generators for cryptographic applications, nist special publication 800-22." Online. Available at: http://csrc.nist.gov/, 2001.

[32] M. S. Turan, E. Barker, J. Kelsey, K. A. McKay, M. L. Baish, and M. Boyle, "Recommendation for the entropy sources used for random bit generation." NIST Special Publication 800-90B, 2018.

[33] O. Petura, U. Mureddu, N. Bochard, V. Fischer, and L. Bossuet, "A survey of AIS-20/31 compliant TRNG cores suitable for FPGA devices," in *26th International Conference on Field-Programmable Logic and Applications, FPL '16, Lausanne, Switzerland*, Aug. 2016.

[34] N. Bochard, C. Marchand, O. Peťura, L. Bossuet, and V. Fischer, "Evariste III: A new multi-FPGA system for fair benchmarking of hardware dependent cryptographic primitives." Workshop on Cryptographic Hardware and Embedded Systems, CHES 2015, Sept. 2015. Poster.

[35] A. Cherkaoui, V. Fischer, A. Aubert, and L. Fesquet, "A very high speed true random number generator with entropy assessment," in *Cryptographic Hardware and Embedded Systems-CHES 2013*, pp. 179–196, Springer, 2013.

[36] O. Petura, U. Mureddu, N. Bochard, V. Fischer, and L. Bossuet, "A survey of AIS-20/31 compliant TRNG cores implemented on FPGAs," in *TRUDEVICE – 6th Conference on Trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE 2016), Barcelona, Spain*, Nov. 2016.

[37] M. Deutschmann, S. Lattacher, J. Delvaux, V. Rozic, B. Yang, D. Singelee, L. Bossuet, V. Fischer, U. Mureddu, O. Petura, A. A. Yamajako, B. Kasser, and G. BATTUM, "HECTOR deliverable D2.1 – report on selected TRNG and PUF principles," Feb. 2016.

[38] C. C. Coello, G. Lamont, and D. van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*. Genetic and Evolutionary Computation, Berlin, Heidelberg: Springer, 2nd ed., 2007.

[39] A. Schreyer, "GA optimization for MS EXCEL," 2005.

[40] M. Laban, S. Lattacher, M. Deutschmann, V. Rozic, B. Yang, D. Singelee, V. Fischer, U. Mureddu, A. Anzala-Yamajako, F. Melzani, M. Kleja, M. Eichlseder, and G. B. M. Wakker, "HECTOR deliverable D4.1 – demonstrator specification," Feb. 2017.

[41] V. Fischer, F. Bernard, and N. Bochard, "Modern random number generator design – case study on a secured PLL-based TRNG," *Methods and Applications of Informatics and Information Technology*, 2019.

[42] Altera, *Cyclone V Device Datasheet (CV51002)*, 2015.

[43] G. Battum, S. Lattacher, M. Deutschmann, B. Kasser, M. Agoyan, J. Nicolai, M. Madau, R. Sussella, J. Balasch, M. Grujic, V. Fischer, O. Petura, M. Laban, J. Luhman, M. Wakker, and R. Malafre, "HECTOR deliverable D2.4 – robustness tests on TRNGs and PUFs," July 2018.

[44] E. N. Allini, O. Petura, V. Fischer, and F. Bernard, "Optimization of the PLL configuration in a pll-based TRNG design," in *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018*, pp. 1265–1270, 2018.

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

[46] J. Balasch, F. Bernard, V. Fischer, M. Grujić, M. Laban, O. Petura, V. Rožić, G. van Battum, I. Verbauwhede, M. Wakker, and B. Yang, "Design and testing methodologies for true random number generators towards industry certification," in *2018 IEEE 23rd European Test Symposium (ETS)*, May 2018.

[47] B. Valtchanov, A. Aubert, F. Bernard, and V. Fischer, "Characterization of randomness sources in ring oscillator-based true random number generators in FPGAs," *Design and Diagnostics of Electronic Circuits and Systems, 2008. DDECS 2010. 13th IEEE Workshop on*, pp. 1–6, 2010.

[48] C. Costea, F. Bernard, V. Fischer, and R. Fouquet, "Implementation of ring oscillators based physical unclonable functions with independent bits in the response," *International Journal of Reconfigurable Computing*, vol. ID 168961, 2012.

[49] P. Haddad, F. Bernard, V. Fischer, and Y. Teglia, "Random number generators, does jitter realizations can still be considered as mutually independent ?," in *colloque du GDR SOC-SIP*, (Paris, France), June 2014.

[50] D. W. Allan and J. A. Barnes, "A Modified "Allan Variance" with Increased Oscillator Characterization Ability," in *Proceedings of 35th Annual Frequency Control Symposium*, pp. 470–475, 1981.

[51] E. Noumon Allini, M. Skórski, O. Petura, F. Bernard, M. Laban, and V. Fischer, "Evaluation and monitoring of free running oscillators serving as source of randomness," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2018, pp. 214–242, Aug. 2018.

Index