KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Attacking and Securing Hardware Random Number Generators

Adriaan Peetermans

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: elektrotechniek, optie Elektronica en geïntegreerde schakelingen

Promotor:
Prof. dr. ir. Ingrid Verbauwhede

Assessoren:
Prof. dr. ir. Bart Preneel
Prof. dr. ir. Wim Dehaene

Begeleiders:
Dr. Vladimir Rožić
Bohan Yang
Miloš Grujić
Dr. Josep Balasch

Academiejaar 2017 – 2018

Preface

Na een jaar intensief werken aan deze masterproef dat boordevol zat aan nieuwe inzichten en me een stap dichter heeft gebracht bij mijn toekomst als elektrisch ingenieur, zou ik graag stil willen staan bij de mensen die dit mogelijk maakten.
Als eerste zou ik graag Prof. Ingrid Verbauwhede willen bedanken. Ze gaf me de mogelijkheid om deze masterproef op COSIC af te leggen.
Daarnaast wil ook graag mijn dagelijkse begeleiders: Vladimir Rožić, Bohan Yang, Miloš Grujić en Josep Balasch bedanken om me steeds van raad en goede ideeën te voorzien.
Vervolgens zou ik ook graag mijn ouders en twee zussen willen bedanken om me tijdens de weekends steeds voldoende motivatie te geven en ervoor te zorgen dat ik nooit iets tekort kwam.
Als laatste zou ik ook nog mijn vriendin Mira willen bedanken. Ze voorzag steeds een luisterend oor en zorgde er voor dat mijn ontspanning steeds van hoge kwaliteit was.

After a year of intensive work on this master thesis that was full of new insights and that brought me one step closer to my future as electrical engineer, I would like to stand still at the people who made this possible.
Firstly, I would like to thank Prof. Ingrid Verbauwhede. She gave me the opportunity to make this master thesis at COSIC.
In addition, I also want to thank my daily supervisors: Vladimir Rožić, Bohan Yang, Miloš Grujić and Josep Balasch to always provide me with advice and good ideas.
Then I would also like to thank my parents and two sisters to give me sufficient motivation during the weekends and to ensure that I was never short of anything.
Finally, I would also like to thank my girlfriend Mira. She always provided a listening ear and ensured that my relaxed moments were always of high quality.

Adriaan Peetermans

Contents

Abstract

A True Random Number Generator (TRNG) is a critical component in many embedded security applications. As they serve as an input for cryptographic algorithms, the importance of their security cannot be overestimated. The fact that TRNGs often magnify small analog phenomena (noise), makes them a very vulnerable part of the system. In this work, we implemented a recently proposed Self-Timed Ring (STR)-based TRNG and verified its working by evaluating the randomness of the generated output data by executing the NIST SP 800-22, and the AIS 31 test suites on it. In the second phase, we studied the effects of both physical attacks and imperfect implementations on the behaviour of the TRNG. More in particular, we performed experiments with surrounding temperature- and power supply voltage variations. To evaluate the possible influence of an imperfect implementation, we experimented with routing asymmetry, surrounding activity and placement. To support the obtained results, we created a model of the STRs in *MATLAB* to estimate the dependency on critical parameters. The outcome of this model agreed with what was obtained from the experiments. In the final phase, we designed on-line tests as a countermeasure and evaluated its performance experimentally. These tests are based on the results of the physical attacks. These attacks revealed useful features that are closely related to the quality of the output stream. They are therefore used in the generation of an alarm signal that notifies the users of the random data.

Samenvatting

Introductie

De vraag naar beveiliging in veel ingebedde apparaten wordt steeds veeleisender. Op zijn beurt, stelt dit strengere eisen aan de on-board True Random Number Generator (TRNG), die vaak de basis is van vele cryptografische protocollen. Vooral het gebruik van TRNG's in Field Programmable Gate Arrays (FPGA's) krijgt veel aandacht. Hier is slechts beperkte (enkel digitale) hardware beschikbaar om de generatie van willekeurige getallen mogelijk te maken. TRNG's worden vaak gebruikt in cryptografische toepassingen om bijvoorbeeld willekeurige sleutels, parameters in challenge-responsprotocollen, padding-waarden en nonces te creëren. Vanwege de beschikbaarheid van alleen digitale hardware in FPGA's, zijn deze TRNG's ofwel gebaseerd op de onvoorspelbaarheid van een metastabiel geheugenelement [1], [2] of op de timing jitter aanwezig in vrijlopende oscillatoren [3], [4].

De gegenereerde data moet van voldoende kwaliteit zijn, wat betekent dat er geen statistische zwakheden zoals een aanzienlijke hoeveelheid bias of correlatie tussen de gegenereerde bits zichtbaar mag zijn. De output van de TRNG zou enkele vooraf gedefinieerde statistische testen moeten doorstaan, zoals bijvoorbeeld de testen beschreven door NIST [5] of AIS 31 [6]. Voldoende willekeurigheid van de gegenereerde data alleen is niet voldoende. De TRNG moet ook bestand zijn tegen passieve en actieve implementatieaanvallen. Passieve aanvallen gebruiken ongewenste nevenkanaal lekkage (side-channel leakage) om informatie te verzamelen over de interne toestand van de TRNG. Deze informatie wordt vervolgens gebruikt om toekomstige data te modelleren en te voorspellen. Actieve aanvallen brengen de TRNG uit zijn stabiel werkingsgebied, door de werkingsomstandigheden zoals omgevingstemperatuur, voedingsspanning, klokfrequentie, enz. te wijzigen [7], [8], [9]. De eerste twee van deze referenties laten zien dat de klassieke TRNG met Ring Oscillators (RO's) erg gevoelig is voor aanvallen die de vrijlopende RO's vergrendelen (locking), door ofwel glitches toe te voegen aan de voeding of door elektromagnetische straling met een nauwkeurige frequentie naar de FPGA te sturen.

Om deze zwakte te overwinnen, werd een nieuwe, op oscillator gebaseerde TRNG voorgesteld in [10]. Het maakt gebruik van de robuustere Self-Timed Ring (STR) in plaats van een normale RO om de jitterachtige signalen te creëren. Dit idee om STRs te gebruiken om de willekeur te genereren, is verder uitgewerkt door [11]. Zij beweren dat door het feit dat de uitgang van elk element in de STR Gaussiaanse jitter bevat die alleen wordt gegenereerd door dat zelfde element en dus onafhankelijk is van de

jitter aan de andere uitgangen, de uitgang van elke fase kan worden gebruikt als een onafhankelijke bron van willekeurigheid. Deze jitter wordt vervolgens geëxtraheerd met behulp van het Coherent Sampling (CS) principe. De uitgangen van twee verschillende STR's worden vergeleken en een bit wordt gegenereerd volgens het verschil in fase.

In dit werk zullen we een model voor deze TRNG maken om de effecten van verschillende parameters (routing vertraging en poort vertraging), die voorheen meestal werden genegeerd, te onderzoeken en om de gevolgen ervan op het gedrag en de gegenereerde data te bestuderen. Vervolgens voeren we experimenten uit om deze resultaten te bevestigen en komen we met algemene ontwerp richtlijnen. Bovendien voeren we ook omgevingstemperatuur- en voedingsspanning aanvallen uit om de robuustheid van deze TRNG tegen actieve aanvallen te bepalen. Aan het einde wordt een online testset voorgesteld, specifiek voor dit TRNG-ontwerp, om verminderde veiligheid zo snel en efficiënt mogelijk te detecteren.

Model en simulaties

Een STR wordt gekenmerkt door het feit dat het mogelijk is om meerdere positieve flanken (gebeurtenissen) tegelijkertijd te laten propageren door de ring. Dit is in tegenstelling tot standaard RO's waar slecht één enkele positieve- en negatieve flank aanwezig is. De STR kan zich hierdoor in twee toestanden bevinden. Een gelijkmatige gespreide toestand, waar alle gebeurtenissen zich uniform over de ring verspreiden en een burst toestand, waar de gebeurtenissen snel na elkaar komen waarna een lange tijd zonder enige actie volgt. De gelijkmatig gespreide werkingsmodus wordt verondersteld in veel veiligheidsanalyses van STR's, zoals bijvoorbeeld in [12]. Wanneer dit echter niet het geval is vanwege een slechte implementatie of een verandering in de werkingsomstandigheden (actieve aanval), gelden deze beveiligingsanalyses ook niet meer en kan pure willekeur niet meer worden gegarandeerd. In welke modus een STR zich zal bevinden wordt grotendeels bepaald door de aanwezige symmetrie. Meer bepaald, de symmetrie van de interconnectie lijnen. Elk element van een STR heeft twee ingangen, een voorwaartse (komende van het vorige element) en een achterwaartse (komende van het volgende element). Deze lijnen moeten een symmetrische vertraging hebben voor een goede werking van de ring.

Het STR-model is grotendeels gebaseerd op het voorgestelde model in [13]. Oorspronkelijk was het gebaseerd op de vijf parameters van het Charlie-diagram [14]. Een zesde parameter is toegevoegd om de willekeurige vertraging van de Look-Up-Tables (LUT's) die de elementen van de STR's vormen mee in rekening te brengen. Dit wordt de gate jitter magnitude genoemd. De vertraging van elke fase bestaat dan uit een deterministisch gedeelte (gegeven door het Charlie-diagram) en een willekeurig deel (gegeven door de gate jitter). Een zevende parameter is toegevoegd om rekening te houden met de routing vertragingen. Vanwege de beperkte routing mogelijkheden in een FPGA, zullen de routing vertragingen niet exact hetzelfde zijn voor elke poort. Daarom wordt een lijst met voorwaartse en achterwaartse routing

vertragingen (één voorwaartse en achterwaartse vertraging voor elke poort) aan het model toegevoegd.

Om het hoog-niveau model te simuleren, worden numerieke waarden aan deze parameters toegewezen. Merk op dat dit model alleen wordt gebruikt om de relatieve grootte van het effect van variaties van deze parameters te meten. Daarom is de absolute nauwkeurigheid van de simulatieresultaten van minder belang en is slechts een ruwe schatting van de parameters noodzakelijk. Het model is geïmplementeerd in *MATLAB*.

Dit model is dan gebruikt om vier verschillende simulaties uit te voeren: het effect van de bemonsteringssnelheid, variaties in routing asymmetrie, variaties in poort asymmetrie en als laatste de bijdrage van de jitter magnitude aan de uiteindelijk gegenereerde data. Vooral de resultaten van het effect van routing asymmetrie zijn belangrijk. De interconnectie vertraging draagt wel bij aan een groot deel van de gehele propagatie vertraging van gebeurtenissen door de ring. Wanneer deze vertraging symmetrisch is, wat betekent dat de vertraging van de voorwaartse en achterwaartse lijnen ongeveer gelijk zijn, verandert er niets aan de gemaakte assumpties in de veiligheidsanalyses van STR's. Dit verandert volledig wanneer de vertraging niet symmetrisch is. Deze simulatie testte hoe groot de asymmetrie moet zijn, om de ring uit gelijkmatig gespreide modus te halen. Uit de resultaten bleek dat dit al het geval was voor een asymmetrie met een factor van drie of meer. Dus wanneer de voorwaartse vertraging meer dan drie keer groter wordt dan de achterwaartse.

De resultaten tonen aan dat zorgvuldige routing in feite erg belangrijk is om te verzekeren dat de ringen in een gelijkmatig gespreide modus zullen evolueren. Het gevolg hiervan is dat handmatige plaatsing op FPGA's altijd noodzakelijk zal zijn. Wanneer de plaatsing en routing worden overgelaten aan automatische synthese programma's, is de kans groot dat de routing erg asymmetrisch wordt. Hierdoor is de TRNG zeer kwetsbaar voor aanvallen.

Actieve aanvallen en implementatie defecten

Om de in de vorige paragraaf verkregen resultaten te verifiëren, hebben we twee ontwerpen op FPGA geïmplementeerd. De eerste focust alleen op de STR, om het effect van routing asymmetrie op de grootte van de burst-modus te meten. De tweede implementeert de volledige TRNG om daarop dan actieve aanvallen uit te voeren.

De STR wordt in beide gevallen op de zelfde manier ontworpen. Elk element van de STR wordt geïmplementeerd met behulp van slechts één LUT met 6 ingangen. Elke slice bevat acht elementen, dus er zijn in totaal vier slices nodig om een 32-elementen STR te maken. Om de routing asymmetrie te wijzigen, wordt een extra buffer ingevoegd tussen elke uitgang van elk element en de voorwaartse ingang van het volgende element. Deze buffer kan dan verder weg van de STR worden verplaatst om de asymmetrie te vergroten. Tijdens het ontwerp van de volledige TRNG moet veel aandacht besteed worden aan de symmetrie van de twee STR's en aan het uit

de weg houden van naburige componenten. Dit is gedaan door Pblocks te maken die bepalen welk gebied alleen beschikbaar is voor de TRNG.

Eerst wordt experimenteel bepaald in welke maten de asymmetrie het werkingspunt van de STR kan beïnvloeden. Voor elke test wordt de STR geïnitialiseerd met een maximaal aantal gebeurtenissen, al in burst-modus. Alle gebeurtenissen zijn bij elkaar gegroepeerd tijdens de initialisatie. Op deze manier wordt duidelijk of de STR in staat is om te evolueren naar een gelijkmatige gespreide toestand of niet. De resultaten van dit experiment tonen een positieve correlatie tussen de routing asymmetrie en de burst-modus van de STR en blijkbaar is de STR zeer gevoelig voor kleine stijgingen van de asymmetrie. Dit komt overeen met de resultaten die zijn verkregen uit de simulaties.

Er worden drie posities in de FPGA gekozen om het effect van de omgevingstemperatuur te bestuderen. In [9], zijn temperatuur, voltage (statisch en met glitches) en clock glitches aanvallen uitgevoerd op een soortgelijke TRNG. Ze ontdekten geen significante afhankelijkheid van de uitgang willekeurigheid op de omgevingstemperatuur. Dezelfde resultaten worden hier verkregen, de STR-frequentie hangt wel af van de temperatuur. Maar omdat deze TRNG enkel werkt met het verschil in frequentie, is er geen meetbare verandering aan de uitgang. Er kan dus geconcludeerd worden dat de voorgestelde TRNG zeer robuust is voor variaties in omgevingstemperatuur.

Als laatste is het effect van een lagere voedingsspanning geanalyseerd. Opnieuw, zijn er drie verschillende posities gekozen. De voedingsspanning wordt verlaagd van de standaardwaarde van 1,2 V naar 0,9 V in stappen van 0,1 V. Nogmaals, is de STR-frequentie erg afhankelijk van de voedingsspanning. Een van de STR's stopt zelfs met oscilleren bij 0,9 V. Als we dan kijken naar de slaagkans van de NIST-test bij verschillende voedingsspanningen, is de vermindering van de performantie bij lagere voedingsspanningen duidelijk. De reden hiervoor kan een grotere frequentie-offset tussen de twee ringen zijn en/of meer evolutie richting burst-modus.

Online testen

In de laatste sectie stellen we een efficiënte online testset voor, specifiek voor deze TRNG. We volgen de redenering, voorgesteld door [15]. Volgens de resultaten van de fysieke aanvallen kunnen drie handige kenmerken worden geëxtraheerd: burst-index (hoe hard de STR in burst-modus zit), CS-telling (hoe goed beide STR's in fase zijn) en de STR-frequentie. Omdat de voedingsspanning aanval de enige aanval was die de uitgangs-willekeurigheid aanzienlijk kon verlagen, zal deze aanval worden gebruikt om de werking van de onlinetests te valideren. De histogrammen van de CS-telling en STR-frequentie tonen een duidelijke afhankelijkheid aan de voedingsspanning. Een online test wordt beoordeeld op basis van zijn vermogen om een falende TRNG te scheiden van een goed werkende. De histogrammen tonen echter dat perfecte scheiding niet altijd mogelijk is.

Een ander probleem is het vaststellen wanneer de TRNG daadwerkelijk begint te falen. Verschillende definities van TRNG-gezondheid zijn mogelijk. We definiëren een TRNG als gefaald als het totale slaagpercentage van de NIST-testset lager is dan

70 %. Deze drempelwaarde is arbitrair en kan worden aangepast aan de behoeften van de toepassing. Het bepaalt vervolgens welke voedingsspanningen nog steeds een werkende TRNG opleveren. Hieruit kunnen dan drempels bepaald worden die de misclassificatie van de online testen minimaliseren. Met behulp van deze strategie, kon een online testset gemaakt worden die een gemiddelde misclassificatie heeft van slechts 2,82 %.

Toekomstwerk

Een toekomstig werk kan zich richten op het uitvoeren van een meer diverse aanvalsstrategie. Er kan meer onderzoek worden gedaan naar de weerstand van STR's tegen vergrendeling op externe oscillerende signalen. Omdat de TRNG erg gevoelig is voor de implementatie, kan de prestatie op andere FPGA-families en andere synthese programma's ook nog verder worden onderzocht.

List of Figures

List of Tables

List of Abbreviations

Abbreviations

AIS	Anwendungshinweise und Interpretationen zum Schema
CLB	Configurable Logic Block
CS	Coherent sampling
CV	Coefficient of Variation
DFF	D-Flip Flop
EM	Electromagnetic
FF	Flip Flop
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
HDMI	High-Definition Multimedia Interface
HECTOR	Hardware Enabled CrypTO and Randomness
IC	Integrated Circuit
IoT	Internet of Things
LFSR	Linear Feedback Shift Register
LSB	Least Significant Bit
LUT	Look-Up-Table
MSB	Most Significant Bit
NIST	National Institute of Standards and Technology
PDF	Probability Density Function
PLL	Phase Locked Loop
PMF	Probability Mass Function
PRNG	Pseudo Random Number Generator
PUF	Physically Unclonable Function
RO	Ring Oscillator
RRA	Relative Routing Asymmetry
RSA	Relative Stage Asymmetry
SATA	Serial AT Attachment
SoC	System on Chip
STR	Self-Timed Ring

STRNG	Self Time Ring True Random Number Generator
TERO	Transition Effect Ring Oscillator
TFF	Toggle-Flip Flop
TOTAL	TRNG On-the-fly Testing for Attack detection using Lightweight hardware
TRNG	True Random Number Generator
UART	Universal Asynchronous Receiver-Transmitter
UCF	User Constrains Files

Chapter 1

Introduction

1.1 Introduction to random number generators and physical attacks

1.1.1 What is a random number generator

This thesis studies what it takes to generate on-chip random (in the true sense of the word) data. A True Random Number Generator (often abbreviated as TRNG) is a device, especially designed to realize this task. This random data is an essential input to numerous cryptographical algorithms and protocols. It is often used to generate random keys, masks, challenges, nonces, etc. These numbers enable then the encryption of secret messages or allow to check the authenticity of certain data. It is of utmost importance that these numbers remain secure and don't become readily available for adversaries.

Random numbers also play an important role outside the security domain. They are important in on-line gaming, such as poker tournaments or other games where a random component is needed to determine e.g. a player's cards or throw a dice. Certainly when money is involved, one must be sure that all interactions happen fairly. Another domain is the use in scientific and mathematical calculations. Certain algorithms behave better when they can be initialized in a random state (e.g. genetic algorithms, neural networks, etc.). Also for the use of Monte Carlo simulations to estimate an unknown probability distribution, e.g. to estimate the yield when manufacturing integrated circuits. In this last example, the unpredictability of the random data is of less importance than its good statistical properties (uniform distribution, low correlation, etc.).

The design of a TRNG has many (often contradicting) requirements. First of all, they must be extremely light-weight. In particular, they can only have limited area- and energy requirements. This property enables them to be used in small embedded devices. These devices have become increasingly popular since the advent of the Internet of Things (IoT). Nowadays, embedded electronic devices are present everywhere around us. This presence is often without the user being aware of it. Many people carry bankcards, IDs, rely on sensor networks, have multiple smart

devices in their home, car and even inside their body. These devices collect data in a person's near surroundings and this data is usually considered to be private. It is therefore essential for these devices to be able to communicate in a secure way.

Because these devices are everywhere, it is not difficult for an adversary to get his hands on one. This situation is in strong contrast with the past where security was only concerned on the channel connecting sender and receiver. Nowadays, both the connecting channel and the two endpoints can be available to an attacker. Therefore, it is important that not only the algorithm itself, but also its implementation is resistant to any type of abuse. Because the TRNG is a critical component in many cryptographical protocols, its implementation must be resistant to attacks. It must contain ways of detecting an attack and notify the users of the random data.

1.1.2 Attacking a random number generator

It is very important to perform different attacks on a TRNG. These attacks enable the designer to study the effects on the behaviour of the TRNG and leads to the design of effective countermeasures and tests to detect such an attack. Attacks can roughly be divided into two groups. Firstly, the passive attacks [22], [23]. They do not alter the operating conditions of the TRNG. The TRNG has thus no means of detecting this kind of attack. A passive attack extracts as much information out of the device (generated output data, power consumption, etc.) and then tries to model and predict with a higher than random chance the output data. Secondly, the active attacks [7], [8]. They are performed in this thesis. These attacks do change the operating conditions and aim to get a (controllable) reduction of the randomness in the output data stream.

1.2 Contributions

The main contributions of this thesis are:

- The design and implementation of a not yet fully explored TRNG. This thesis implements a new TRNG topology that was only recently proposed. It offers a reasonably high speed at the expense of only limited area requirements.
- A model of the TRNG has been created in *MATLAB* and various simulations are performed. This model enables the study of implementation defects and other parameters that are impossible to control in a real environment.
- Two sets of active attacks are performed on the designed TRNG: both the effects of changing surrounding temperature and power supply voltage are measured and interesting features that indicate such an attack are selected.
- Some imperfect implementations are created to validate the results obtained from the simulations and guidelines for general TRNG design are proposed.
- On-line tests, specific for the designed TRNG are created and their ability to detect attacks is tested.

1.3 Outline

This thesis contains 9 chapters:

Chap. 1 contains a general introduction to this thesis.

In Chap. 2, most of the theory concerning random number generation is explained and different types of TRNGs are presented. The chapter ends with a detailed description of the TRNG implemented in this thesis.

Chap. 3 explains the model used to simulate the behaviour of the proposed TRNG and shows the results from the different simulations.

Then, Chap. 4 presents the principals about statistical testing and gives a description of the two test suites used to test the generated random data.

The steps in the design process are further explained in Chap. 5. It also presents the results from performing the statistical tests on the generated random data.

In Chap. 6, the results from the simulations are validated through enlargement of possible implementation mistakes. Some general guidelines of good TRNG design are proposed.

Chap. 7 illustrates the set-up of the physical attacks and further analyses its results. The statistical test suites are applied on the generated random data to get a detailed look on the changes in behaviour.

Chap. 8 illustrates how useful features coming from the physical attacks can be exploited, to create efficient and fast on-line tests for the detection of possible attacks.

At the end, Chap. 9 presents a summary of this thesis and proposes possible directions for future work.

Chapter 2

Background in random number generators

This chapter is an introduction in the world of random number generation. It starts by describing the general workings of both True Random Number Generators (TRNGs) and Pseudo Random Number Generators (PRNGs). Then comes a section that describes which indeterministic (noise) processes are responsible for random behaviour in integrated circuits. The third section distinguishes and explains in more detail, the different types of TRNGs. The last section then describes the mechanism and architecture of the TRNG design studied in this thesis.

2.1 Principles of random number generation

A Random Number Generator (RNG) is a device that can create large amounts of data, which can only be predicted by no more than a random chance. For the RNGs described in this thesis, the output data is a sequence of binary values. This sequence can be regarded as a discrete random process with only two possible states: high (one) or low (zero). This process should meet some criteria concerning its randomness:

- The amount of ones and zeros observed should roughly be equal. It means that all values in the Probability Mass Function (PMF) of the random variable representing one output bit are equal, which is called a uniform PMF. Both a one and a zero have the same chance, equal to one half, of occurring. To asses this uniformity, the bias is defined:

$$e = |P(\text{"1"}) - \frac{1}{2}| = |P(\text{"0"}) - \frac{1}{2}|, \tag{2.1}$$

  with $P(\text{"1"})$ and $P(\text{"0"})$, the chance of obtaining a one or a zero at the output respectively. In practice, they are estimated as:

$$P(\text{"1"}) \approx \frac{\sum_{i=1}^{N} b_i}{N} \quad and \quad P(\text{"0"}) \approx 1 - \frac{\sum_{i=1}^{N} b_i}{N},$$

with $b_1, b_2, \ldots, b_N \in \{0, 1\}$, the sequence under test.

- When large amounts of data are collected, there should be no deterministic patterns visible in the data. It means that there is no correlation between different output bits. At every moment in time, the PMF should remain constant, independent from the previously generated data. A measure for this property is given by the autocorrelation with shift k:

$$c(k) = \frac{\frac{1}{N}\sum_{i=1}^{N-k}(b_i - p_1)(b_{i+k} - p_1)}{\frac{1}{N-1}\sum_{i=1}^{N}(b_i - p_1)^2}, \tag{2.2}$$

with $b_1, b_2, \ldots, b_N \in \{0, 1\}$, the sequence under test.

All results showing bias and autocorrelation in this thesis are calculated according to the definitions given above.

It should be highly infeasible to distinguish the data generated by the RNG from a true random binary sequence and almost impossible to predict future data with significant probabilities. Different structures exist that can meet these requirements. Most of them can be divided into two groups [24]: the Pseudo Random Number Generators and the True Random Number Generators.

2.1.1 Pseudo Random Number Generators

A PRNG is a device or an algorithm that can output data that seems highly random and is generally computationally infeasible to predict. The underling functionality is however always deterministic and thus also reproducible. The working of a PRNG relies on two elements that should remain secure, otherwise the quality of the output data can become compromised:

- **The initial seed value**: this seed is the source of randomness for the output data. It must remain secret to the outside world. The PRNG expands the seed into a sequence many times longer than the length of the original seed value. This seed is most of the time created by a TRNG.

- **The algorithm**: this algorithm describes how the output data should be created out of the initial seed value. It must be non-invertible, meaning that given the output values, neither the secret seed nor the internal state can be reconstructed. The algorithm is implemented in some sort of Final State Machine (FSM), so it depends on an internal state. Because this state is only represented by a finite amount of bits, after a while it repeats itself. The output data is therefore periodic and the initial seed should be refreshed before this repetition happens.

A general architecture of a PRNG is depicted in Fig. 2.1, as in [25]. PRNGs are especially designed to meet the criteria of seemingly high randomness in the output data. The output often has good statistical properties (uniform and low correlation)

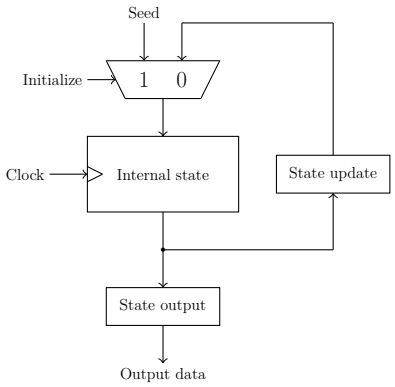


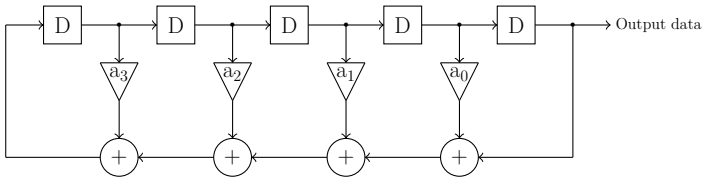
Figure 2.1: General architecture of a PRNG.

and a relatively high throughput, but it should be pointed out that when the initial seed value and the algorithm are known, theoretically this system can perfectly be replicated and it is possible to predict future data.

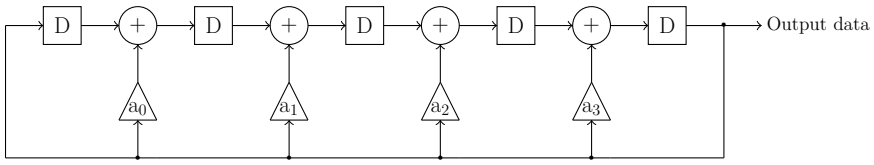
A popular topology for PRNGs is the Linear Feedback Shift Register (LFSR). As described in [26], it is a shift register where some internal bits (taps) are used to generate the next input bit for the shift register. For the ease of implementation, only XORs are used to create the feedback function. The output bit is then used as the output of the PRNG. The working of a LFSR is determined by its feedback polynomial, which is a polynomial modulo 2 in a finite field arithmetic. When this feedback polynomial satisfies some property (it is a primitive polynomial), then it can be proven that the LFSR is maximum-length. This property means that the LFSR cycles through every $2^n - 1$ states. The state where all bits are zero is not possible, because then the LFSR gets stuck in that state. For every LFSR there exists a Fibonacci and a Galois implementation. The Fibonacci is the straightforward implementation, while the Galois implementation has the advantage of a shorter critical path. It is proven that both implementations give the same output. Both implementations are visualized in Fig. 2.2.

2.1.2 True Random Number Generators

The output data from a TRNG should be random in the true sense of the word. Even if the internal working of the TRNG is known, it should still be impossible to predict future data. Two identical TRNGs, in identical circumstances still output completely different data. This difference is because the internal working is based on analog noise sources, which, if designed correctly, are by definition unpredictable. The generic topology of a TRNG consists of a source of entropy, a digitization unit which converts the analog entropy noise source to a usable bit stream and a post-processing function to increase the entropy density of the generated random data [25]. This topology is depicted in Fig. 2.3. On-line testing is needed to monitor the quality of the raw output from the digitization module. It triggers an alarm to notify the user of the random data in case of reduced randomness. More about on-line testing is



(a) Fibonacci implementation



(b) Galois implementation

Figure 2.2: Fibonacci- and Galois implementation for a LFSR.

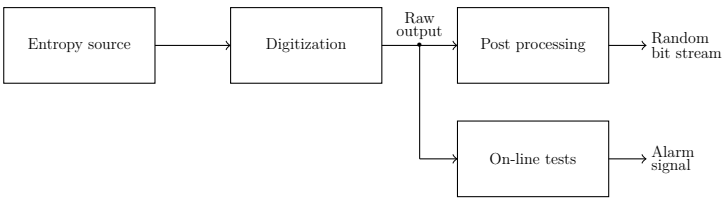


Figure 2.3: General topology of a TRNG.

given in Chap. 8.

Entropy source

In this part, an analog source of randomness is created. This structure is implemented in such a way, that the noise is amplified and prepared to be sampled by the digitization unit. The specific implementation is highly dependent on the type of TRNG. These different types are presented in Sect. 2.3.

Digitization

This part samples the analog noise source and converts it to a more usable format. It determines the sampling speed and the throughput of the raw bit stream. A trade-off can be made between throughput and entropy density (the amount of randomness contained into one single bit). The precise implementation of this component is very dependent on the type of TRNG.

Post processing

In some cases, the quality of the raw bit stream does not have a high enough entropy density. It can be biased (the PMF of the generated data is not uniform) or the

Table 2.1: Input-output relation of the Von Neumann Corrector.

Input	Output
00	-
01	1
10	0
11	-

output data can be correlated. In this case, post processing is added to remove these statistical weaknesses. As a consequence, the throughput is lowered.

Many post processing algorithms exist, two of them are:

- **The Von Neumann Corrector** [27]. This algorithm completely removes the bias from the random data. It groups the output bits into groups of two bits and it only outputs a bit if two bits in a single group are different. The working is visualized in Table 2.1. In the uncorrelated case, the chance of getting an output bit is given as: $P(\text{"01" } or \text{ "10"}) = 2(\frac{1}{2} - e)(\frac{1}{2} + e) = 2(\frac{1}{4} - e^2)$, with e representing the bias of the raw bit stream: $P(\text{"1"}) = \frac{1}{2} + e$ and $P(\text{"0"}) = \frac{1}{2} - e$. The probability of getting an output bit is then at most equal to $\frac{1}{2}$ for every group. But a group consists out of two bits, so the output rate gets decreased by an additional factor of two. The output throughput is then at most only a fourth of the initial throughput.

- **Linear codes corrector** [28]. In this algorithm, the output bits are also grouped and for every group, exactly one bit is generated. This generation is based on a linear code. The simplest example is just grouping the raw bits in pairs and XORing every pair to generate a new bit. This XORing reduces the throughput by a factor of two and reduce the bias to $2e^2$, with e also the input bias. More general XORing n consecutive bits reduces the throughput by a factor of n and reduce the bias to $2^{n-1}e^n$.

  A better approach would be to use more complicated codes. In [29], they show that by using the linear code:

$$L : GF(2)^8 \times GF(2)^8 \rightarrow GF(2)^8$$

$$L(X, Y) = X \oplus (X \ll 1) \oplus (X \ll 2) \oplus (X \ll 4) \oplus Y,$$

  the bias is reduced to 2^4e^5 and the throughput by a factor of two. This code is many times better than only XORing the data. E.g., a bias of 1 % is reduced to 0.02 % by an XOR operation and to 0.00000016 % by the linear code shown above. In [29], it is also stated that linear codes can achieve higher throughput than a Von Neumann corrector.

Care has to be taken with the correlation of the data. Both linear codes and the Von Neumann Corrector can even increase the bias in case of high a dependency between the generated bits.

2.2 Origin of randomness

All true randomness in integrated circuits (ICs) has its origin at the physically uncontrollable stochastic processes at the quantum scale inside the semiconductor. TRNGs should only utilize processes that have already been thoroughly studied and classified. Here follows a short summary of the different classes as presented by [30]:

- **Thermal noise**: due to the thermal motion of electrons in a resistive channel, an electric potential is created. It is directly proportional to temperature and is presented in any resistive channel such as regular passive resistors or the channel of a MOSFET.
- **Shot noise**: this noise is due to the quantization of the carriers which make up a current. It is presented in diodes, MOS- and bipolar transistors.
- **Flicker noise**: traps, that capture and release carriers in a random way, induce noise on a current flow. These traps are caused by defects or contaminations in the crystal. It has a spectral density that is inversely proportional to the frequency and it mainly affects active devices such as transistors.
- **Burst noise** (Popcorn noise): it is caused by the presence of heavy metal contaminations inside the semiconductor crystal. They generates noise pulses in the audio frequency range and is perceived as a popping noise when played through a loudspeaker.
- **Avalanche noise**: a strongly reverse biased pn junction can create an avalanche breakdown i.e., a random series of large current peaks. As with shot noise, avalanche noise is also related with a current flow, but the current spikes are much greater in magnitude.

2.3 Types of True Random Number Generators

There already exists a wide range of TRNGs. They can be classified into two groups. First, there are designs based on noise in analog components. These structures exploit sources of noise presented in analog building blocks such as resistors [31], chaotic oscillators [32], emitter-coupled pairs [33], etc. Second is the group containing only digital components readily available in digital libraries and Field Programmable Gate Arrays (FPGAs). The advantage of using only digital components is that they often offer better scalability, lower power consumption, lower cost and easier design in general. For these reasons, only the second group is further discussed in this text. However keep in mind that essentially the source of randomness in both groups originates from the same noise principles, handled in Sect. 2.2.

According to [16], the group of digital TRNGs can further be divided into two main classes. Those that rely on jitter between two free running oscillators and those that rely on a metastable event of a memory element. A clarifying presentation of the different architectures is given in Fig. 2.4. Both classes are further discussed below.

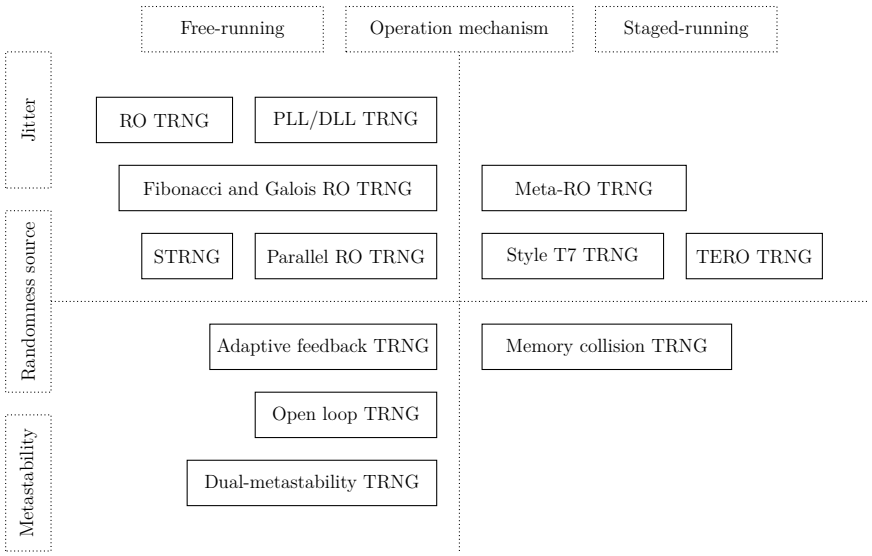


Figure 2.4: Classification of the different types of TRNGs, based on [16].

2.3.1 Phase noise-based TRNGs

Ring Oscillator (RO) TRNG

The first and very widely used architecture was originally proposed by [3]. It uses the difference in accumulated jitter between two ROs in its advantage. A RO is a chain containing an odd number of inverter logic gates, with the output fed back to the input. When two identical ROs on the same IC are initiated perfectly in phase and are left running, they won't stay in phase, their rising edges won't happen at exactly the same moment in time. This difference is due to the stochastic processes happening inside the semiconductor where the ROs are fabricated in. Notice that even this simple hypothetical example is not possible in the real world. Two identically designed ROs are in practice never perfectly identical. Random process variations cause one RO to have a slightly larger period than the other. This variation eventually gives a deterministic component in the phase difference. Note that this component does not contribute to the randomness of the output, because when measured, it is perfectly predictable.

A generic architecture for this principle is depicted in Fig. 2.5. One RO is used as a clock to sample the other RO using a D-Flip Flop (DFF). Notice that this principle only generates an unpredictable output when the RO is sampled in its jittery interval very close to an edge of its output signal. This interval is very small and therefore the event of sampling a jittery bit is very rare. To cope with this problem, many of these ROs are placed in parallel and are all sampled by a common clock. This architecture was proposed by [34]. It should be noted that it is never certain that even only one RO is sampled in its jittery interval. Therefore they provide tables giving the minimal amount of ROs needed for a certain reliability. Another problem is the possibility that these many ROs lock. This effect causes all the ROs to have an edge

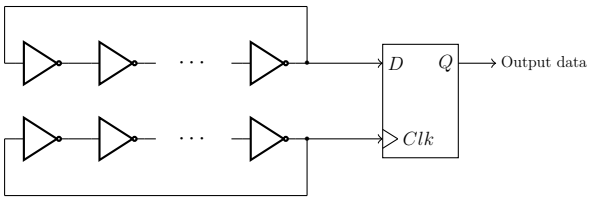


Figure 2.5: Generic architecture for the RO TRNG.

very close to each other. It reduces the jittery interval significantly and lowers the chance of sampling a random bit. The origin of this effect can be either from inside the IC or from the external world. In the first case, current spikes due to switching of one RO can cause voltage fluctuations in the supply lines. The other ROs react to these fluctuations by synchronising to the fluctuations and locking to the same phase. In the second case, external Electromagnetic (EM) radiation impinging at the IC that has a frequency inside the locking range of the ROs, also causes the ROs to lock. This method has already been used by [7] and [8]. They generated EM waves in close proximity of a TRNG that was based on ROs. Using this technique, they successfully reduced the randomness of the output data. A solution for the first case of locking can be found in placing on-chip decoupling capacitors. These capacitors smooth out the voltage fluctuations at the supply lines. A solution for the second case would be to construct an EM shield that absorbs the impinging EM radiation.

The oscillating signals can also be created by Phase Locked Loops (PLLs). Here, multiple oscillating signals of frequencies that are an integer multiple of each other are already available. The low frequency signal can then be used to sample the faster oscillating signal and generate random bits based on the difference in jitter. Note that PLLs are essentially hybrid structures containing both analog and digital components. It means that they are not always available in digital-only designs.

The RO TRNG can achieve high throughputs, but this high throughput comes at a cost of very large area requirements.

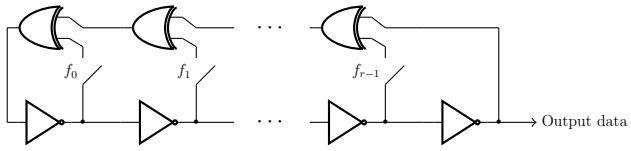
Fibonacci and Galois RO TRNG

Fibonacci and Galois RO TRNGs are described by [35]. A short summary of its working is repeated here: regular Fibonacci and Galois LFSRs are used where the memory elements of the register are replaced by an odd number of inverter logic gates. The principle is depicted in Fig. 2.6. When a switch is open, the XOR-gate is not present. The working of these ROs can also be described by its feedback polynomial. This polynomial is defined as:

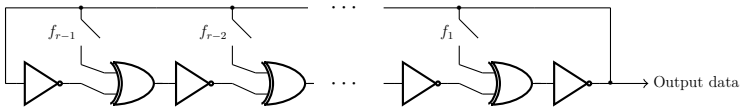
$$f(x) = \sum_{i=0}^{r} f_i x^i, \quad f_0 = f_r = 1,$$

with $f_i = 1$ if a feedback connection is present and $f_i = 0$ otherwise. The author proves that if this polynomial can be written as:

$$f(x) = (1 + x)h(x),$$



(a) Fibonacci RO TRNG



(b) Galois RO TRNG

Figure 2.6: Architecture of the Fibonacci- and Galois RO TRNG.

with $h(x)$ a primitive polynomial, then these ROs have a state transition diagram witch contains only one long cycle of length $2^r - 2$.

These ROs benefit from the same properties concerning jitter as regular ROs, but they have the additional advantage of also generating a pseudo random signal. The author suggests combining both a Fibonacci RO together with a Galois RO using an XOR gate. After that, the output from the XOR is sampled using a DFF. The generated data contains a true random component, but it suffers from a large bias and additional post processing is necessary.

Transition Effect Ring Oscillator (TERO) TRNG

The TERO TRNG was proposed by [4]. The architecture is given in Fig. 2.7. It consists out of a loop with an even number of inverter logic gates together with some non-inverting gates. Because the even number of inverting elements, this loop has, in contrast to a regular RO, two stable states. Thanks to the AND gates, the loop can however be brought in a metastable state. In this state, two edges start racing through the loop and after some time, the faster edge catches up with the slower one. This difference in speed can be due to random process variations, favouring either the positive transition or the negative one. After this phase, the loop returns back to one of its two stable states. The time it takes to get back to this stable state is the source of randomness for this TRNG. Its length is measured by counting the number of times the positive edge passes at some point in the loop. In most cases, only the Least Significant Bit (LSB) of this number is used as the random output bit, but in case of large variations, even more bits can be used.

The TERO TRNG can achieve reasonably high throughputs while not occupying large amounts of area.

Self-Timed Ring (STR) TRNG (STRNG)

STRs are explained in more detail in Sect. 2.4.2. For now, it can be seen as a kind of RO, where multiple edges (events) propagate independently through the ring. The

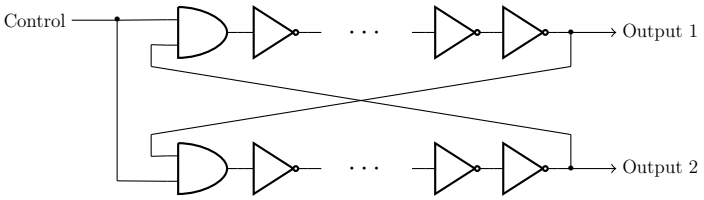


Figure 2.7: Architecture for the TERO TRNG.

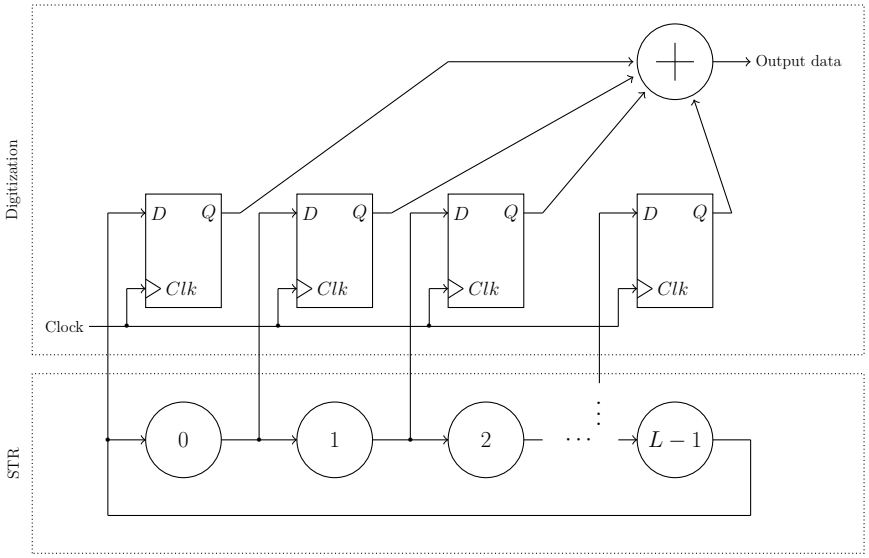


Figure 2.8: Architecture for the STRNG, based on [17].

architecture is depicted in Fig. 2.8. A handshake protocol ensures that these events cannot collide. Analog effects then make sure that these events are spread evenly over the ring. The difference in phase of these multiple events can in contrast to ROs be precisely controlled. When this phase difference is made smaller than the edge-jitter magnitude, it is certain that when all these signals corresponding to the different stages are sampled, at least one is in its jittery interval and the generated output data is unpredictable. This control over the phase is the main advantage over the RO TRNG, where it was never absolutely certain that the generated data was truly random and not only pseudo random. This principle was proposed by [10]. They provide formulas to determine the length of the STR, to ensure that the phase resolution of the different events is smaller than the length of the jittery interval.

This type of TRNG can generate data at a very high throughput. In [10], it is claimed that when the ring is chosen to be sufficiently long, the randomness of the generated data is independent of the frequency of the sampling clock. This independence means that in theory, the throughput could be infinite. The sampling speed is however limited by the maximal system clock and the clock period should also be longer than the length of the jittery interval. Otherwise, the same random bit is sampled twice. It should be noted that also for this TRNG, the large throughput

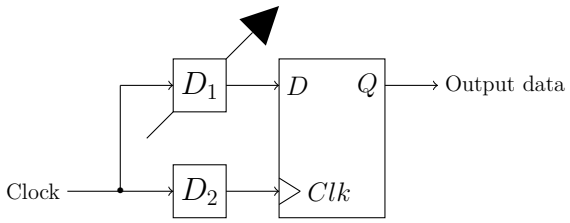


Figure 2.9: Architecture for the open loop TRNG, based on [18].

comes at the expense of a large area impact. To have a sufficient amount of randomness, a ring with a length of more than 500 stages is necessary.

2.3.2 Metastability-based TRNGs

Open loop TRNG

This design was proposed by [1]. The architecture is depicted in Fig. 2.9. The main purpose of this TRNG is to sample a DFF into its metastable interval. When an edge of the data signal arrives very close before or after the positive edge of the clock signal (in the range $[-t_{setup}, t_{hold}]$, when the positive edge of the clock signal arrives at time zero), the output of the DFF becomes undefined for a certain amount of time. When this metastability is resolved, the output converges in an unpredictable but biased way to one of the two supply rails. This metastable time interval is however very small, cautious signal routing is needed to generate the highly synchronised clock and data signals. Therefore, the authors of [19] make use of course and fine delay chains, see Fig. 2.10. The course delay chains are used to roughly synchronise the clock and data signals. After this course synchronization, the signals are fed into the fine delay chains, these chains consist out of a fixed number of delay elements. After every element, the data gets sampled by a DFF. The outputs of all these DFFs are then combined using an XOR gate, to generate an output bit. The purpose of this fine delay chain is to cope with the possible temperature and supply variations. When this fine delay chain is long enough, at least one DFF is always sampled in its metastable time interval and the output bit is truly unpredictable.

Dual metastability TRNG

This type of TRNG does not use the unpredictable output of a metastable DFF as its source of randomness. It uses the time it takes to resolve the metastable event. This architecture was proposed by [2], it is depicted in Fig. 2.11. The authors state that the problem of only using the outcome state of the metastable event introduces a large amount of bias. The reason for this bias are the process variations that make a DFF favour one of its two stable states when resolving the metastability. Here, this problem is solved by not using the outcome, but by using the propagation time of the DFF when it is driven close to metastability. The authors also claim that truly driving a DFF in metastability is very difficult. To handle variations in supply

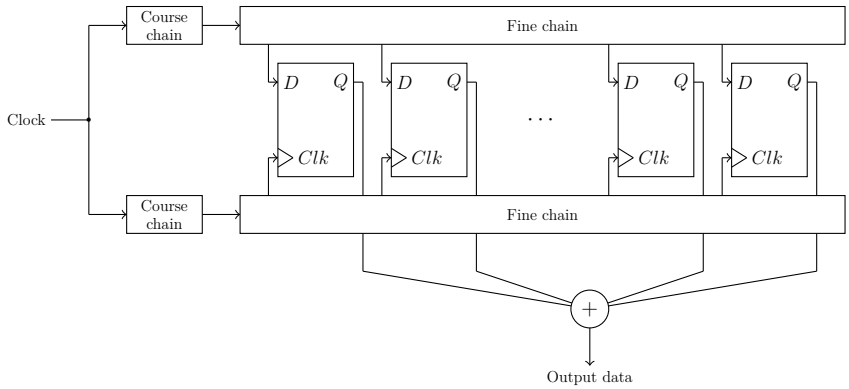


Figure 2.10: Architecture for the open loop TRNG with fine delay chain, based on [19].

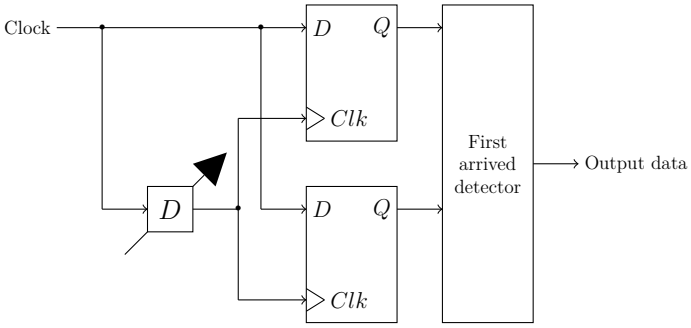


Figure 2.11: Architecture for the dual metastability TRNG.

and temperature, the propagation delays of two DFFs are compared and a random bit is generated by selecting the fastest DFF.

2.4 CS based STRNG

This design combines the known technique of generating multiple oscillating signals with independent jitter by using a STR, with the coherent sampling (CS) technique to extract unpredictable binary data from the jittery signals. This idea was recently proposed by [11]. These two techniques are explained in more detail in the following sections.

2.4.1 Combination of STR with CS

The combination brings together the advantages provided by the STR topology and efficient sampling of the CS scheme. Thanks to the independence of the jitter of every stage's output, each of them can function as an input signal for a separate CS unit. Two identical STRs of length L are used, where the corresponding stages are used to drive L different CS-units. This architecture is depicted in Fig. 2.12.

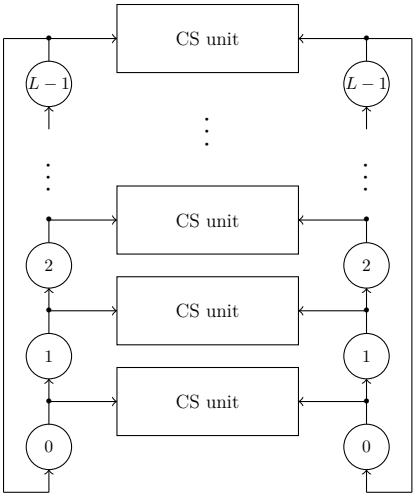


Figure 2.12: Architecture of the complete TRNG, consisting out of two STRs and L CS units.

This topology offers a reasonable throughput for only limited area requirements, which is ideal for demanding applications such as in the IoT.

2.4.2 Self-Timed Rings

The STRNG was originally proposed by [17] and its stochastic model by [10].

Working and architecture

A STR is made of a sequence of stages, where the last stage is connected back to the input. Such a stage is made out of a Muller C-element and an inverter logic gate and is illustrated in Fig. 2.13. It was proposed by [36]. A Muller C-element is a gate with two inputs and one output. When both inputs are equal, the output is the same as the inputs. When the inputs disagree, the output is remembered and kept at the previous output. To make an oscillator with this gate, an additional inverter logic gate is needed. Together, they form one STR-stage. Its transition table is given in Table 2.2. These stages are connected according to the following rules:

$$F_i = C_{i-1 \bmod L}, \quad with \quad i = 0 \ \ldots \ L-1,$$

$$R_i = C_{i+1 \bmod L}, \quad with \quad i = 0 \ \ldots \ L-1.$$

With F_i and R_i the non-inverted (forward) and the inverted (reverse) inputs of stage i respectively. C_i is the output of stage i and L is the number of stages in the ring. This architecture is depicted in Fig. 2.14. The output of a stage is only allowed to change if it meets the conditions below:

$$C_i \neq C_{i-1 \bmod L}, \quad with \quad i = 0 \ \ldots \ L-1,$$

$$C_i = C_{i+1 \bmod L}, \quad with \quad i = 0 \ \ldots \ L-1.$$

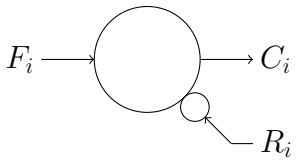


Figure 2.13: One STR-stage, containing a Muller C-element (large circle) and an inverter logic gate (small circle).

Table 2.2: Transition table for one STR-stage.

F_i	R_i	C_i
0	0	C_i^{-1}
0	1	0
1	0	1
1	1	C_i^{-1}

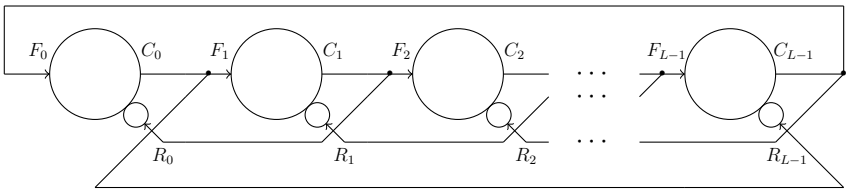


Figure 2.14: Architecture of a STR, consisting out of L stages.

The output of stage i needs to be different than the output of the previous stage and the same as the output of the following stage. The output toggles and this event then propagates through the rest of the ring.

To simplify the analysis, the working of a STR can also be described in terms of tokens and bubbles. Each stage contains either a bubble or a token. If its output is not equal to the output of the previous stage, it holds a token, otherwise a bubble. When the conditions for an event to propagate are then described in terms of tokens and bubbles, it becomes a lot simpler. A token at a stage is only allowed to move forward in the ring if the next stage contains a bubble. In the next time step, the token and bubble are then interchanged. With this principle, tokens move forward in the ring and bubbles backwards. By only interchanging the tokens and bubbles, the amount of tokens and bubbles always remains the same and they must sum up to the length of the ring, L. Due to its definition, the number of tokens is always even. The movement of tokens and bubbles is illustrated in Fig. 2.15. Here, "T" and "B" represent a token and bubble respectively. The bits indicate the output value of each stage. The possible token-bubble interchanging pairs are inside a dotted rectangle and the activated token-bubble interchanging pair is inside a solid rectangle.

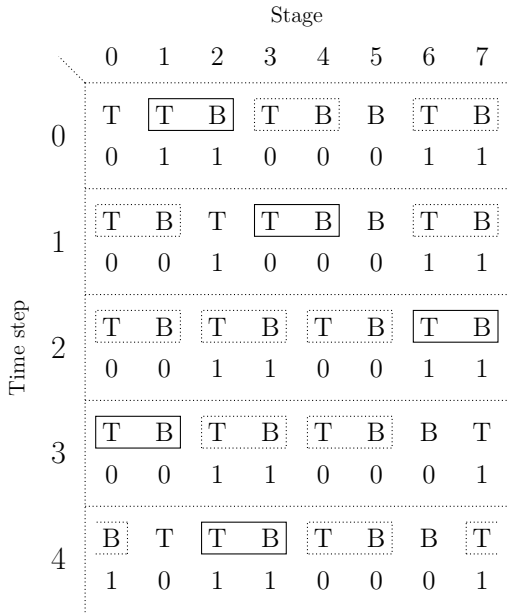


Figure 2.15: Movement of tokens and bubbles in an eight stage STR, based on [20]

Temporal properties

The temporal properties of a STR are very well described by [20]. They analyse the timing of the different events propagating through the ring. There are two modes that can be distinguished: evenly spread mode and burst mode. In the former and often wanted case, the timing between the different events is the same. They are spread nicely around the ring. In the second case, the events tend to stick together leading to a fast burst of events and a long period of no action. These two behaviours can be explained by two counteracting analog effects. The STR operating mode is determined by the relative strength of these two effects.

The Charlie effect This effect tries to push events out of each other. The authors of [20] describe this effect by the analog representation of a Muller C-element, depicted in Fig. 2.16. If for example both the a- and the b-input have a rising edge and a changes a long time after b, then the pMOS and nMOS connected with the b-input are fully off and on respectively. This state enables a fast reaction of the output to a change of the a-input. In the other case, when a and b arrive very close to each other, the output has a greater delay because the transistors connected with the b-input are still partially conducting.

Mathematically, this effect can be visualized in a Charlie diagram, [14]. An example of such a diagram is given in Fig. 2.17. The horizontal axis represents the event separation and the vertical one, the effective delay of the stage. These are given as:

$$m = \frac{t_F + t_R}{2},$$

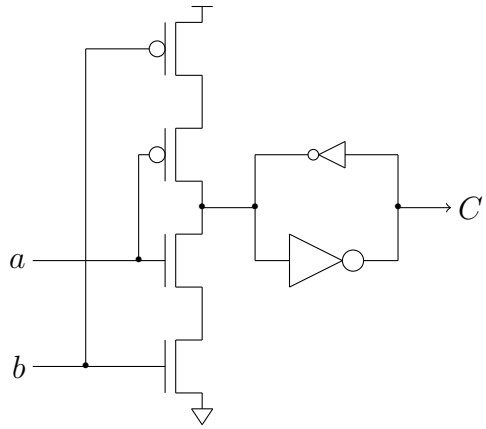


Figure 2.16: Analog representation of a Muller C-element, based on [20].

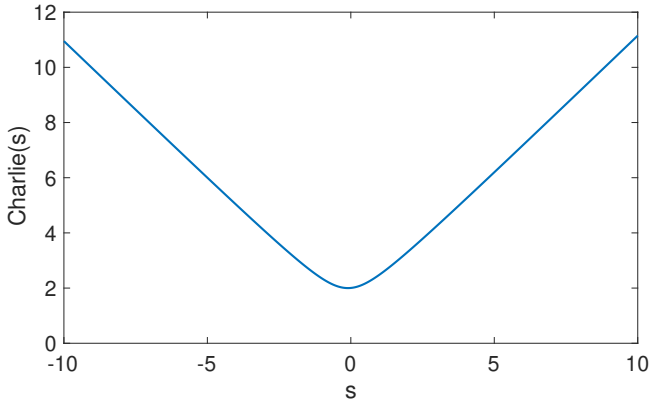


Figure 2.17: Example of a Charlie diagram, based on [20].

$$s = \frac{t_F - t_R}{2},$$

$$t_C = m + Charlie(s). \tag{2.3}$$

Here, t_F and t_R represent the arrival times of the forward and reverse inputs respectively. m is the mean arrival time, s the separation and t_C the time when the output changes. As seen in the Charlie diagram, the term $Charlie(s)$ in the above equations is composed of a fixed part (the delay if the events where infinitely separated), a part that is proportional with the separation (the time equal to the delay of the last arriving input signal to the mean arrival time of the inputs) and a part that is always positive and is at its largest when the separation is zero. This part causes the graph of the effective delay to diverge from its two asymptotes.

The Charlie effect is the main contributor to an evenly spaced mode. When this effect dominates, the events push each other as far as possible until an evenly spread state is obtained.

The Drafting effect This effect takes into account the capacitance at the output node of each stage. When an event just passes a stage, due to the output capacitance, it takes some time for the output voltage to switch from one logic level to the other. Because of this latency, when a second event arrives shortly after the first one, the output voltage is still transitioning. The output is then pulled back to its original logic level. Because this voltage had not yet fully reached the other logic level, the second transition has a reduced delay.

This principle favours a burst of events propagating through the ring. The leading event reduces the delay of consecutive events. The handshake protocol of each stage avoids that events collide into each other.

The 3D Charlie model Both the Charlie and the Drafting effect can be combined into one model. According to [13], this model is determined by five parameters:

- D_{ff}, the static forward propagation delay. It is the delay between the forward input and the output if the separation of the events is long enough for the Charlie effect to be negligible.
- D_{rr}, the static reverse propagation delay. The same delay, but with respect to the reverse input.
- $D_{Charlie}$, the amplitude of the Charlie effect.
- A, the duration of the Drafting effect.
- B, the amplitude of the Drafting effect.

The combined effect is then given as:

$$Charlie(s,y) = D_{mean} + \sqrt{D_{Charlie}^2 + (s - s_{min})^2} - Be^{-\frac{y}{A}},$$

with:

$$y = m - t_C^{-1},$$
$$D_{mean} = \frac{D_{ff} + D_{rr}}{2},$$
$$s_{min} = \frac{D_{rr} - D_{ff}}{2}.$$

The time between the mean arrival time and the previous output change is represented by y. The time when the output changes is then again given by Eqn. 2.3. An example of this model is visualized in Fig. 2.18. Initially, all transitions in the stages represent different points on this surface in the 3D space: $[s, y, Charlie(s,y)]$. When the transient behaviour of the ring disappears and the ring evolves in an evenly spaced mode, all points on the surface converge to one single attractor, which is the wanted case. The authors of [13] claim that this mode is always obtained if:

$$\frac{N_T}{N_B} = \frac{D_{ff}}{D_{rr}},$$

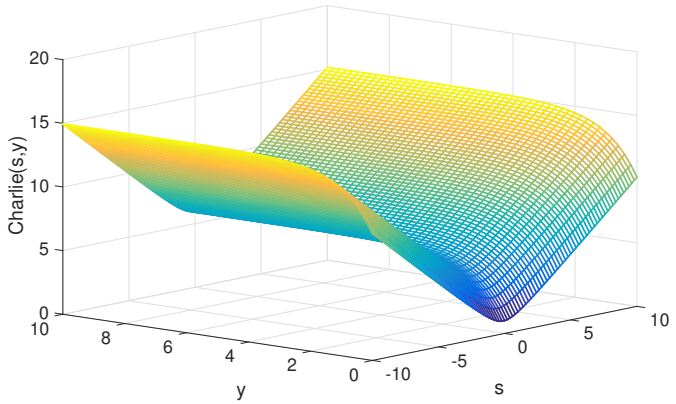


Figure 2.18: Example of a 3D Charlie diagram, based on [20].

with N_T and N_B the number of tokens and bubbles in the ring respectively. In [37], this equation is simplified for FPGAs. The authors claim that by implementing each stage into one Look-Up-Table (LUT) and neglecting the interconnection delays, the forward and reverse static delays are approximately equal. The requirement for a certain evenly spread mode becomes then:

$$N_T = N_B.$$

This requirement is however not always sufficient as proven in further chapters.

Advantages over regular ROs

The biggest advantage of STRs over regular ROs lies in the way the jitter gets accumulated through the ring. A distinction must be made between the purely random Gaussian jitter and the deterministic global jitter. In [37], they calculate and compare these jitter components:

Gaussian jitter This jitter is due to the random processes as described in Sect. 2.2. When implemented in an FPGA, the signals generated by a LUT contain Gaussian jitter. This jitter can then be characterised by its distribution: $\mathcal{N}(0, \sigma_g^2)$, with σ_g^2, the variance of the jitter created by one LUT. This value should be approximately equal for all LUTs in the same FPGA. In the case of regular ROs, during one period of the oscillating output signal, an edge has to propagate through each of the L inverter logic gates, with L the length of the RO. The jitter of each stage then accumulates and the resulting period jitter is then equal to:

$$\sigma_{periodRO} = \sqrt{2L}\sigma_g.$$

Every time a stage receives a token in a STR, its output toggles. The oscillating period of the signals at the output of the stages of a STR is then the time it takes

for two tokens to pass. The period jitter can then be written as:

$$\sigma_{periodSTR} = \sqrt{2}\sigma_g.$$

The Charlie effect maintains the spacing between successive events. In this way, the jitter can't accumulate through the ring and the period jitter at the output of each stage is mostly composed of the jitter generated by that last stage. This conclusion is very important. It means that each of the L outputs of the STR contain independent Gaussian jitter and can therefore be used as separate sources of true randomness. This conclusion was both obtained in [10] and [37].

Deterministic jitter It is the predictable component of the jitter in the oscillating period. It reflects the dependence of the propagation delays with respect to external influences such as temperature or supply voltage. In regular ROs, a change of delay just accumulates, because the edges have to propagate through every stage during one period. With STRs, these effects are also attenuated. The authors of [37] claim that variations in delays get smoothed due to the Charlie effect. This effect makes STRs inherently more robust against variations in operating conditions. In [13], simulations show that STRs are also more robust against process variations than regular ROs.

All these advantages over regular ROs make STRs a very suitable candidate for the use in random number generation.

2.4.3 Coherent Sampling

A CS based TRNG which uses PLLs as a source of jittery signals was proposed by [38]. A mathematical model was described by [39]. CS is a technique that enables a more efficient extraction of random data out of jittery signals than just sampling the signal with a DFF. According to [21], by the traditional sampling technique, the amount of accumulated jitter should be in the order of the length of the period of the sampled signal. It means a large sampling interval is needed to provide high enough entropy, resulting in a lower throughput. With CS, the required amount of accumulated jitter should be in the order of the period difference of the sampled and the sampling signal. This difference can be made very small by matching the two oscillators that generate these oscillating signals, which results in a much larger throughput.

Principles of operation and architecture

The architecture of the simplest form of CS is illustrated in Fig. 2.19. It consists out of four DFFs and one XOR-gate. It has two inputs: the sampled and the sampling signal, S_A and S_B in the figure respectively. The output of the sampling DFF (DFF 1) is a low frequency beat signal (S_{beat}). The period of this signal is longer as the two signals S_A and S_B are longer in phase. It is due to the accumulated phase difference of these two signals (caused by the Gaussian jitter) that the beat signal toggles. The length of the period of this beat signal is then measured by the asynchronous

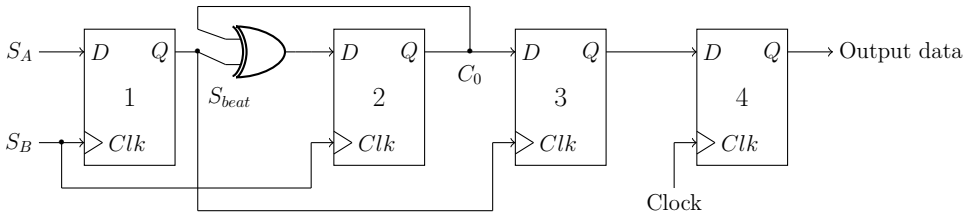


Figure 2.19: Architecture of the simplest form of CS.

one-bit counter (only the LSB is obtained) which is implemented by DFF 2 and the XOR-gate. This combination of XOR-gate and DFF implements a Toggle-Flip Flop (TFF). Due to the random jitter accumulation, the length of the period of the beat signal is random and so is its LSB. After this step, the LSB (C_0) is sampled at every positive edge of the beat signal by DFF 3. The last step is making everything synchronous to the system clock by sampling it again with DFF 4. This clock signal (Clock) determines the throughput.

Stochastic model

A stochastic model based on the frequency difference of the two signals was given by [21]. The principle of CS and the notation are clarified by Fig. 2.20. $T_A^{(k)}$ and $T_B^{(k)}$ are the period lengths of the signals S_A and S_B respectively. These period lengths are independent and have a Gaussian distribution:

$$T_A \sim \mathcal{N}(\mu_{T_A}, \sigma_{T_A}^2),$$

$$T_B \sim \mathcal{N}(\mu_{T_B}, \sigma_{T_B}^2).$$

Take then N_i equal to the amount of rising edges of the signal S_B from time zero to the i^{th} beat period:

$$N_i = min\{k | Y_k > X_{k+i}\},$$

with:

$$Y_k = T_B^{(0)} + T_B^{(1)} + \ldots + T_B^{(k)},$$

$$X_k = T_A^{(0)} + T_A^{(1)} + \ldots + T_A^{(k)}.$$

So every beat means that the signal S_A has had one more rising edge than the signal S_B. Then the beat counter is defined as:

$$R_i = N_i - N_{i-1}.$$

R_i then contains the number of rising edges of signal S_B during the i^{th} beat period. The period difference is then defined as:

$$\Delta_i = T_B^{(i)} - T_A^{(j)},$$

with initially $j = i = 1$, but j gets incremented one more than i each period of the beat signal. So Δ_i represents the period length difference of S_A and S_B that occur at the same moment in time. [21] states then that Δ is also normal distributed:

$$\Delta \sim \mathcal{N}(\mu_\Delta, \sigma_\Delta^2),$$

with:

$$\mu_\Delta = \mu_{T_B} - \mu_{T_A},$$
$$\sigma_\Delta = \sqrt{\sigma_{T_A}^2 + \sigma_{T_B}^2}.$$

In [40], they provide then the equations to describe the distribution of R:

$$R \sim \mathcal{N}(\mu_R, \sigma_R^2),$$

with:

$$\mu_R = \frac{\mu_{T_A}}{\mu_{T_B} - \mu_{T_A}},$$
$$\sigma_R = \sqrt{\frac{\mu_{T_A}}{\mu_\Delta}}\frac{\sigma_\Delta}{\mu_\Delta}.$$

Out of this distribution, the amount of entropy in the LSB can then be calculated. First of all $\mu_R \gg \sigma_R$, the chance of obtaining a negative value for R is negligible. R can only be an integer value, so the normal distribution is only approximate. The LSB is a "1" if and only if R is odd and a "0" otherwise. The probabilities of obtaining a "1" or a "0", $P(\text{"1"})$ and $P(\text{"0"})$ can then be estimated by integrating the odd or even parts under the Probability Density Function (PDF) of R. This principle is illustrated in Fig. 2.21. Integration of the darker areas leads to $P(\text{"1"})$ and the lighter areas to $P(\text{"0"})$. The amount of Shannon entropy can then be estimated by the binary entropy function:

$$H(\mu_{T_A}, \mu_{T_B}, \sigma_{T_A}, \sigma_{T_B}) = -P(\text{"1"})log_2(P(\text{"1"})) - P(\text{"0"})log_2(P(\text{"0"})).$$

This function is visualized for different values of $\mu_{T_A}, \mu_{T_B}, \sigma_{T_A}$ and σ_{T_B} in Fig. 2.22. Three curves with each a different coefficient of variation (CV), defined as: $\frac{\sigma_{T_A}}{\mu_{T_A}}$ ($\sigma_{T_A} = \sigma_{T_B}$) are plotted against the ratio of the two periods: $\frac{\mu_{T_B}}{\mu_{T_A}}$. The curves show a high entropy for a period ratio close to unity. A resonance effect is also visible for larger period ratios. A peak happens if the mean CS count (μ_R) is exactly between two integers. When it is the case, the entropy is always equal to one, even if the CV is small. this effect is because the integration of the different parts under the Gaussian is then perfectly symmetrical.

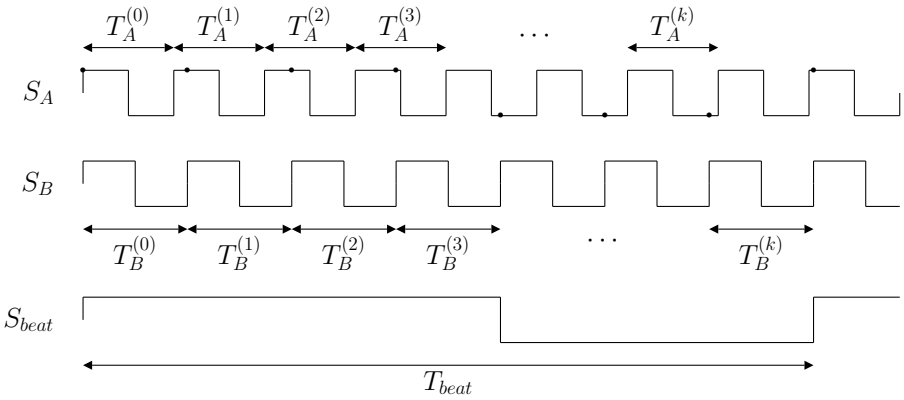


Figure 2.20: Principle and notation of the CS theory, based on [21].

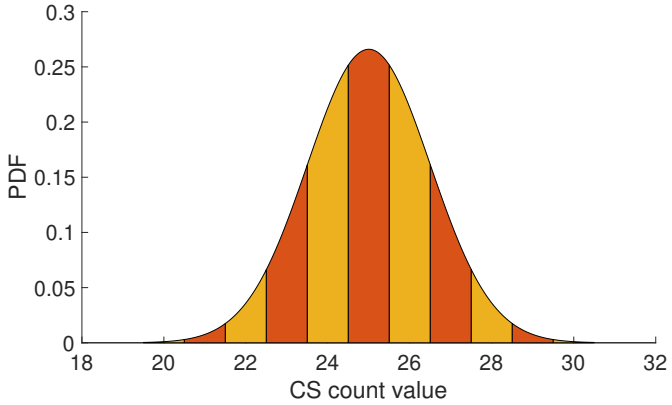


Figure 2.21: Calculation of the probabilities: P("1") and P("0").

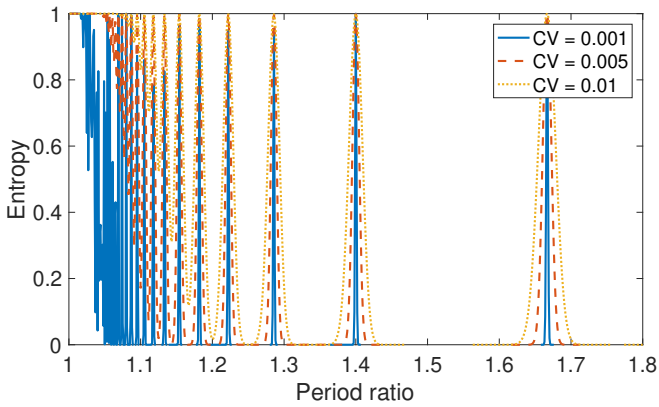


Figure 2.22: Calculation of the binary Shannon entropy function.

Chapter 3

Modelling and simulating the TRNG

To better understand the behaviour of the proposed TRNG and predict the possible side effects on the performance due to unwanted parasitics and potential attacks, a high-level model of the STRs and the CS-units is created in *MATLAB*. This model can reveal what it takes to bring a STR out of the evenly spread mode, which is the mode of operation assumed in the security analysis of [10]. They refer to the conclusion in [37], that due to this evenly spaced mode the jitter at each stage's output is independently originated from that same stage. This conclusion enables them to use the theory of micropipelines [14] (Charlie diagram from Chap. 2). However, when this assumption does not hold, due to an imperfect implementation or a change in operating conditions, these security analyses do also not hold any more and true randomness cannot be guaranteed. This consequence makes the TRNG unsuited according to the AIS 31 standards, which state that a valid stochastic model should be available [6]. In the majority of the literature, they assume the condition of evenly spread mode is almost certainly obtained. In reality however, routing asymmetries are unavoidable in FPGAs. That is why it so important to go into these effects more deeply.

The first section describes the model and its parameters. The second section presents the performed simulations and its results. At the end, an overall conclusion is given.

3.1 Modelling

3.1.1 STR-model

Parameters The STR-model is largely based on the model proposed in [13]. Originally, it was based on the five parameters from the Charlie diagram, explained in the previous chapter. These parameters are:

- D_{ff}, the static forward propagation delay.

- D_{rr}, the static reverse propagation delay.
- $D_{Charlie}$, the amplitude of the Charlie effect.
- A, the duration of the Drafting effect.
- B, the amplitude of the Drafting effect.

A sixth parameter is added to cope with the random delay of the LUTs that make up the stages of the STRs. This parameter is called the gate jitter magnitude: σ_G. The delay of each stage consists then out of a deterministic part (given by the Charlie diagram) and a random part (given by the gate jitter: $\sim \mathcal{N}(0, \sigma_G^2)$).

A seventh parameter is added to account for the routing delays. In theory, they could just be added to the static forward and reverse propagation delays. This reasoning is however not completely correct. Due to the limited routing capabilities in an FPGA, the routing delays are not exactly the same for every gate. Therefore, a list with forward and reverse routing delays (one forward and reverse delay for each gate) should be added to the model. The static forward and reverse propagation delays are taken equal and the same for each stage. This simplification was proposed by [37]. The reasons for it are: one, each state is built out of an identical LUT (as is described in more detail in Chap. 5) and two, both the forward and the reverse input are inputs to the LUT, so the delay from both inputs to the output of the LUT can be taken as equal.

Numerical value of the parameters To simulate the high-level model, numerical values should be assigned to these parameters. Note that this model is only used to measure the relative magnitude of the effect of variations of these parameters. Therefore, the absolute accuracy of the simulation results is of less importance and only a rough estimate of the parameters is necessary. These estimates are obtained as follows, the numerical values are given in Table 3.1:

- D_{ff} and D_{rr}: from the data-sheets of the FPGA used for the implementation (*Xilinx Spartan 6* FPGA, [41]), the maximal propagation delay from the input to the output of a LUT can be obtained.
- $D_{Charlie}$: this parameter is not given directly in any data sheet. However in [13], they calculate the parameters for the Charlie model according to simulations with the *65 nm STMicroelectronics* technology. The values they obtained for D_{ff} and D_{rr} are smaller than these given in the *Spartan 6* data sheets, but to get a sense of the magnitude of $D_{Charlie}$, the value they obtained can be scaled to match the values from the *Spartan 6* data sheet.
- A and B: according to [37], the drafting effect is very low in FPGA devices and can be neglected for this model.
- σ_G: the gate jitter magnitude is measured for FPGA devices in [37]. The FPGA used to measure the gate jitter magnitude was an *Altera Cyclone III* device.

Table 3.1: Numerical values for the parameters of the high-level STR model.

Parameter	Value
D_{ff} and D_{rr}	0.26 ns
$D_{Charlie}$	21 ps
A	1 s
B	0 s
σ_G	2.5 ps
Routing delays forward	[0.65 ns, 0.149 ns, 0.337 ns, 0.506 ns, 0.492 ns, 0.509 ns, 0.303 ns, 0.196 ns]
Routing delays reverse	[0.254 ns, 0.309 ns, 0.15 ns, 0.58 ns, 0.479 ns, 0.28 ns, 0.143 ns, 0.854 ns]

- Routing delays: the i$^{\text{th}}$ element in the list of Table 3.1 indicates the delay of the line arriving at the i$^{\text{th}}$ stage in the ring. These values where obtained after the rings where implemented on the FPGA. A worst case estimate for the delay for each net is given in the FPGA editor from the *Xilinx ISE* design suite.

Working The model is implemented in *MATLAB*. It keeps a vector with the most recent time instances when each of the gates of the ring has toggled. It then computes from these time instances and the Charlie model, the time instances when the stage toggles again. After this step, it selects the stage with the earliest next change and updates the vector of the recent toggle time instances. Then this procedure is repeated again until some maximal simulation time is reached.

3.1.2 CS-model

This model does not have a lot of parameters. It just implements the CS hardware described in the previous sections in *MATLAB*. As input, it takes the two oscillating signals obtained from the STR-model. The only parameter for this model is the sampling speed. It was taken at 0.976 562 5 MHz, equal to the real sampling speed obtained after implementing the TRNG in Chap. 5.

3.2 Simulating

3.2.1 Effect of the sampling speed

As a first simulation, the effect of the sampling speed is studied. As already explained, increasing the sampling speed of the CS units raises the chance of sampling the same bit. This increased chance intuitively leads to a higher autocorrelation coefficient and even maybe to a larger bias. This simulation checks if the model can track this effect.

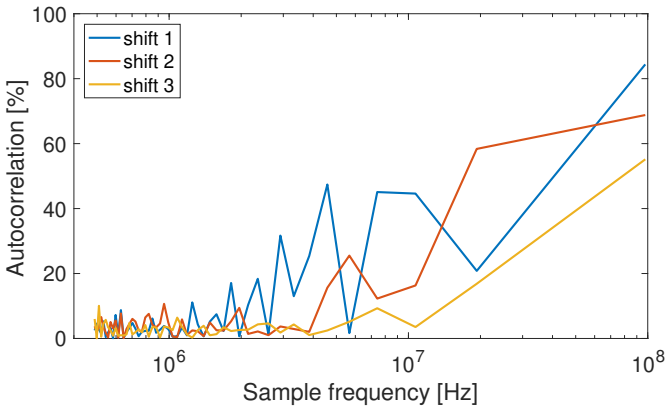


Figure 3.1: Autocorrelation coefficient with a shift equal to one, two and three bits in function of the sampling speed.

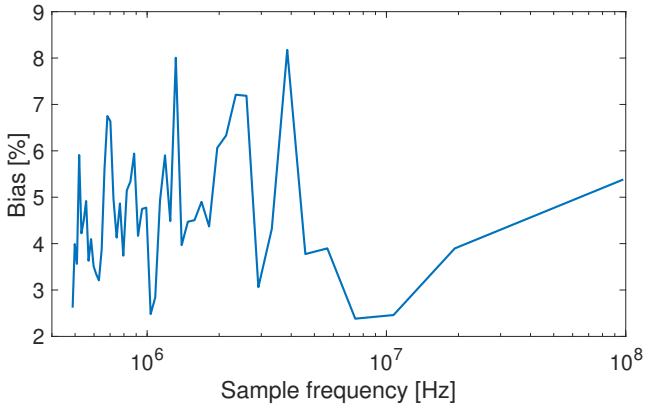


Figure 3.2: Bias of the output data in function of the sampling speed.

Figs 3.1 and 3.2 show the autocorrelation and bias respectively of the simulated output data stream at different sampling speeds. It is clear that the autocorrelation rises with increasing sampling speed. This effect is perfectly in line with the theory. The influence on the bias is less significant, no real trend is visible.

3.2.2 Effect of asymmetric routing delay

For the second simulation, the effect of asymmetric routing together with the Charlie effect is studied. The authors of [37] claim the interconnection delays for STRs implemented in FPGAs can be neglected. In this way, the requirement for the ring to be in an evenly spread mode becomes:

$$\frac{N_T}{N_B} = \frac{D_{ff}}{D_{rr}} = 1, \tag{3.1}$$

$$N_T = N_B, \tag{3.2}$$

with N_T and N_B, the number of tokens and bubbles in the ring respectively. This requirement makes it very easy to construct STRs and be sure that the rings evolve in an evenly spread mode. However, the interconnection delay does contribute to a large part of the overall propagation delay of tokens through the ring. When this interconnection delay is symmetric, which means that the delay of the forward and reverse lines are approximately equal, then the fraction in Eq. 3.1 does not change and the rule of Eq. 3.2 remains valid. This reasoning changes completely when the interconnection delay is not symmetric. The simulation tests how large the asymmetry needs to be, to bring the ring out of evenly spread mode. The results are dependent on the Charlie effect. From the previous sections, this effect tries to keep the ring in an evenly spread mode, so in theory, a higher Charlie effect should keep the ring out of burst mode longer than in the case of no Charlie effect.

As a measure of how much the STR is in burst mode, the Coefficient of Variation (CV) for the oscillation period is calculated. It is called the burst coefficient (BC):

$$BC = \frac{\sigma_{T_{per}}}{\mu_{T_{per}}},$$

with T_{per}, the length of the oscillation period and $\sigma_{T_{per}}$ and $\mu_{T_{per}}$, its standard deviation and mean respectively. A low BC means less in burst mode and when the BC equals zero, the ring is perfectly in evenly spread mode. The asymmetry is applied to only one ring: STR 2 and it is done by changing the relative routing delay difference between the forward and reverse interconnections. In this way, the Relative Routing Asymmetry (RRA) is defined as:

$$RRA = \frac{D_{forward} - D_{reverse}}{\mu_D},$$

with $D_{forward}$ and $D_{reverse}$, the mean forward and mean reverse interconnection delay respectively and μ_D, the average standard interconnection delay. For positive values of the RRA, the forward delay becomes larger and vice versa for negative values.

Fig. 3.3 shows the burst coefficient at different values of the RRA. The colour of the lines indicates the magnitude of the Charlie effect. It is clear that by increasing the routing asymmetry, the ring enters more and more in burst mode. It also shows that a larger Charlie effect does help keeping the ring out of burst mode. When the RRA becomes larger than two in absolute value, the ring is not in evenly spread mode any more. It happens when the the average forward (reverse) delay becomes more than three times as large as the average reverse (forward) delay.

To further investigate in what happens with the STR when the routing asymmetry is increased, the mean frequency of STR 1 and STR 2 are plotted in Fig. 3.4. The frequency of STR 1 is given as a reference, it is not affected by the routing asymmetry. Obviously, the frequency of STR 2 lowers when the forward delay is further increased. The frequency is independent of the Charlie effect. Note that a resonance effect can be observed when the frequency of STR 1 is an integer multiple of the mean frequency of STR 2. This effect is showed in Fig. 3.5. Here, the mean CS oscillation count is plotted for different values of the relative routing asymmetry. This number

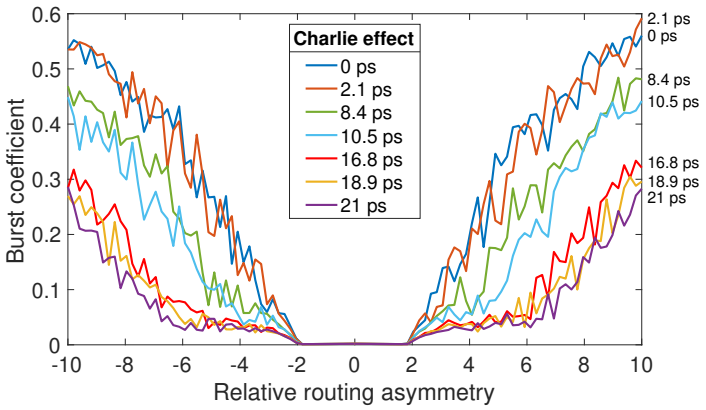


Figure 3.3: Burst coefficient at different RRAs.

is higher when the rings are more in phase. This effect is visible at the left of the graph, where the relative routing asymmetry becomes zero. In practice, there is the wanted operating region of the TRNG.

When the relative routing asymmetry rises, a first peak is encountered at a value of 3.469. Here, the mean frequency of STR 2 is approximately half of the frequency of STR 1 (1.24 MHz vs. 2.47 MHz). Because STR 2 provides the clocking signal for the CS, every time the oscillating signal produced by STR 2 has a rising edge, the oscillating signal produced by STR 1 has done exactly two periods and apart from the random gate jitter, STR 1 is always clocked at the same logical level.

A second peak is observed for a relative routing asymmetry equal to 4.694. Here, the frequency of STR 2 is approximately two and a half times smaller than that of STR 1 (100 MHz vs. 2.47 MHz). It is not an integer ratio, but due to the bursty behaviour of the oscillating signal, the simulations with a low Charlie effect (high burst mode) show a peak in the CS period count. The reason for this peak is the unequal length of consecutive periods for the signal produced by STR 2. Each of these lengths are an integer multiple of the period length of STR 1 (2 and 3 in this case), but their average is not an integer (2.5 in this case). It is further confirmed by observing that the simulations with higher Charlie effect do not show this peak, because they evolve to an evenly spread mode where each period of STR 2 is exactly two and a half times longer than the period of STR 1.

At last, the effect of the asymmetric routing on the output data correlation and bias is studied. The results are given in Figs. 3.6 and 3.7 respectively. The colour of the lines indicates the magnitude of the Charlie effect. For both, the Charlie effect and therefore also the bursty behaviour of the rings does not seem to influence the randomness. This result is because it is very hard to distinguish between deterministic (pseudo) randomness and true randomness. When the CS period counting value is low, the amount of accumulated jitter is also low and the generated bits are not truly random. Due to the complexity of the system, no direct trend is visible and it seems as if the data is random.

The coefficient of autocorrelation however does show a lot of maxima in function

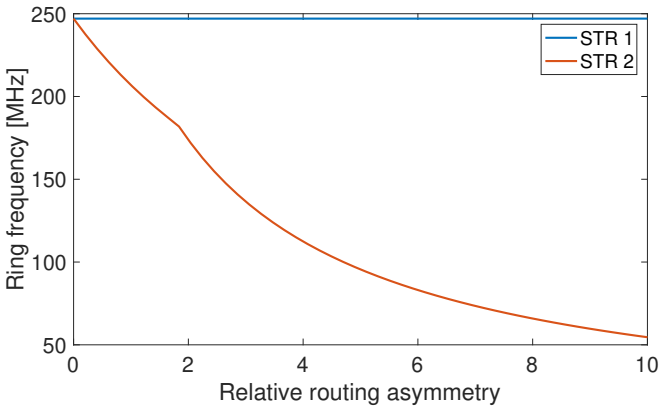


Figure 3.4: Oscillation frequency of the two STRs at different *RRA*s.

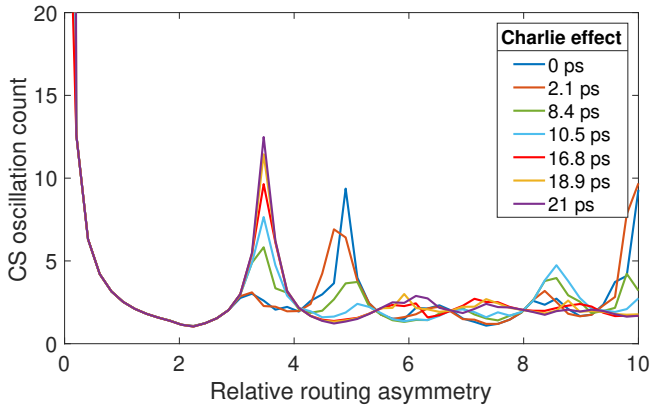


Figure 3.5: Mean CS period count at different *RRA*s.

of the relative routing asymmetry. A partial explanation for these maxima is that when the CS period count is an even number, the LSB does not toggle and a lot of consecutive sampled bits are equal to each other. This phenomenon occurs at regular intervals and is dependent at both the oscillating frequencies of the rings and the bursty behaviour of STR 2. When the relative routing asymmetry goes to zero, a large coefficient of autocorrelation and large bias is observed. The reason for these large values is again the same as for a too high sampling speed, the sampling speed should be lowered to wait until the rings are out of phase or more than only the LSB should be used as random data.

3.2.3 Effect of asymmetric stage delay

The same reasoning can be done, but then by biasing the stage delays instead of the interconnection delays. This simulation studies the effect when the static forward- and backward propagation delays are not equal any more. In a same way, the Relative

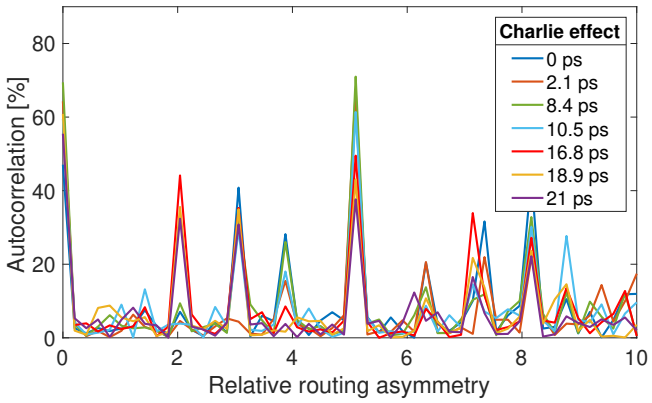


Figure 3.6: Autocorrelation coefficient with a shift of one bit at different RRAs.

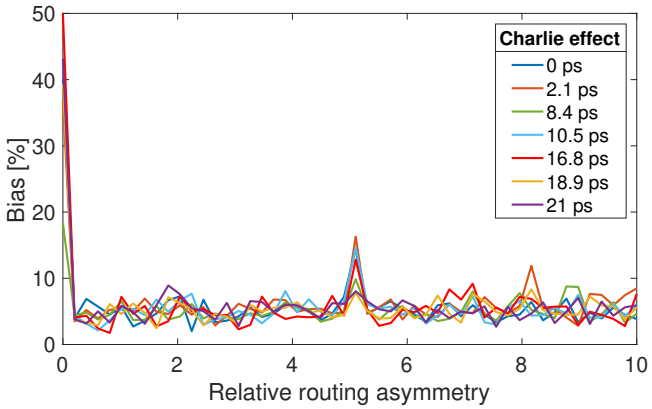


Figure 3.7: Bias of the output data at different RRAs.

Stage Asymmetry (RSA) is defined as:

$$RSA = \frac{D_{ff} - D_{rr}}{\mu_D},$$

with μ_D now the average static propagation of a stage. Again, the amount of bursty behaviour of the ring is given in Fig. 3.8. These results are similar as for the previous simulation, but one can notice that the magnitude of the burst coefficient is smaller for corresponding magnitudes of asymmetry. It should also be noted that the case of a large RSA is far less common than the case of a large RRA. Therefore, it can be concluded that the emphasis when designing a STRNG should be on reducing the RRA and less care has to be taken for reducing the RSA.

3.2.4 Effect of gate jitter

For the last simulation, the effect of the magnitude of the gate jitter is studied. This gate jitter is the only element that generates true randomness in the system.

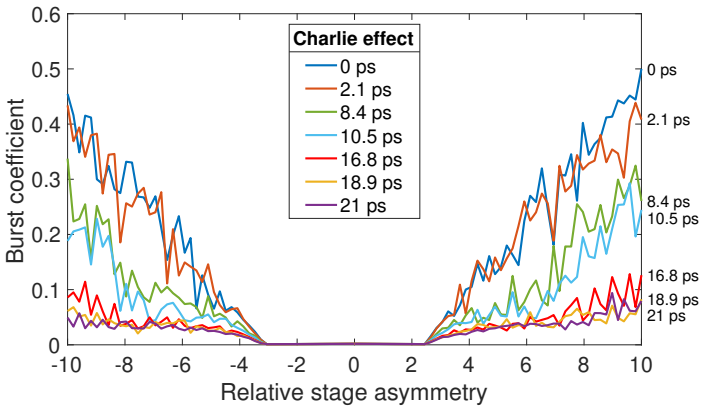


Figure 3.8: Burst coefficient at different RSAs.

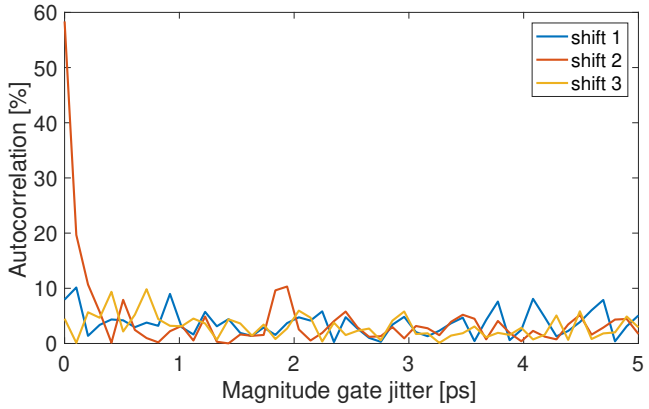


Figure 3.9: Autocorrelation coefficient with a shift of one, two and three bits at different values for the magnitude of the gate jitter.

When the jitter magnitude goes to zero, true randomness is lost and the randomness observed at the output is only pseudo randomness. In theory, the output is fully predictable (deterministic). In practice, this prediction is difficult. Figs. 3.9 and 3.10 do not show a clear trend when the jitter magnitude lowers. Only the autocorrelation coefficient with shift 2 in Fig. 3.9 does show some reduced randomness. The message from this simulation is that it is often very hard to detect reduced randomness by only looking at the output data stream.

3.3 Conclusion

Presented results from Sect. 3.2.2 show that careful routing is in fact very important to ensure that the rings evolve in an evenly spread mode. The consequence of this conclusion is that manual placement on FPGAs is always necessary. When the placement and routing is left to the tool, chances are high that the routing becomes

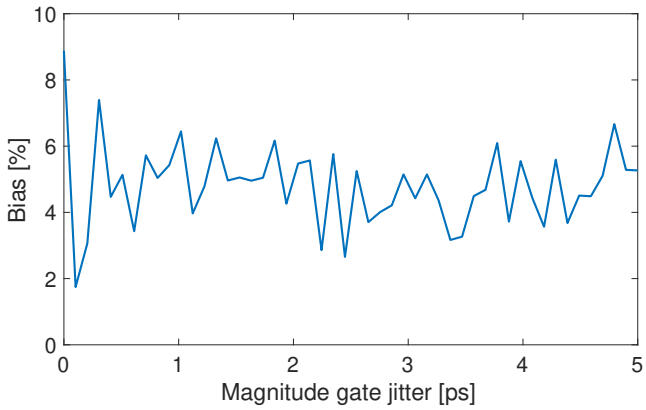


Figure 3.10: Bias of the output data at different values for the magnitude of the gate jitter.

very asymmetrical. Because of this asymmetry, the TRNG is vulnerable to attacks.

Another important conclusion from the results of Sect. 3.2.4 is that it is generally very difficult to almost impossible to detect minor changes in the TRNG behaviour by only looking at the generated output data. By just checking the bias and correlation, no distinction between true- and pseudo randomness can be made. Even more sophisticated test suits, such as the ones described in Chap. 4 can also not do it.

Chapter 4

Statistical testing

This chapter gives some background about statistical testing in general and a more detailed description about the test suites used for testing the TRNG in this thesis.

4.1 Background

The purpose of statistical testing is trying to find enough evidence to accept or reject a certain hypothesis. In the context of this thesis, the hypothesis is the true randomness of the generated data and its source. Note that even if a very large amount of tests are done, it can never be certain that the hypothesis is true or not. There is always a (small) chance of making an error. In statistics, this error occurs in two forms:

- **Error type I**: in this case, the hypothesis is falsely rejected. The generated data under the hypothesis is very unlikely. The chance of obtaining this error is denoted by the level of significance, α. It is the probability that data is generated that seems very not-random. For example, a range of n consecutive zeros has a chance of $(\frac{1}{2})^n$ of occurring. This chance goes down rapidly with increasing n, however it is never equal to zero. In theory, a TRNG can output every sequence of random data, even those that seem very unlikely to be generated by a true random source. Therefore, the significance level (α) is set to a very small value, typically $\alpha = 0.05$ or $\alpha = 0.01$. That last value is used in this thesis.

- **Error type II**: in this case, the hypothesis is falsely accepted. The generated data seems to be generated by a true random process. There is no evidence to suspect that the generator is not truly random. The chance for this error to occur is often denoted by β. This error is a harder problem than the previous one. In the previous case, the chance of having a type I error can be made as small as preferred, by decreasing the significance level α. It can be easily done by testing more data. Extraordinary results are then averaged out by the large amount of data. To some extend, the solution of testing more data can also be used to reduce the chance of obtaining this type of error.

For example, PRNGs can be tested until they repeat a previously generated sequence and hopefully the tests can detect this sequence. This type of error however remains a problem, because for example, well designed LFSRs can pass the tests described in the following sections despite the fact that they do not contain any source of randomness.

4.2 Test standards

The output data is tested according to two test standards: the National Institute of Standards and Technology (NIST) SP 800-22 and the Anwendungshinweise und Interpretationen zum Schema (AIS) 31. Note that each of the standards state that the TRNGs under evaluation should also comply to requirements not related to the output data stream. Because of the high risk of a type II error, it is not enough to pass the statistical tests described below. TRNGs should also be mathematically proven to generate random data. A stochastic model of the entropy source should be available. To accommodate these models, both standards distinguish different classes, according to the security of the TRNG.

4.2.1 NIST SP 800-22

This standard is described in [5]. It is created by the United States Department of Commerce that deals with measurements standards. The standard defines fifteen tests, listed in Table 4.1. Each of them calculates a P-value. This value indicates the probability of obtaining an outcome that is even less probable in case of a truly random data stream. When this probability is lower than a certain threshold, the test is indicated as failed. This bound on the P-value is the significance level (α), the chance of obtaining a type I error.

The software for this test standard is freely available on the internet from the NIST website. The recommended parameter settings described in [6] are used.

4.2.2 AIS 31

This standard is described in [6]. It is created by the German Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik (BSI)). They define nine tests, indicated from T0 to T8. For most of the tests, they also provide the probability of an error type I (α). These are listed in Table 4.3. What follows is a summary of these tests.

T0 (disjointness test): This test checks if an input of 2^{16} strings of bits with length 48 are mutually disjoint. It passes if none of them are exactly the same.

T1 (monobit test): This test has 20 000 bits as an input and checks if the bias is within a certain range. More specific:

$$b_1, b_2, \ldots, b_{20000} \in \{0, 1\},$$

Table 4.1: NIST SP 800-22 test suite

1	Frequency (Monobit) Test
2	Frequency Test within a Block
3	Runs Test
4	Test for the Longest Run of Ones in a Block
5	Binary Matrix Rank Test
6	Discrete Fourier Transform (Spectral) Test
7	Non-overlapping Template Matching Test
8	Overlapping Template Matching Test
9	Maurer's "Universal Statistical" Test
10	Linear Complexity Test
11	Serial Test
12	Approximate Entropy Test
13	Cumulative Sums (Cusum) Test
14	Random Excursions Test
15	Random Excursions Variant Test

$$T = \sum_{i=1}^{20000} b_i,$$

$$9654 < T < 10346.$$

T2 (poker test): This test checks if the groups of four consecutive bits are uniformly distributed. More specific:

$$b_1, b_2, \ldots, b_{20000} \in \{0, 1\},$$

$$c_j = 8b_{4j-3} + 4b_{4j-2} + 2b_{4j-1} + b_{4j} \quad for \quad j = 1 \ldots 5000,$$

$$f[i] = |\{j | c_j = i\}|,$$

with $|\cdot|$ denoting the size of a set. So the number of elements it contains. Then $f[i]$ is equal to the number of four bit numbers that are equal to i. Further:

$$T = \frac{16}{5000} \sum_{i=0}^{15} f[i]^2 - 5000,$$

$$1.03 < T < 57.4.$$

T3 (runs test): This test checks if the amount of sequences of consecutive bits with the same value is within acceptable limits. More specific:

$$b_1, b_2, \ldots, b_{20000} \in \{0, 1\},$$

$$\begin{aligned} T(\lambda, p) = |\{i \in 1, \ldots, 20000 | (b_{i-1}, b_i, \ldots, b_{i+\lambda}, b_{i+\lambda+1}), \\ b_{i-1} \oplus b_i = 1, b_i = \ldots = b_{i+\lambda} = p, b_{i+\lambda} \oplus b_{i+\lambda+1} = 1\}|, \end{aligned}$$

Table 4.2: Acceptable range for the number of sequences of given length

Sequence length	Lower bound, $m(\lambda)$	Upper bound, $M(\lambda)$
1	2267	2733
2	1079	1421
3	502	748
4	223	402
5	90	223
6	90	223

with:

$$p \in \{0,1\},\ b_0 \oplus b_1 = 1,\ b_{20000} \oplus b_{20001} = 1,\ \lambda = 1,2,\ldots,20000.$$

The value $T(\lambda, p)$ contains the number of consecutive sequences of length λ and bits with value p. For the test to pass, $T(\lambda, p)$ should be in the ranges given in Table 4.2:

$$m(\lambda) \leq T(\lambda, p) \leq M(\lambda), \quad for \quad \forall p \in \{0,1\},\ \forall \lambda \in 1,\ldots,5,$$

$$m(6) \leq \sum_{\lambda=6}^{20000} T(\lambda, p) \leq M(6) \quad for \quad \forall p \in \{0,1\}.$$

T4 (long run test): This test keeps track of the amount of sequences with length greater than 34 of consecutive bits with the same value. To pass this test, no such sequences should occur.

T5 (autocorrelation test): This test gives a measure of how well correlated bits are. It then checks if this correlation is within acceptable limits. More specific:

$$b_1, b_2, \ldots, b_{10000} \in \{0,1\},$$

$$T(\tau) = \sum_{i=1}^{5000} (b_i \oplus b_{i+\tau}) \quad for \quad \tau \in \{1,2,\ldots,5000\}.$$

To pass this test, $T(\tau)$ should be in the range:

$$2326 < T(\tau) < 2674 \quad for \quad \forall \tau \in \{1,2,\ldots,5000\}$$

T6 (uniform distribution test): This test checks if the bit stream is uniform within acceptable limits. It has three parameters:

- k: the length of each sequence, a value of 1 is proposed by the authors of this test set.
- n: amount of sequences to be tested, a value of 100 000 is proposed.
- a: a measure of the significance of the test, a value of 0.025 is proposed.

Define the n k-bit words:

$$(w_1, w_2, \ldots, w_n),\ w_i \in \{0,1\}^k,\ i = 1, 2, \ldots, n.$$

The test then calculates:

$$T(x) = \frac{|\{i|w_i = x\}|}{n} \quad for \quad x \in \{0,1\}^k$$

The test passes if $T(x)$ is within a certain range:

$$2^{-k} - a \leq T(x) \leq 2^{-k} + a \quad for \quad \forall x \in \{0,1\}^k.$$

T7 (test for homogeneity): This test checks if the generator has no memory by comparing the distribution of repeated experiments. It has four parameters:

- α: the significance level of the test.
- h: the number of repeated experiments.
- n: the length of each experiment.
- k: length of evaluated sequences of bits.

Define h repeated experiments:

$$(w_{11}, w_{12}, \ldots, w_{1n}),\ (w_{21}, w_{22}, \ldots, w_{2n}),\ \ldots,\ (w_{h1}, w_{h2}, \ldots, w_{hn}) \in \{0, 1, \ldots, 2^k - 1\}^n.$$

Then:

$$f_i[t] = |\{j|w_{ij} = t\}|,\ i = 1, 2, \ldots, h,\ t \in \{0, 1, \ldots, 2^k - 1\},$$

$$p_t = \frac{\sum_{i=1}^{h} f_i[t]}{hn},\ t \in \{0, 1, \ldots, 2^k - 1\},$$

$$T(h, n, k) = \sum_{i=1}^{h} \sum_{t=0}^{2^k - 1} \frac{(f_i[t] - np_t)^2}{np_t}.$$

The test passes then if:

$$T(h, n, k) \leq \chi^2(\alpha, (h-1)(2^k - 1)),$$

with $\chi^2(\alpha, (h-1)(2^k-1))$ the χ^2 distribution with $(h-1)(2^k-1)$ degrees of freedom, evaluated at the significance level α.

Table 4.3: Probability of a type I error for the AIS 31 tests

Test	α	Test	α
T0	$2^{-17} = 7.63 \cdot 10^{-6}$	T5	10^{-6}
T1	$\approx 10^{-6}$	T6	-
T2	$1.014 \cdot 10^{-6}$	T7	-
T3	10^{-6}	T8	-
T4	10^{-6}		

T8 (entropy estimation): Under certain conditions, this test gives a good approximation for the amount of entropy of the generated data. The test has three parameters:

- k: length of the evaluated sequences of bits, a value of 8 is proposed.
- Q and K: these parameters define the number of tested sequences, values of 2560 and 256000 are proposed respectively.

Consider the sequences:

$$(w_1, w_2, \ldots, w_{Q+K}) \in \{0,1\}^k.$$

A distance is defined as follows:

$$A_n = \begin{cases} n \quad \text{if no } i \in \{1, 2, \ldots, n-1\} \text{ exists with } w_n = w_{n-i} \\ min\{i | i \geq 1, w_n = w_{n-i}\} \quad \text{in all other cases.} \end{cases}$$

Then:

$$g(A_n) = \frac{1}{log(2)} \sum_{i=1}^{A_n - 1} \frac{1}{i},$$

$$T(Q,K) = \frac{1}{K} \sum_{n=Q+1}^{Q+K} g(A_n).$$

The test passes if:

$$T(Q,K) > 7.976.$$

Chapter 5

Design of the CS based STRNG

This chapter explains the methodology followed to design the proposed TRNG in Chap. 2. The first section explains the used set-up. In the second section, the architecture and all the building blocks it consists of are defined. The last section shows the results from the implementation and from the statistical tests, described in Chap. 4, on the generated random data.

5.1 Set-up

First, the used hardware and software are described.

5.1.1 Board

The first design is implemented in a *Xilinx Spartan 6 XC6SLX9-2CSG324C* FPGA on the *Spartan 6 FPGA LX9 MicroBoard* from *Avnet*. According to the data sheet [42], this FPGA contains 1,430 slices. Each slice holds 4 6-input LUTs and 8 Flip Flops (FF). The LUTs can be configured to represent any logical operator with 6 inputs and 1 output. The board also has 32 Block RAM (BRAM) blocks of size 18 kb each. Note that in a later stage of this thesis, another board is used, to perform certain attacks. This board is described in Chap. 7.

5.1.2 Tools and settings

The tool used for programming the FPGA is the *Xilinx ISE* design suite, version 14.7. The design is written in *Verilog*. A few settings need to be adjusted to avoid the tool optimizing away the circuit by combining LUTs or removing unused elements. These settings are listed in Table 5.1.

5.2 Architecture

The schematic of the complete architecture is depicted in Fig. 5.1. The two STRs at the left are the entropy generators. The CS unit block extracts the generated entropy.

Table 5.1: Design goals and strategies for the *Xilinx ISE* design suite, version 14.7.

Property name	Strategy value
FSM Encoding Algorithm	None
Resource Sharing	False
Register Duplication	False
Equivalent Register Removal	False
LUT Combining	No

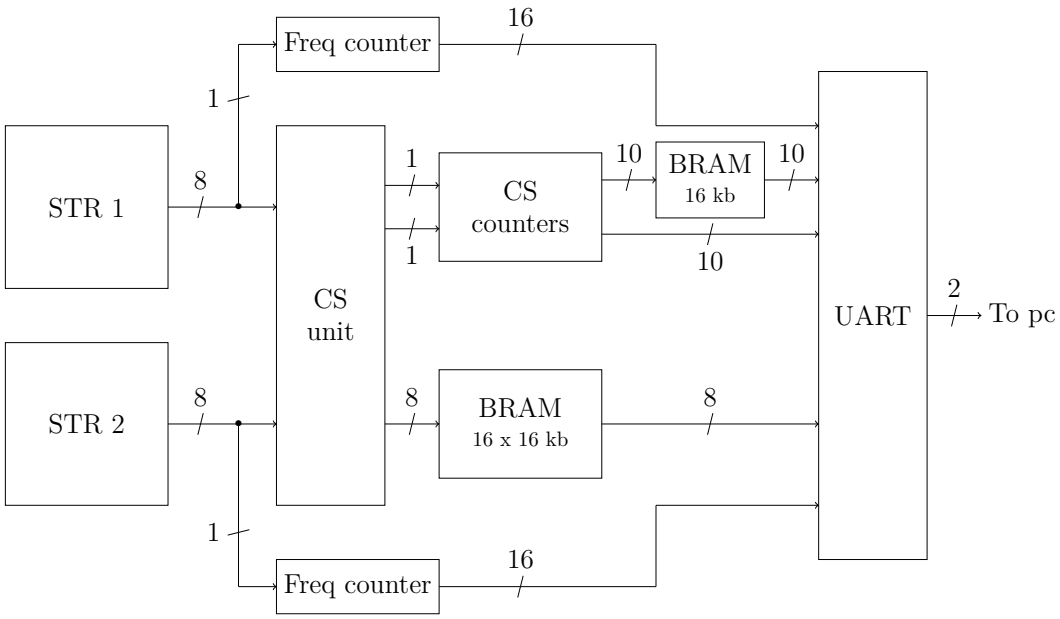


Figure 5.1: Complete architecture of the TRNG.

Two frequency counters are added to monitor the frequency of the two rings. A third counter (CS counter) counts how long the two rings are in phase. There are also two BRAM blocks. One is 16 kb large and stores one of the outputs of the CS counter. The other block contains 16 16 kb BRAM blocks. This block stores the generated random bits coming from the CS unit. The two BRAM blocks are implemented as First In First Out (FIFO) memory buffers. They are needed, because the bit rate of the connection to the computer is much lower than the generation of the random bits. The FIFO buffers permit to examine a large amount of consecutive bits. The last part of the architecture is the Universal Asynchronous Receiver-Transmitter (UART) interface. This block is responsible for the communication with the computer. The bit stream is analysed and additional post-processing is added (if necessary) on the computer.

5.2.1 STR

These two blocks represent the core of this TRNG. Each STR contains eight stages. Every stage consists of a Muller gate and an inverter. These two blocks can be easily implemented into one 6-input LUT, which is done as described in [17]. Fig. 5.2 shows how the LUT is connected. A feedback loop is necessary, because the stage needs to remember its previous state when both the F- and R-inputs are equal. Table 5.2 shows the truth table of a STR stage. The LUT implements this truth table. One slice of the used FPGA board contains four LUTs, so an eight-stage STR can be implemented into two slices. Every Configurable Logic Block (CLB) has exactly two slices in it, but these slices are most of the time not equal. According to [43], some slices contain extra carry logic (*SLICEL* and *SLICEM*). Because of this extra logic, when the STR is implemented into these different type of slices, the propagation delays are not equal and the two STRs are badly matched. Good matching is preferred, because it increases the effects of accumulated random jitter. To increase the matching between the two rings, they are implemented into the same type of slices (*SLICEX*), on different CLBs, that are situated next to each other. The wiring between the stages should also be made as symmetrical as possible. It is done by placing every stage manually. This placing is then written in a User Constrains File (UCF). It forces the *ISE*-tool to place the stages into the wanted positions. When the stages are placed symmetrical, the tool automatically wires them with good symmetry. The two STRs are thus placed into four different CLBs in the same row in the FPGA. Because of this placement, only one column is suited for positioning the rings. It is depicted in Fig. 5.3. This figure shows the zoomed in area of the two bottom clock domains. The whole FPGA is depicted in the top left corner. The TRNG is positioned in these columns that have four CLBs next to each other, marked with a rectangle. The five different positions tested are marked with an arrow. The other columns do not have four CLBs next to each other, they only have two. When it is tried to place the STRs into these columns, the two STRs are put into different columns. This placement results into bad matching and therefore also in low entropy in the outputs.

To increase the bit rate, the output of each stage is wired to one CS unit. According to [44] it can be done, because the accumulated jitter in each stage is independent.

The overall layout of the two rings is given in Fig. 5.4. As can be seen, the placement and wiring is very symmetrical, except for the wires coming from stage 0 (wires going to the bottom slices in the figure). The reason for this asymmetry is explained in Sect. 5.3. The two STRs together with the CS units are placed inside a Pblock. It is a way of telling the *ISE*-tool that these components belong together. An extra advantage is that in this way, the tool can be prohibited from placing components that do not belong to the STR inside the same slices as the STR stages. These other components would also ruin the matching of both rings.

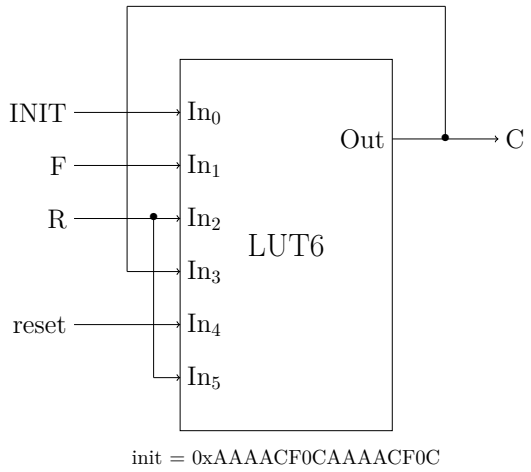


Figure 5.2: LUT representation of the STR stage.

Table 5.2: Truth table of a STR stage.

In0	In1	In2	In3	In4	In5	Out
INIT	F	R	C	reset	R	C
1	X	X	X	1	X	1
0	X	X	X	1	X	0
X	0	0	1	0	0	1
X	0	0	0	0	0	0
X	1	1	1	0	1	1
X	1	1	0	0	1	0
X	0	1	X	0	1	0
X	1	0	X	0	0	1

5.2.2 CS unit

This block needs to extract as much randomness as possible out of the two jittery signals coming from the STRs. An extra difficulty for this stage is that it needs to sample the two STRs without disturbing its functionality. The capacitive loading of the CS unit increases the propagation delays in the rings, which leads to an asymmetry if not designed carefully. The main clue is that the CS unit thus needs to be designed as symmetrical as possible. It is also done by placing the components manually and by writing these placements into the UCF of the design.

The basic structure of a CS unit is given in Fig. 5.5. The whole CS block contains eight of these units. One for each two signals coming from the two STRs. The four DFFs are implemented using the DFFs already available in the slices on the FPGA board. Notice that all the DFFs in one slice have their clock routed to the same line. It is thus impossible to place all the four DFFs into the same slice. The first two

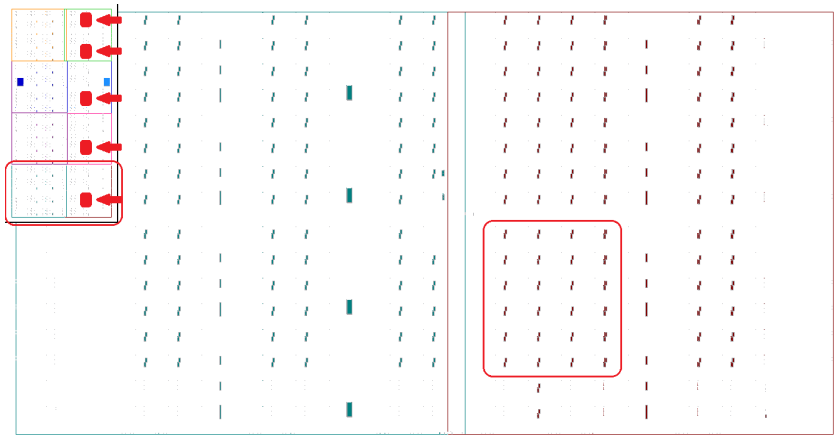


Figure 5.3: Placement of the TRNG.

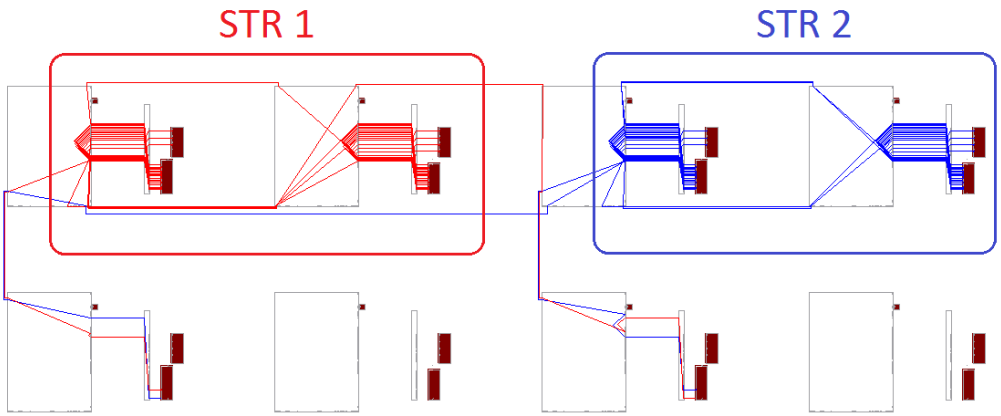


Figure 5.4: Placement and routing of the two STRs.

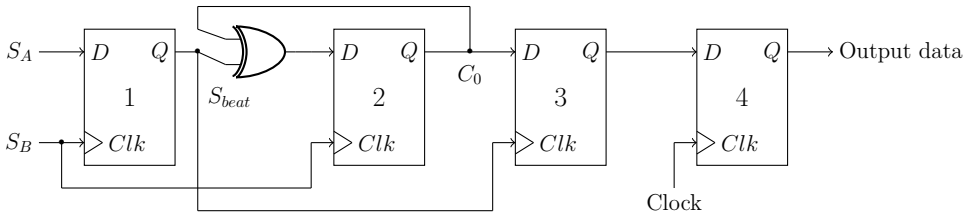


Figure 5.5: Basic structure of a CS unit.

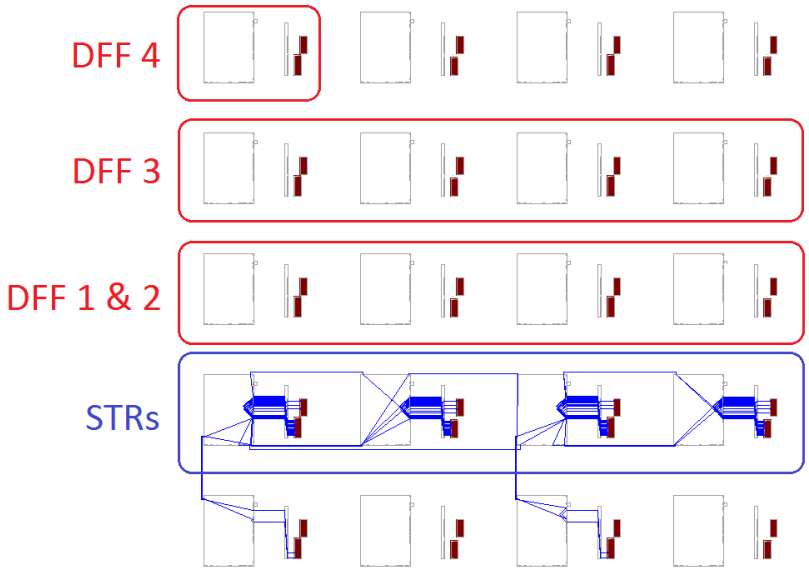


Figure 5.6: Placement of the CS units.

DFFs have the same clock, they can thus be placed in pairs of two into the same slice. Their inputs are coming straight from the STRs, so careful placement is necessary. They are placed as close as possible to the STRs, to minimize the capacitive load and thereby also the asymmetry. The fourth (last) DFF is clocked by the sample clock. This clock is the same for all the samplers, so they can be placed into the same slices. The third DFF is placed in between, to minimize routing delays. The overall placement of the samplers is given in Fig. 5.6. The DFFs are placed in layers, with the ones connecting to the STRs as close as possible to them. The figure only shows the units sampling the signals coming from stage 1 to 7. The one for stage 0 is identical, except that an extra counter is added to monitor the quality of the random output. This counter is further explained in the next section.

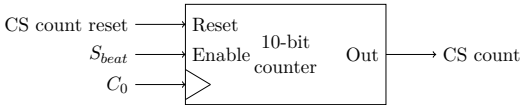


Figure 5.7: Block diagram of the CS counter.

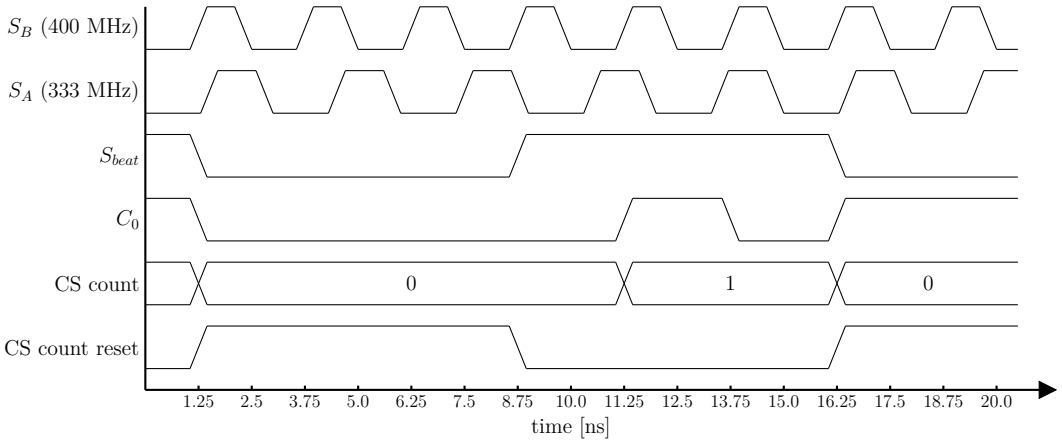


Figure 5.8: Time diagram of the different signals at the input of the CS counters.

5.2.3 CS counter unit

This counter counts the relative time that S_{beat} is high. Its block diagram is showed in Fig. 5.7. The counted value is a measure for the duration that the two signals coming from the 0^{th} stage of the two STRs are in phase. The higher this value, the better the matching is between the two rings. This principle is clarified in Fig. 5.8. The counter is a synchronous counter, clocked by C_0. It keeps counting as long as S_{beat} is high. When S_{beat} has a negative edge, the counter is reset. Actually two of these counters are implemented, one for each STR. When the matching between the stages is good, they both count to approximately the same value. The output of one of the counters is passed to a memory block, before it is send to the UART interface. It gives the possibility to inspect a large amount of consecutive counter values, which is a useful measure to determine the maximal frequency of the sampling clock. When the counter has not changed between two rising edges of this clock, the frequency has to be lowered.

5.2.4 Ring frequency counter

This block is an asynchronous counter, which counts the amount of cycles of the output of the 0^{th} stage of the two STRs. This counter is reset every 1024 cycles of the system clock, which runs at 100 MHz. Before every reset, the counted values are put in a register, which can be send to the computer. The computer can then calculate the frequency of the two STRs. This frequency also gives a measure for

the amount of matching and thus the quality of the output random bit stream. It is also a useful feature to check the effect of possible attacks on the TRNG. When the temperature or the supply voltage changes, it is reflected in a change of frequency of the STRs.

5.2.5 Block RAM

The design contains two blocks with memory capacity. The main function of these blocks is to buffer the data coming from either the CS counter unit or the CS unit itself. In this way, the data sent to the computer through the UART interface is in blocks of consecutive bits. The BRAM blocks are implemented as FIFO memories. In a first phase, they are filled with random bits or counter values. When the memory is full, it gives a sign to the UART interface that data is available. The data is then read out with a four-phase handshake protocol. A Finite State Machine (FSM) schema is given in Fig. 5.9. It is very important that a large amount of consecutive random bits is available on the computer. It permits to accurately assess the quality of the TRNG. Therefore, the memory which buffers the random bit stream consists out of 16 16 kb BRAM blocks, which is half of the available 32 BRAM blocks. When this memory was made larger, the wiring would cover the whole FPGA and this wiring would interfere with the STRs and degrade its performance. Because of this limitation, only the leftmost column of BRAM blocks is used, as can be seen in Fig. 5.10. The buffer to store the random bits uses half of the available 16 kb BRAMs. It still gives a lot of routing over the left side of the FPGA. The memory that buffers the output of the CS counter consists only out of one 16 kb BRAM block. There, a large amount of consecutive values is not needed.

5.2.6 UART interface

This module implements the communication with the computer. To validate the randomness of the output bit stream, a large amount of bits is needed. Typically, around 100 Mbit. Because of this large amount, the throughput of the UART interface should be as large as possible. The baud rate is set at 921 600 symbols per second. The actual bit rate achieved in the connection is 781.25 kbit/s. At this bit rate, it would then take 128 seconds to extract 100 Mbit out the FPGA. But due to memory latency on the computer it is a bit slower. The actual bit rate is around 160 kbit/s.

The UART interfaces takes care of both sending the random data and receiving commands from the computer. It can be switched between two modes. The difference is in the way they output the data to the computer. The first one is the testing mode, it outputs all the results from the counters and some generated random bits. Here throughput is not that important, because this mode is not used for extracting large amount of bits out of the FPGA. The interface sends packets of 14 bytes to the computer. The structure of these packets can be seen in Fig. 5.11. The second mode is the mode used for extracting large amount of bits. Here the interface just sends bytes containing the random output from every sampling unit. Throughput is

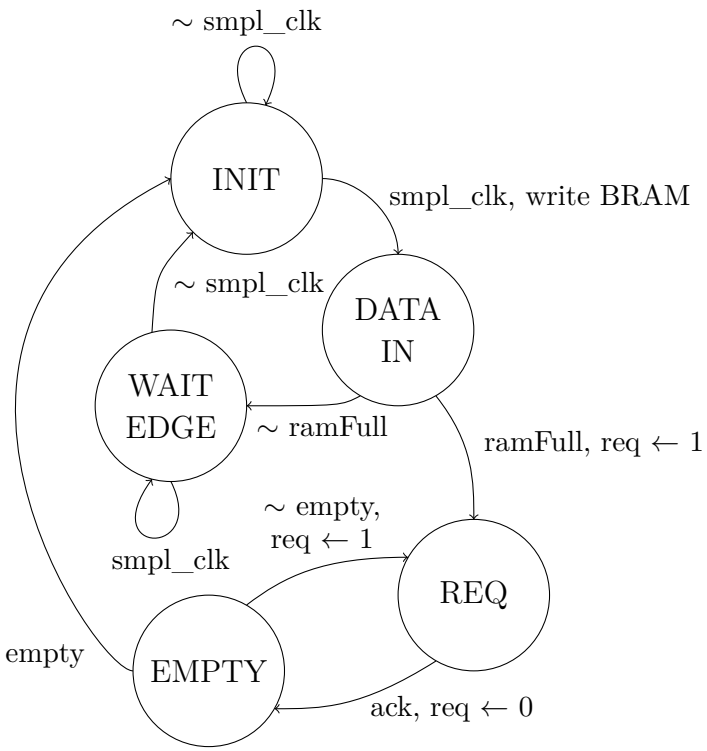


Figure 5.9: FSM schema of the BRAM blocks.

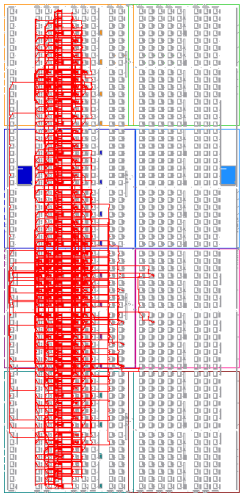


Figure 5.10: Occupation of the BRAM block in the FPGA.

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13
	0x01	0x02	4 random bytes				CS cnt 1		CS cnt 2		Freq cnt 1		Freq cnt 2	

Figure 5.11: Structure of packet the UART interface sends in testing mode.

very important here, so no extra information is send. The switching between these two modes can be done, by sending a command to the UART interface from the computer.

To receive instructions from the computer, the UART interface listens for commands. It distinguishes four instructions: reset, change sampling speed, change STR initialization and change the sending mode of the UART interface. In this way, the FPGA can be reconfigured in inaccessible locations such as e.g. in a temperature chamber, which is needed for the attacks that are performed in Chap. 7.

The actual implementation of the UART interface was not self made, but was downloaded from [45].

5.3 Results

5.3.1 General results

To compare the effects of different placement of the STRs, the design is implemented on five different locations. For reasons described earlier, they are all on the same column. The reference position is the coordinate of the slice in the bottom left of the Pblock containing the STRs and CS unit. A histogram of the output of the CS counters, together with a graph showing the transient behaviour of those counter values for the five different positions are given in Figs. 5.12 and 5.13 respectively. All the graphs have been created using 71 428 counter values. The histograms show a Gaussian distribution, which proves the working of the CS unit to extract the randomness out of the two jittery signals. It is clear that the position of the STRs is an important factor in determining the working of the TRNG. A lower CS count however does not automatically mean that the quality of the output data is lower. This quality is determined by applying statistical tests and observing which filter order is needed in the post processing (done on the computer) to pass the tests or by looking at the test success rate.

Fig. 5.14 shows the distribution of the extracted random bytes. These graphs are generated with 2 285 712 random bits. Every bit in a random byte corresponds with an output of a CS unit. Ideally, the distribution of the random bytes would be flat (uniformly distributed). When some outputs of the CS unit have a large bias, this bias would be reflected in a periodic behaviour in the distribution of the random bytes. The period of this effect is determined by the position of the biased bit stream in the random byte.

Table 5.3 gives an overview of the properties of the STRs in function of the different positions. All the rings oscillate with frequency of around 400 MHz. It is

Table 5.3: Properties of the STRs in function of the different positions.

Position	X12Y2	X12Y16	X12Y32	X12Y48	X12Y57
Mean CS counter value	20.7300	34.7580	45.0561	97.5966	19.8177
Standard deviation CS counter value	0.4989	3.3348	1.2114	6.0458	0.4697
Coefficient of variation [%]	2.4067	9.5943	2.6886	6.1947	2.370
Frequency STR 1 [MHz]	319.55	324.79	328.35	325.02	338.02
Frequency STR 2 [MHz]	323.99	321.77	326.09	324.00	343.39
Minimal post processing filter order	2	3	1	1	3
Throughput [Mbit/s]	3.125	2.0833	6.25	6.25	2.0833

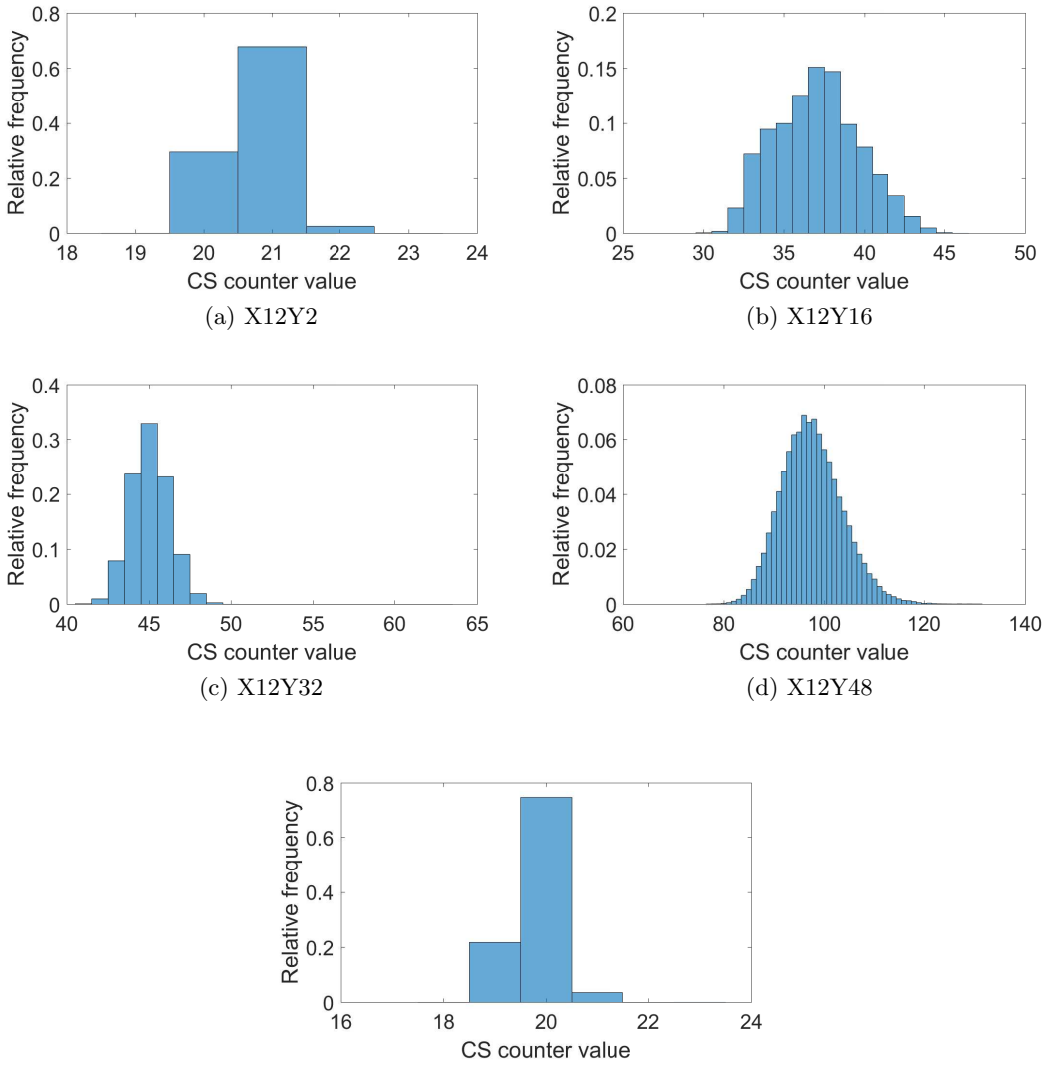
observed that a larger difference in frequency between the two rings gives a lower CS counter value, which is in line with the theory. It is also clear that the coefficients of variation of the CS counters are of the same magnitude for the different placements. It means that the standard deviation scales approximately linear with the mean CS count.

As noted earlier, there is an asymmetry in the wiring of the 0^{th} STR stage. It is deliberately done to lower the matching between the two rings. Otherwise, the rings would be too long in phase and it would take a long time to create a random bit. The sampling frequency together with the output bit rate should then be reduced. The asymmetry is added to keep the CS count under 100.

As a last step, the sampling frequency is chosen. This frequency should be as large as possible to have good output bit rate. But when chosen too high, there is correlation between consecutive bits, which is unwanted. The sampling frequency is chosen the same for all five positions and is equal to 781.125 kHz. It gives an overall bit rate of 6.25 Mbit/s at the output of the TRNG. This bit rate is lowered if additional post processing is needed to pass the statistical tests.

5.3.2 Statistical test results

The test results for the AIS 31 and NIST SP 800-22 standards are shown in Tables 5.4 and 5.5 respectively. For the AIS 31 test suite, only the first four tests are evaluated, which is due to the large computational demands of the other tests. The tests are both executed using 100 Mbit of generated random data. As already noted earlier, positions X12Y32 and X12Y48 pass the tests without any post processing. Position X12Y57 only just fails the NIST SP 800-22 tests. Note also that the four tests from the AIS 31 standard are far less sensitive than these of the NIST SP 800-22 standard.



(a) X12Y2

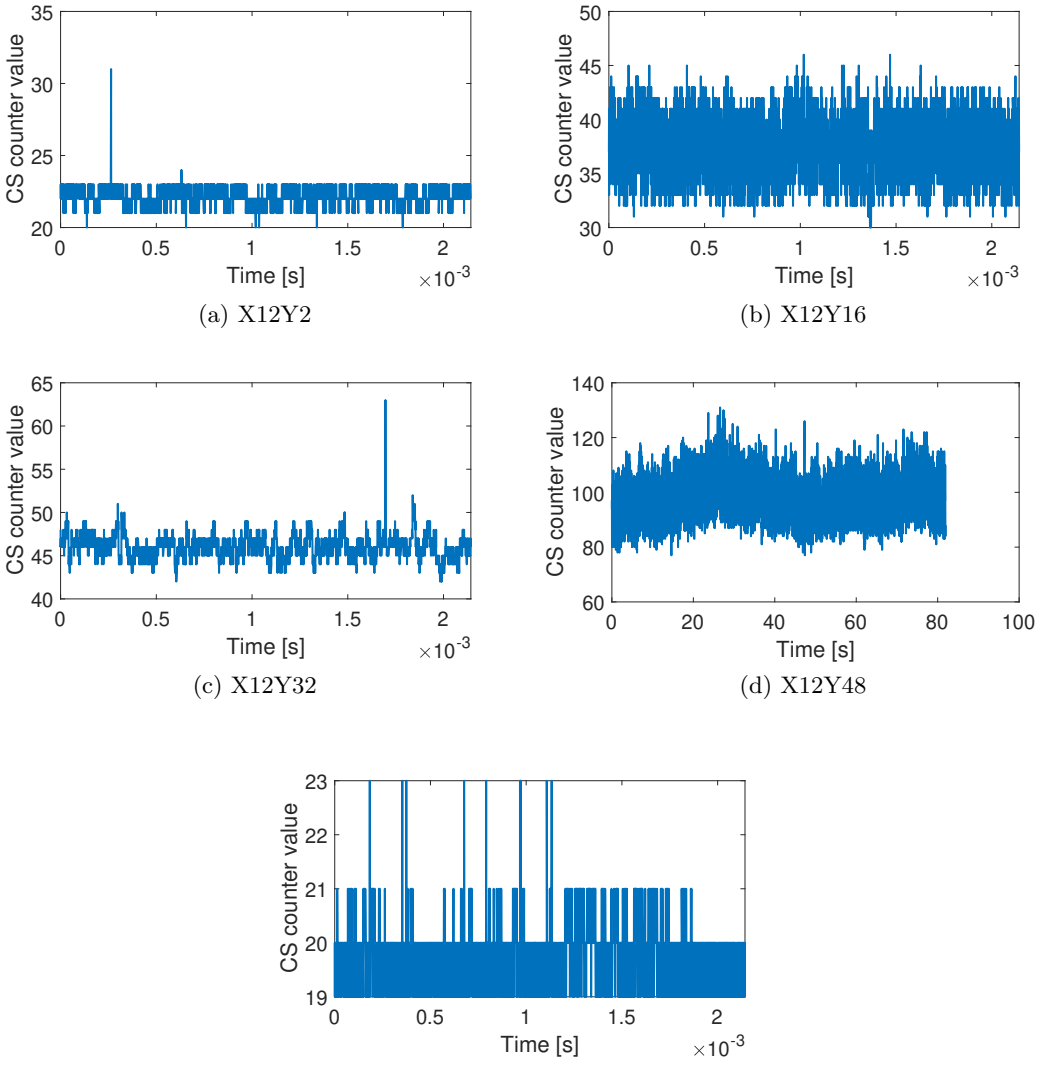
(b) X12Y16

(c) X12Y32

(d) X12Y48

(e) X12Y57

Figure 5.12: CS counter value histograms for different positions.



(a) X12Y2

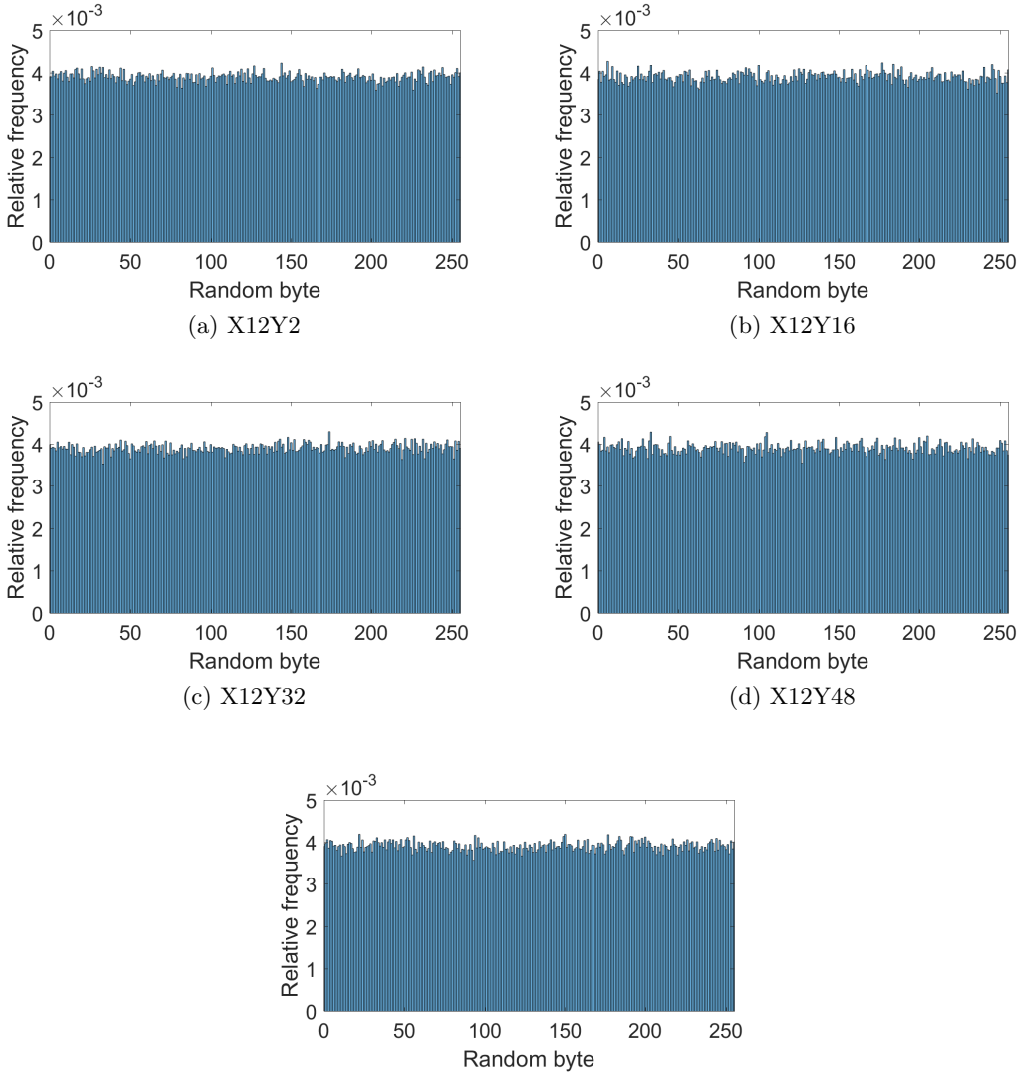
(b) X12Y16

(c) X12Y32

(d) X12Y48

(e) X12Y57

Figure 5.13: CS counter value transient behaviour for different positions.



(a) X12Y2

(b) X12Y16

(c) X12Y32

(d) X12Y48

(e) X12Y57

Figure 5.14: Distribution histogram of the created random bytes.

Table 5.4: AIS 31 test results.

Position	X12Y2		X12Y16		X12Y32	
Test	Proportion	Rate	Proportion	Rate	Proportion	Rate
Monobit	5001/5002	99.98%	5001/5002	99.98%	5001/5002	99.98%
Poker	5002/5002	100%	5002/5002	100%	5002/5002	100%
Runs	5001/5002	99.98%	5000/5002	99.96%	5001/5002	99.98%
Long run	5002/5002	100%	5002/5002	100%	5002/5002	100%
Position	X12Y48		X12Y57			
Test	Proportion	Rate	Proportion	Rate		
Monobit	5001/5002	99.98%	5001/5002	99.98%		
Poker	5002/5002	100%	5002/5002	100%		
Runs	5001/5002	99.98%	5001/5002	99.98%		
Long run	5002/5002	100%	5002/5002	100%		

Table 5.5: NIST SP 800-22 test results.

Position	X12Y2		X12Y16		X12Y32	
Test	Proportion	Success	Proportion	Success	Proportion	Success
Frequency	97/100	yes	99/100	yes	97/100	yes
Block frequency	98/100	yes	94/100	no	98/100	yes
Cumulative sums	99/100	yes	98/100	yes	97/100	yes
Runs	100/100	yes	99/100	yes	98/100	yes
Longest run	97/100	yes	86/100	no	100/100	yes
Rank	96/100	yes	100/100	yes	98/100	yes
FFT	97/100	yes	98/100	yes	96/100	yes
Non overlapping template	95/100	no	27/100	no	96/100	yes
Overlapping template	97/100	yes	37/100	no	99/100	yes
Universal	98/100	yes	95/100	no	98/100	yes
Approximate entropy	0/100	no	0/100	no	100/100	yes
Random excursions	60/63	yes	57/59	yes	50/51	yes
Random excursions variant	62/63	yes	58/59	yes	50/51	yes
Serial	93/100	no	11/100	no	98/100	yes
Linear complexity	99/100	yes	99/100	yes	99/100	yes
Position	X12Y48		X12Y57			
Test	Proportion	Success	Proportion	Success		
Frequency	98/100	yes	99/100	yes		
Block frequency	98/100	yes	99/100	yes		
Cumulative sums	98/100	yes	98/100	yes		
Runs	99/100	yes	100/100	yes		
Longest run	99/100	yes	100/100	yes		
Rank	100/100	yes	99/100	yes		
FFT	99/100	yes	97/100	yes		
Non overlapping template	97/100	yes	96/100	yes		
Overlapping template	99/100	yes	95/100	no		
Universal	100/100	yes	99/100	yes		
Approximate entropy	99/100	yes	89/100	no		
Random excursions	61/63	yes	58/61	yes		
Random excursions variant	62/63	yes	60/61	yes		
Serial	98/100	yes	98/100	yes		
Linear complexity	97/100	yes	98/100	yes		

Chapter 6

Implementation defects

This chapter studies the effect of possible implementations defects of the TRNG proposed in Chap. 2. The first section explains in more detail what implementation defects and its possible consequences are. The following sections: 6.2, 6.3 and 6.4 take a closer look at the effects of routing asymmetry, surrounding activity and placement respectively. A conclusion is given in the last section.

6.1 Introduction to implementation defects

Implementation defects are mistakes made during the implementation of the TRNG. A TRNG is not a completely digital component. It is made of building blocks whose goal is to amplify an analog noise source. Because of this amplification, they are by construction very sensitive to parasitics that would otherwise not affect a regular digital circuit. When a TRNG is carelessly implemented, the behaviour can be altered a lot, most of the time in a bad way. In essence, a TRNG implementation must be concerned about two things:

- **Symmetry**: TRNGs rely significantly on a symmetric placement and routing. For example, in the open loop TRNG, the delay lines of the data- and clock lines must be highly symmetrical to ensure that the sampling DFF is brought into metastability. Also in the TRNG implemented in this thesis, it is very important that both the STRs are symmetrically implemented. When the amount of symmetry is low, deterministic randomness can overtake the true random component and make the TRNG predictable to some extend. The problem of symmetry can be solved by the TRNG designer by defining hard macros. In this way, all placement and routing stay fixed relatively. The user of the TRNG can then place the module inside his design.

- **Neighbouring activity**: because TRNGs are designed to amplify noise, they are very sensitive to it. The source of the noise should be controlled as much as possible. Only when the noise source is exactly known, a stochastic model for the TRNG can be made and its security can be analysed. A TRNG however is usually not alone on an IC or FPGA. Other digital blocks, especially those

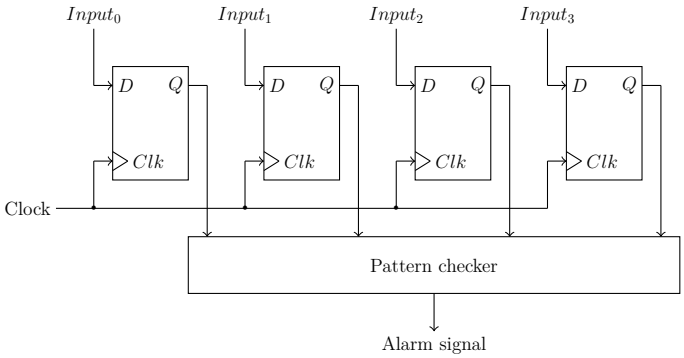


Figure 6.1: Architecture of the burst detector.

with a high activity, do create a lot of noise. This noise naturally couples to the TRNG, either via the supply lines or via the substrate. In the presence of large amounts of unwanted noise, the behaviour of the TRNG is then altered and security analyses do not hold any longer. It can for example be the case for a RO TRNG. During the analysis, it is assumed that the ROs do not lock. However when periodic switching of another digital device causes the ROs to lock, randomness is lost. Similarly for the TRNG, implemented in this thesis, it is assumed that both the STRs are in evenly spread mode. This condition does not always hold, as experiments show in the following sections. This problem is harder to overcome for TRNG designers, because they have no control over the environment where the TRNG is implemented.

The following sections show the results of various experiments that were conducted to measure the effects of possible implementation defects and try to discover features that can be used to detect a bad functioning TRNG.

6.2 Routing asymmetry in a STR

6.2.1 Methodology

The simulations from Chap. 3 show that routing asymmetry can cause a STR to go into burst mode. This mode violates the stochastic model presented in Chap. 2 and therefore affects the theoretical soundness of the TRNG. To study this effect, a way of measuring the magnitude of the burst behaviour is created. It is done with the help of a burst detector. The architecture of the burst detector is showed in Fig. 6.1. Each burst detector has as input the output of four consecutive stages of the STR. It then checks if the four outputs contain a pattern that would indicate that the STR is not in evenly spread mode. These patterns are given in Table 6.1. When one of those patterns occur, an alarm signal is raised. To properly test the effects of asymmetric routing, a STR with 32 stages is used. Each burst detector checks four stages, so a total of eight burst detectors work in parallel. At each clock period, one byte is created and sent to the computer for further analysis.

Table 6.1: Illegal patterns when the STR is in evenly spread mode.

Tokens - Bubbles	Stage outputs
- B B B	0 0 0 0
- B B B	1 1 1 1
- T T T	1 0 1 0
- T T T	0 1 0 1

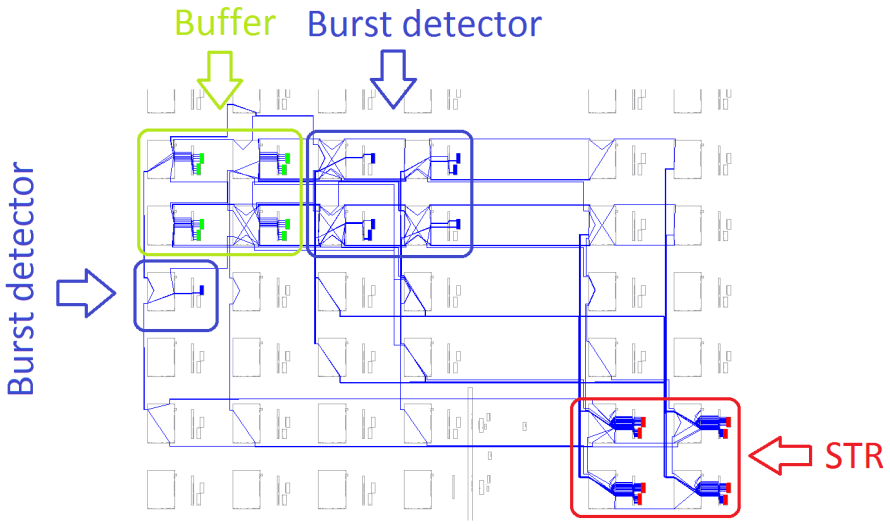


Figure 6.2: Placement of buffer (X10Y12) and STR (X18Y8) in the FPGA.

To increase the routing asymmetry, an additional buffer is added that buffers all the forward going signals. For a signal to go from the output of a stage to the forward input of the next stage, it first has to pass through this buffer. By altering the position of this buffer, the forward delay can be increased. The reverse delay is unaltered, so an increase in forward delay increases the routing asymmetry with a same amount. Fig. 6.2 shows an example placement for the buffer and the STR.

For each test, the STR is initialized already in burst mode, so all tokens are grouped together during initialization. In this way, it becomes clear if the STR is able to evolve into an evenly spread state or not.

6.2.2 Results

Fig. 6.3 shows the burst index (BI), defined as:

$$BI = \frac{\#Alarms}{\#Signals},$$

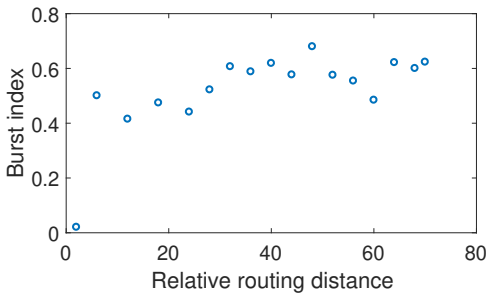


Figure 6.3: Burst index for different values of the relative routing distance of the forward interconnection lines.

with $\#Alarms$ and $\#Signals$, the number of alarms and the total number of signals received from the burst detectors respectively. The relative routing distance is defined as the number of slices the forward signal has to cross horizontally and vertically. There is an increasing trend and apparently, the STR is very sensitive to small increases in asymmetry. The results from this experiment are of huge importance. Precise placement and routing of the STRs is needed to obtain a good functioning STR with no burst mode. It is thus not possible to let the placement and routing over to a standard automatic tool for FPGA programming.

6.3 Surrounding activity

6.3.1 Methodology

In this experiment, the effect of surrounding activity is studied. ROs are used as surrounding circuitry, because they are good noise creators and relatively easy to implement. When the RO consists out of very few stages, the activity is high. For this experiment, a RO with two stages is made. Each stage is implemented in one LUT, one stage is non inverting and the other is inverting to get the oscillatory behaviour. Each slice contains four stages of four different ROs. Each CLB contains four complete ROs. These ROs are then placed around the TRNG. The amount of ROs is gradually increased to study its effects. Fig. 6.4 gives the placement of the TRNG together with 1000 ROs.

6.3.2 Results

The effect on the frequency of the STRs is presented in Fig. 6.5. The frequency lowers with increasing amount of neighbouring ROs. A reason for this effect is that the supply network is unable to handle the increasing amount of current consumption by the ROs. Therefore, the supply can drop slightly witch causes a lowering in oscillation frequency of the STRs. The mean CS count is showed in Fig. 6.6. At the beginning, the CS count does not alter significantly. However, after adding more than 2 000 ROs, the mean CS count drops rapidly. This drop may have many causes.

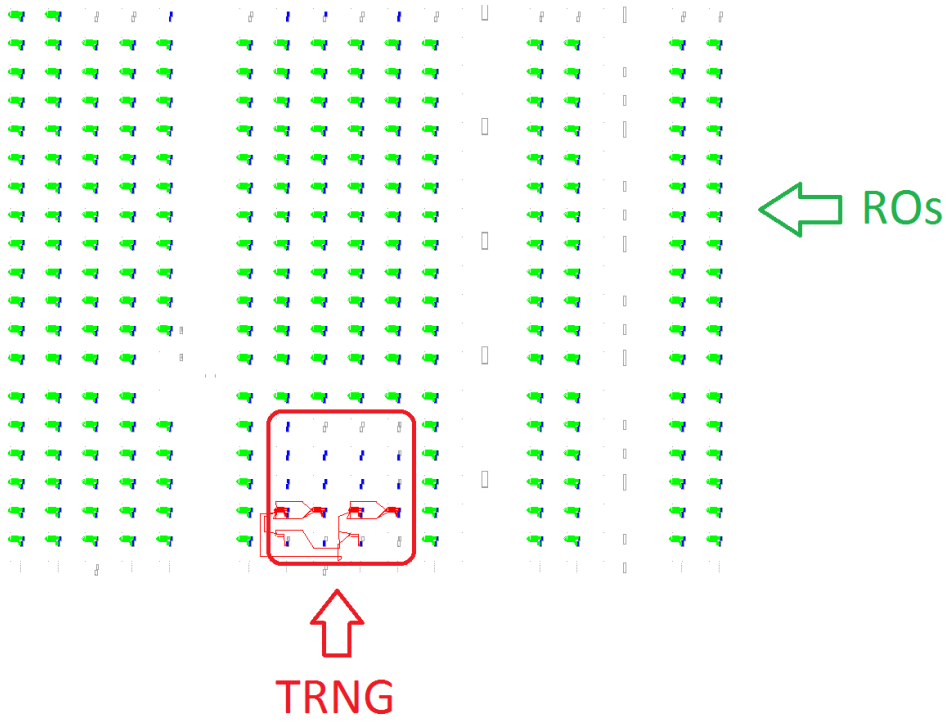
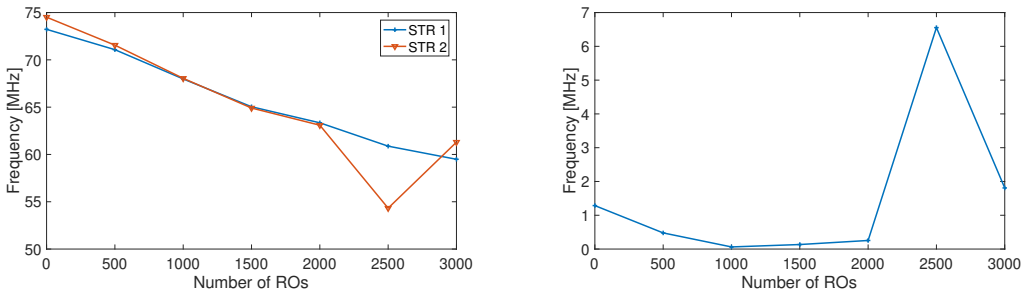


Figure 6.4: Placement of the TRNG (X20Y2) together with 1000 ROs.



(a) STR frequencies

(b) STR difference frequency

Figure 6.5: Frequency of both the STRs and the difference at different amounts of surrounding ROs.

The STRs can have an increased amount of bursty behaviour, which can lower the mean CS count. Fig. 6.7 shows the bias and correlation for different amounts of neighbouring ROs. No significant effect is visible, however one could argue an increasing amount of randomness with an increasing number of ROs, according to the bias graph.

Then, the NIST tests where performed on 100 sequences of length 1 Mbit. These results are presented in Fig. 6.8. A filled square marks a failed test (less than 96 %

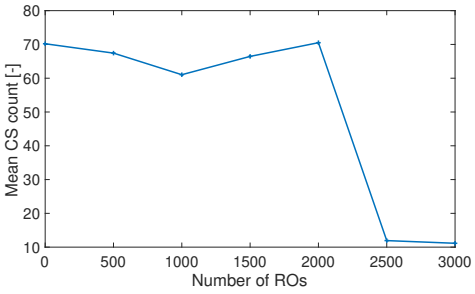
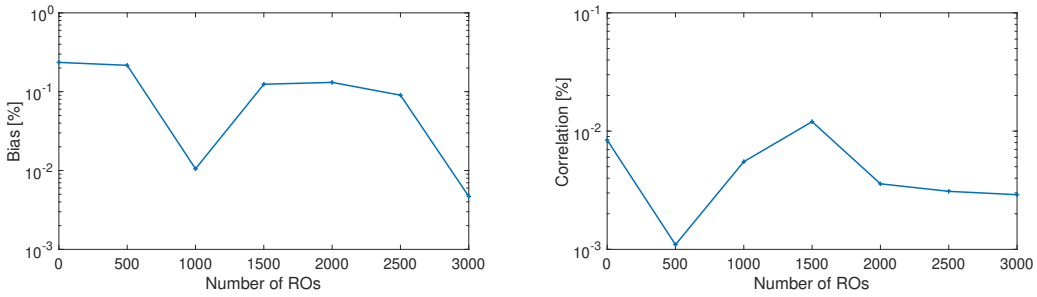


Figure 6.6: Mean CS count values at different amounts of surrounding ROs.



(a) Bias (b) Correlation

Figure 6.7: Bias and correlation of the output data at different amounts of surrounding ROs.

success rate). There is no significant trend visible. A slight increase of randomness can be seen for the number of ROs ranging from 1 000 to 2 000. This increase is further confirmed by Fig. 6.9, which shows the total success rate for each number of ROs. This experiment shows that the behaviour of the TRNG is dependent on the sensitivity difference between the two STRs. Both STRs react differently to the noise generated by the surrounding ROs. This asymmetry can cause the TRNG to fail. In this experiment however, the reactions of both STRs were quit similar (see Fig. 6.5) and the reduction of randomness in the output stayed within acceptable limits.

6.4 Placement

6.4.1 Methodology

As a last experiment the effect of placement is studied. The vertical position of the TRNG is changed from the bottom of the FPGA to the top. The position of the TRNG is then always described by X20Y-, with Y ranging from 2 to 57. Only the effect on the CS count is measured.

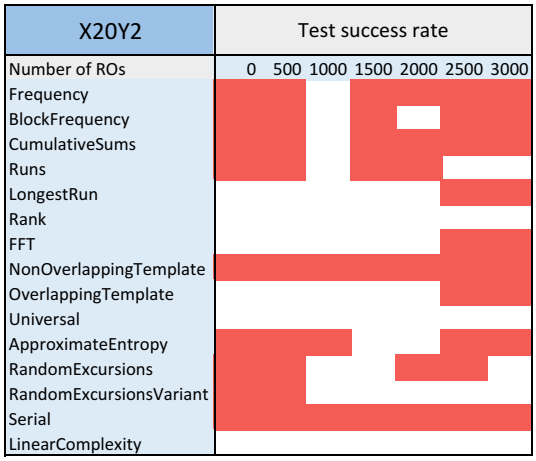


Figure 6.8: Failed tests for each position at different amounts of surrounding ROs.

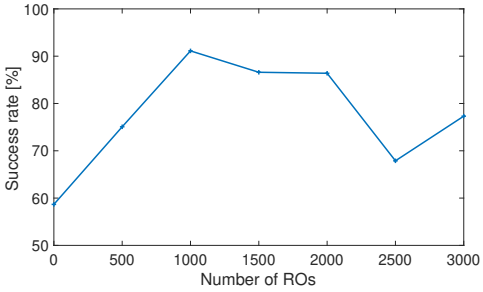


Figure 6.9: Total success rate at different amounts of surrounding ROs.

6.4.2 Results

Fig. 6.10 shows the mean CS count at different vertical positions of the TRNG. At the edges, there are more spikes with higher CS count. It means that at the edges, the two rings are more frequently in phase than in the middle of the FPGA. This effect can be explained by observing that most of the other logic is implemented in the centre of the FPGA. When the TRNG is then placed at the centre, there is more neighbouring activity and the behaviour of the STRs gets more bursty. This bursty behaviour then leads to a lower mean CS count. Another observation is that the behaviour of the TRNG is indeed very placement dependent. A reason for this behaviour is that not all slices contain the same circuitry. As previously explained, a *Spartan 6* FPGA contains three types of slices: *SLICEM*, *SLICEL* and *SLICEX*, each with different components and therefore also different propagation delays. Also process variability ensures that identically designed LUTs do not have identical propagation delays, it too can give variability with placement.

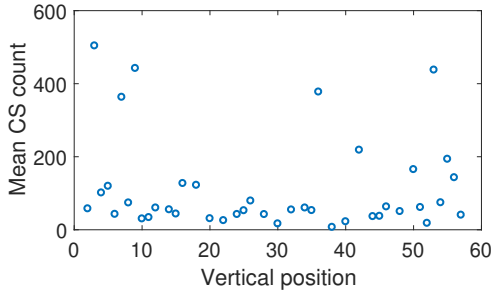


Figure 6.10: Mean CS count at different vertical positions in the FPGA.

6.5 Conclusion

The effect of implementation defects may not be underestimated. All three conducted experiments show that the behaviour of the TRNG does change with surroundings that would otherwise not influence standard digital circuitry. Especially the results from Sect. 6.2 show that the assumption of an evenly spread mode in the security analysis of [10] is not just trivially fulfilled. A lot of care to the design has to be given, especially with FPGAs. It makes the design and implementation of a TRNG look very similar to the design of analog circuits. Factors such as symmetry, noise sensitivity and placement are decisive for the quality of the generated random data. A next element that complicates the design is the fact that there is not a lot of freedom when implementing TRNGs on FPGAs. Standard tools are developed for deterministic behaviour and this determinism is often contradicting the requirements of a good TRNG.

Chapter 7

Physical attacks

In this chapter, the methodology and results of two physical attacks performed on the TRNG, designed in Chap. 5, are presented. The first section gives a brief explanation why performing physical attacks on TRNGs is of great importance. Sections 7.2 and 7.3 describe the temperature- and voltage attack respectively in more detail. A separate conclusion is given at the end of each of these sections.

7.1 Introduction to physical attacks

This section describes the methodology and results of performing physical attacks on the previously designed TRNG. The goal of this analysis is to investigate how changes in the operating conditions of the FPGA affect the TRNG. By doing this analysis, features are extracted from the internal working of the TRNG that can be used to detect reduced security as fast and efficient as possible. Reduced security of cryptographical systems is a very undesirable outcome and can be caused by a failure of the TRNG. Changes in operating conditions happen with certainty in real-life use. Either they happen accidentally or deliberately by physical adversaries with access to sensitive applications, for instance, smart cards, smart cars, devices implanted in the human body, etc. In the past, the emphasis was mostly on the mathematical security of cryptographic algorithms. These algorithms where then executed on servers, located in a very controlled and difficult to attack environment. Now, the environment is completely in control of the attacker and a mathematical very secure algorithm can fail completely due to a vulnerable implementation.

One could argue that means of detecting TRNG failures already exist. The statistical tests (NIST or AIS), described in Chap. 4 are very sensitive to reduced randomness in the output data. These tests however require huge amounts of data and processing power, which is not available in energy and area constrained devices as used in many embedded applications. Therefore, it is important to study each type of TRNG case by case, to discover weaknesses and implement on-line tests that are far more efficient than the standard statistical tests suites.

The different attack methodologies can be divided into two groups: passive and active attacks. Passive attacks do not alter the operating conditions of the target

device. Consequently, they cannot be detected by on-line tests. Passive attacks try to extract as much information as possible out of the TRNG, either directly via the output data or indirectly via leaked side-channel information. They attempt to model and predict future output data with a sufficiently large probability. Active attacks, on the other hand, do change the operating conditions. They attempt to stress the hardware to a point where the output data is not truly random any more. Different attack strategies can be distinguished: temperature, power supply voltage (static or by introducing glitches), over-clocking, clock glitches, EM waves, etc. Some attack strategies have already been tested at TRNG implementations. The first attack was done by [7], they injected a sine wave on the power supply of a RO TRNG and where able to generate a bias at the output data stream. Another example is given by [8], here also a RO TRNG was biased by locking the ROs with EM waves impinging on the FPGA. The benefit of this system was that it required no direct contact with the device under attack, which would be ideal to break smart cards. In [9], temperature, voltage (static and with glitches) and clock glitches attacks where performed to a classical STRNG. As already explained, this TRNG also uses STRs to generate jittery signals, but the main difference is in the way the digitization works. However, it can be expected that these results are fairly similar. This work investigates two types of active attacks: temperature and static voltage.

The NIST tests, together with some basic bias and correlation calculations monitor the quality of the output data. The number of failed tests is used as a robustness metric. For each operating condition, 100 NIST tests are executed on sequences of length 1 Mbit. The significance level is again set to 0.01.

7.2 Temperature attack

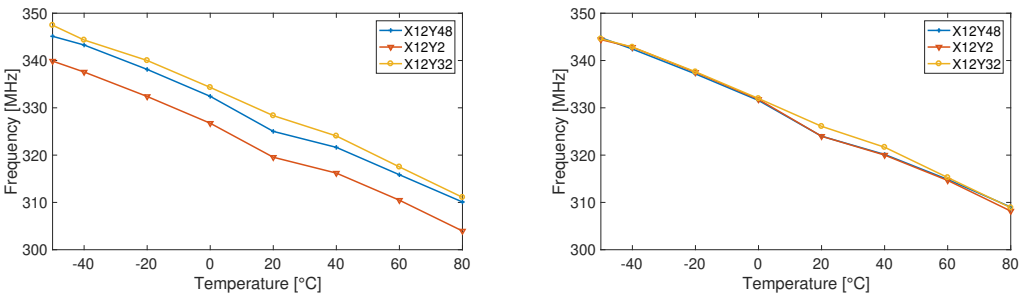
7.2.1 Attack methodology

The first attack is a temperature attack. Here, the FPGA (still the *Spartan 6 XC6SLX9-2CSG324C* on the *Spartan 6 FPGA LX9 MicroBoard* from *Avnet*) is placed inside the *ESPEC SH-662* temperature and humidity chamber. The temperature is varied from -50 °C to 80 °C in steps of 20 °C. At each temperature, before the generation of random data, an isolation test is executed. This test makes sure that the surrounding logic (BRAM, UART interface, frequency counters, etc.) still work correctly and the results can be interpreted as only influenced by the TRNG itself.

At each temperature, three different placements inside the FPGA are studied: X12Y2, X12Y32 and X12Y48. As already noted earlier, placement has a large influence on the performance. This topic was addressed in the previous chapter.

7.2.2 Results

The STR frequency of the different TRNG designs across different temperatures is given in Fig. 7.1. The frequency decreases with increasing temperature. It is because the reduced electron- and hole-mobility at higher temperatures. The CS unit does not depend on absolute frequencies, but on the difference frequency between the two



(a) STR 1 (b) STR 2

Figure 7.1: Frequency of both the STRs at different surrounding temperatures.

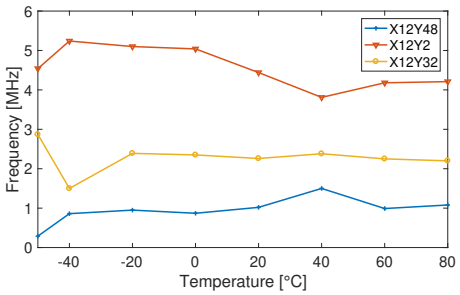


Figure 7.2: Frequency difference between both STRs at different surrounding temperatures.

STRs. This difference is plotted in Fig. 7.2. As the difference remains fairly constant, not a large effect on the output data is expected. The mean CS counting values are presented in Fig. 7.3. Here, a distinct difference in placement is visible. In particular, position X12Y48 does show a large dependency on temperature in contrast with the other two placements. The bias and correlation with shift one are given in Fig. 7.4. They do not show a specific trend. The values stay fairly close to the ideal value of 0 %. Lastly, Fig. 7.5 shows the results of the NIST statistical tests. A filled square indicates that the given test at the given temperature for the given position did not get a sufficiently high success rate (minimal 96 %). For positions X12Y2 and X12Y32, again no significant difference is visible. Position X12Y48 however does show a slight increase of randomness at higher temperatures, which is in line with the higher dependency of the CS count, visible in Fig. 7.3.

7.2.3 Conclusion

Fig. 7.6 summarizes the total number of failed tests, of all three TRNG designs. A slight decrease with temperature is again visible. It can thus be concluded that temperature has a very minor effect on the randomness of the generated data. A reason for this effect could be the increased thermal noise. It is also clear that the CS

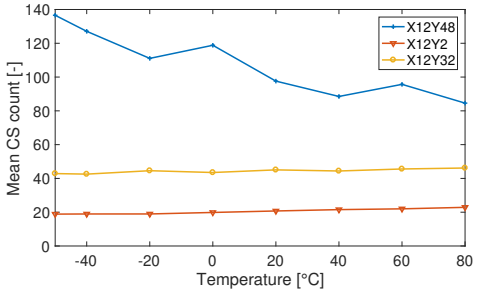
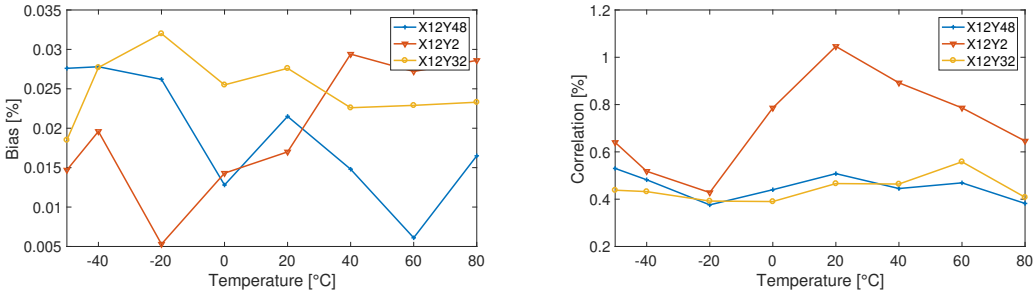


Figure 7.3: Mean CS count values at different surrounding temperatures.



(a) Bias (b) Correlation

Figure 7.4: Bias and correlation of the output data at different surrounding temperatures.

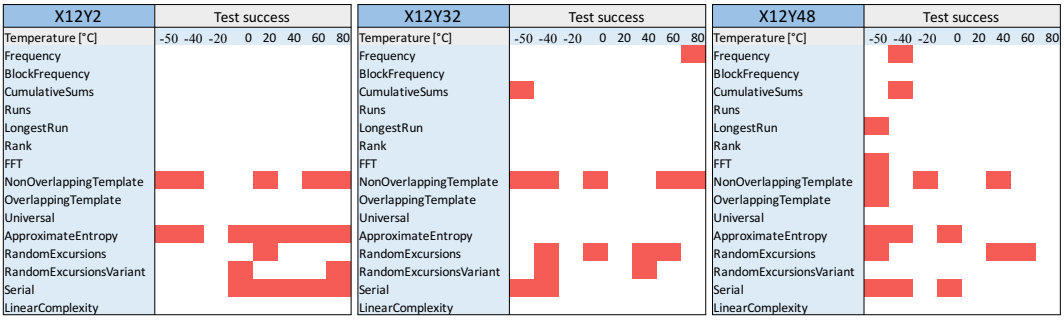


Figure 7.5: Failed tests for each TRNG design at different surrounding temperatures.

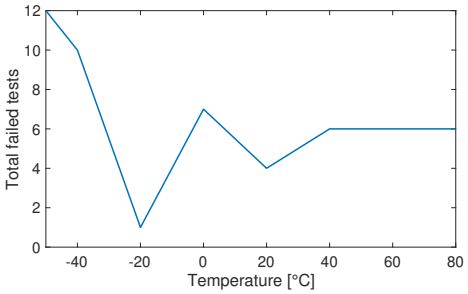


Figure 7.6: Total failed tests, each TRNG design summed up at different surrounding temperatures.

count values can be used as an indicator to detect when the device is under attack. Reduced values do in general mean reduced randomness.

7.3 Voltage attack

7.3.1 Attack methodology

The second attack is a voltage attack. The power supply voltage is lowered from the standard value of 1.2 V to 0.9 V in steps of 0.1 V. To enable this change in supply voltage, another FPGA board is used. The original *Spartan 6 LX9 Microboard* from *Avnet* is replaced by a custom made board, especially designed to study TRNGs and Physical Unclonable Functions (PUFs): the Hardware Enabled CrypTO and Randomness (HECTOR) evaluation board version 1.1/1.2. HECTOR is a three-year project, funded by the European Union's Horizon 2020 research and innovation programme [46]. The board consists out of two modules: a motherboard and interchangeable daughterboards. The motherboard is build around the *Microsemi SmartFusion2 M2S025-FGG484* System on Chip (SoC). It contains an ARM processor together with some FPGA logic elements. This chip is only responsible for the communication from the TRNG to the PC. In this way, the FPGA containing the TRNG can be relieved from all tasks not related with the actual TRNG functionality. The motherboard also contains switches that allow a simple manipulation of the supply voltage of the FPGA containing the TRNG. The daughterboard contains then the actual FPGA with TRNG on it. Multiple options are available and for ease of the porting process, a daughterboard with also a *Spartan 6* FPGA is chosen. More specifically a *Xilinx Spartan 6 XC6SLX16-2FTG256* FPGA. The two modules can be connected through either a Serial AT Attachment (SATA) or a High-Definition Multimedia Interface (HDMI) (for longer connections e.g. testing the daughterboard inside a temperature chamber) connector.

When porting the design to this new hardware platform, the positions of the STRs had to be changed. This change led to three new positions: X22Y2, X20Y2 and X8Y56. Their characteristics (at the nominal voltage of 1.2 V) are presented in Table 7.1.

Table 7.1: Properties of the TRNGs in function of the different positions

Position	X22Y2	X20Y2	X8Y56
Mean CS counter value	33.6106	57.908	58.843
Standard deviation CS counter value	0.9067	2.2416	2.127
Coefficient of variation [%]	2.698	3.871	3.615
Frequency STR 1 [MHz]	75.495	74.835	75.550
Frequency STR 2 [MHz]	77.013	74.954	74.616
Minimal post processing filter order	4	3	3
Throughput [Mbit/s]	1.953	2.604	2.604

Note that there are some differences between the previous implementation. Firstly, the STR frequencies are lower. Secondly, the CV is on average smaller. It can be due to a reduced gate jitter magnitude of the LUTs. This lower CV is then also reflected in the minimal post processing filter order, needed to pass all the NIST tests. A third difference is that the used system clock is now 125 MHz instead of the previous 100 MHz. This higher clock frequency means that the sampling speed and thus also the throughput is higher.

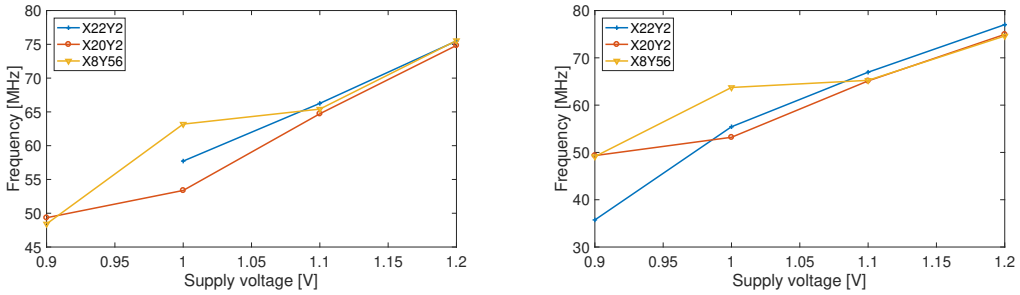
7.3.2 Results

The frequencies of the different TRNG designs are given in Fig. 7.7. Both frequencies decrease with decreasing supply voltage. This decrease corresponds with the theory, which predicts an increased propagation delay due to a reduced supply voltage. The difference in frequency between the STRs is showed in Fig. 7.8. For all three positions, it stays fairly constant. The mean CS count values, presented in Fig. 7.9, do however decrease with decreasing supply voltage. This effect cannot be caused by the difference in frequency between the STRs, as this frequency stays constant. A reason for the decrease of mean CS count can be the more bursty behaviour of the STRs at lower voltage. For placement X22Y2 at 0.9 V, STR 1 stopped oscillating. This halt caused the TRNG to fail completely and the output got stuck at an eight-bit sequence. Therefore, this data point is not added to the graphs.

The bias and correlation of the output data is showed in Fig. 7.10. Especially at 0.9 V, there is a clear reduction of randomness visible. Fig. 7.11 shows the results of the NIST tests. Again, a filled square marks a failed test (less than 96 % success rate).

7.3.3 Conclusion

The previous results definitely show a dependency on supply voltage. This dependency is also confirmed by Fig. 7.12. Here, the total number of failed tests and the total success rate are shown. There is a trend of reduced randomness when the supply



(a) STR 1 (b) STR 2

Figure 7.7: Frequency of both the STRs at different supply voltages.

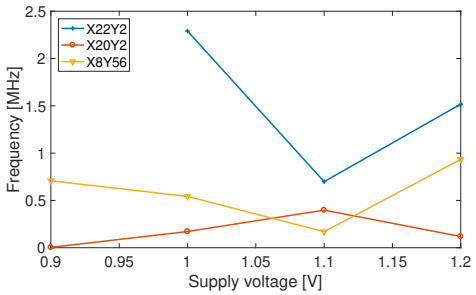


Figure 7.8: Frequency difference between both the STRs at different supply voltages.

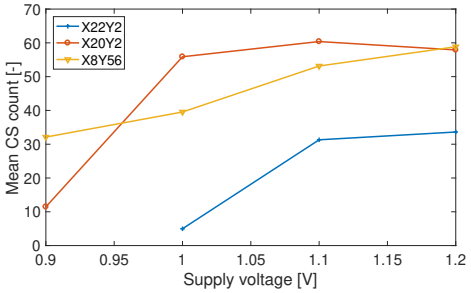
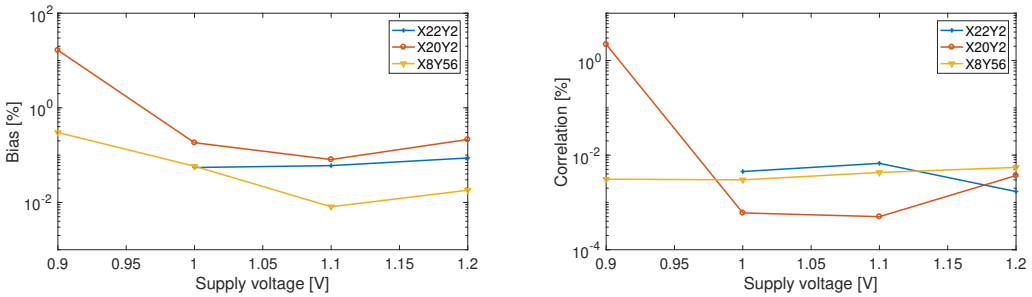


Figure 7.9: Mean CS count values at different supply voltages.



(a) Bias (b) Correlation

Figure 7.10: Bias and correlation of the output data at different supply voltages.

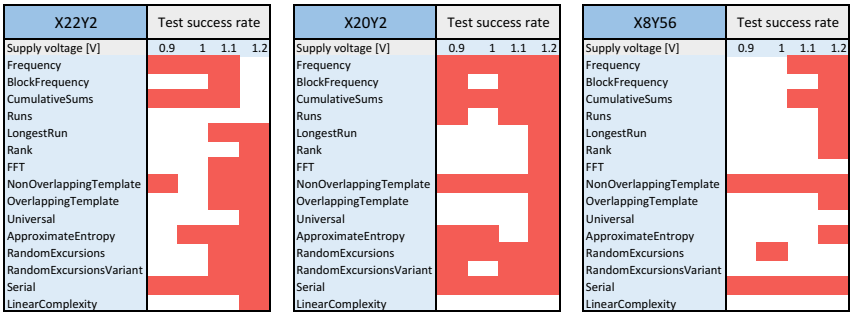
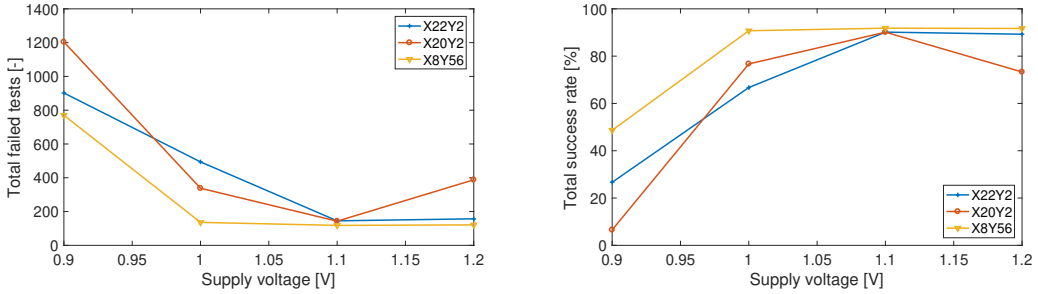


Figure 7.11: Failed tests for each position at different supply voltages.



(a) Total failed tests (b) Success rate

Figure 7.12: Total number of failed tests and corresponding success rate for each supply voltage.

voltage lowers. It can be caused by either a larger offset between the two rings and possibly also by a larger burst magnitude for both STRs.

Chapter 8

On-line testing

This chapter brings together the findings of the previous chapters. A strategy to implement effective and efficient tests has been carried out and the tests are validated under a voltage attack. The first section gives an introduction to on-line testing. The second section describes the methodology and the last section discusses the results and gives a conclusion.

8.1 Introduction to on-line testing

As already explained earlier, on-line testing is needed to monitor the quality of the generated output data. These tests should happen with low latency and low area and energy requirements. Therefore, it is not feasible to try to implement the extensive test suits illustrated in Chap. 4. Specially designed on-line testing modules are needed to handle this task.

A general methodology to design such on-line tests is given by [15]. Here, they propose the TRNG On-the-fly Testing for Attack detection using Lightweight hardware (TOTAL) method. The method roughly proceeds according to the following steps:

- **Data collection**: collect data from the TRNG both when it is under standard operating conditions and when it is under attack, with possibly multiple attacking efforts.
- **Detection of useful features**: determine then which features show a significant change whether or not the TRNG is under attack. It implies according to the authors of [15], a ranking based on the feature's usefulness. This usefulness is defined as the chance that an attack is detected ($1 - P_2$ and P_2 the chance of a type II error, false negative).
- **Implementation**: a hardware implementation is designed to trigger an alarm if one of the previously selected features goes outside its predefined boundaries. This implementation usually does not have large area or energy requirements, as it only consists of checking if a value is larger or smaller than some constant and raising an alarm if needed.

- **Validation**: the last step is redoing the attacks and checking if indeed a possible attack can be noticed by the on-line testing.

8.2 Methodology

The previously explained steps are executed for the designed TRNG.

8.2.1 Data collection

In this step, as much data as possible should be collected from the TRNG. It has already been done when attacking and evaluating the TRNG in Chaps. 6 and 7. Data under temperature and voltage attacks have been recorded. Only the voltage attack showed a significant reduction in randomness, therefore this data is used to extract useful features.

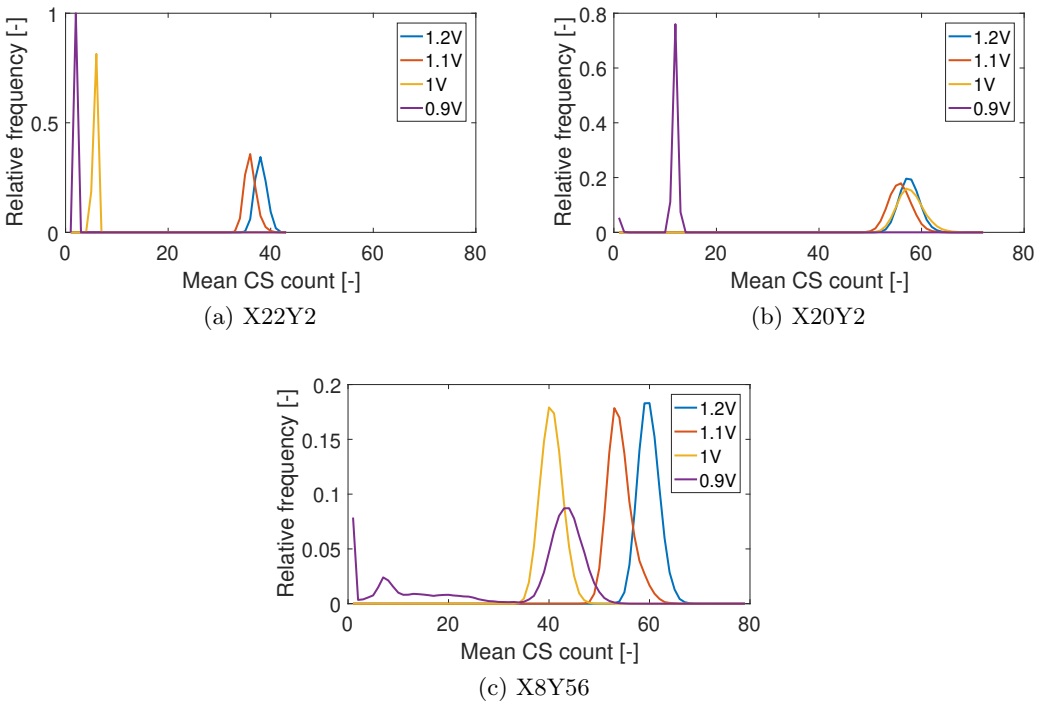
8.2.2 Detection of useful features

There exist two types of features: those based on the output data (bias, correlation, matching templates, etc.) and those based on the internal workings. Here, only features derived from the internal workings are examined. There are three possibilities: CS count, STR frequency and burst magnitude. All three of them have already showed to change in significant amount during attack. Figs. 8.1, 8.2 and 8.3 show the behaviour of the mean CS count, STR frequency and burst magnitude respectively for each of the tested positions: X22Y2, X20Y2 and X8Y56. Clearly both the CS count and the STR frequency can be used for attack detection. They have a large usefulness of often almost 100 %. The burst magnitude is not suited to detect voltage attacks.

8.2.3 Implementation

For each of the features, acceptable boundaries need to be defined. This choice is however not trivial, because it is not certain which levels of randomness are acceptable. For example, it could be that the output data generated by the TRNG at 1.1 V is still sufficient for certain applications. This decision has to be made by the user of the TRNG. Another problem is that there is no standard way to qualify the quality of the output data. One could simply use the bias or correlation, but these are often not very sensitive. Here, the total success rate of the NIST test suite is used as a measure for the quality of the generated data. More specifically, at a threshold of 70 % success rate. For the reader's convenience, the success rates are repeated in Table 8.1.

According to these success rates and the set threshold, it can be determined which values of CS count and STR frequency are acceptable. A threshold is then determined at their turn. Notice that not for all positions, there exists a threshold capable of separating the two regions with zero chance of type I or II errors. For example the CS count at position X8Y56 overlaps for voltages of 1 V and 0.9 V. This



(a) X22Y2

(b) X20Y2

(c) X8Y56

Figure 8.1: Relative frequency of occurrence of the CS count for each of the supply voltages and for every position.

Table 8.1: Success rates for the voltage attack for every position.

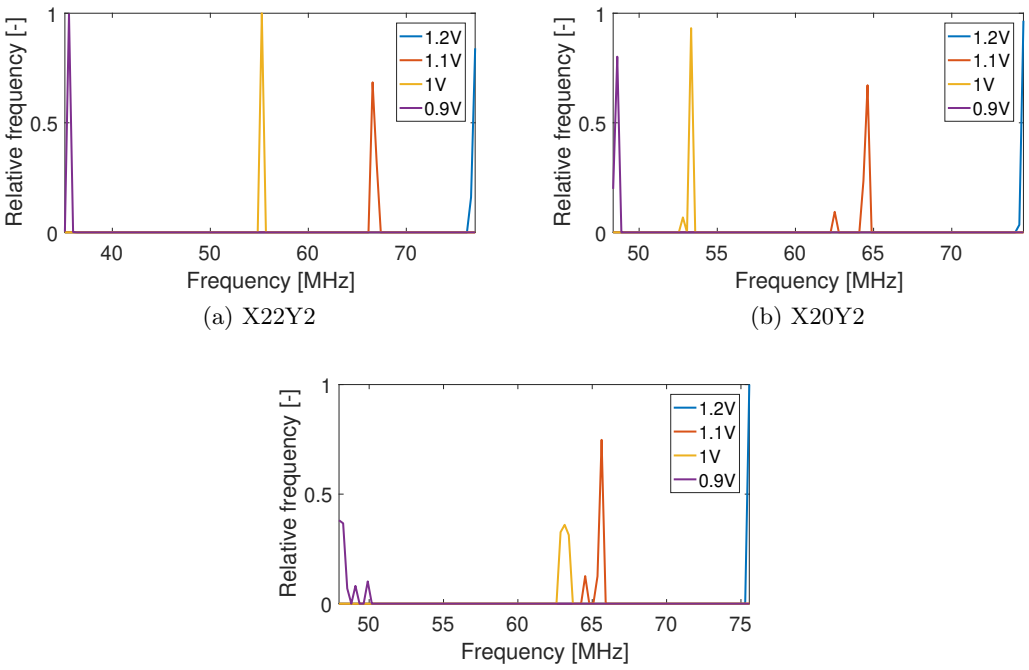
Voltage [V]	X22Y2	X20Y2	X8Y56
1.2	89.27%	73.31%	91.70%
1.1	90.15%	90.13%	91.83%
1.0	66.72%	76.71%	90.74%
0.9	26.67%	6.53%	48.67%

problem is overcome by using two features, both CS count and STR frequency. When one of them fails, an alarm is generated and the attack can be detected. Another problem is that not every position produces the same values for CS count and STR frequency. A threshold needs to be defined for every position separately.

Table 8.2 shows the different thresholds. They are found by taking the threshold that gives the smallest error. When a whole interval can separate good from bad TRNG behaviour, the middle value of that interval is taken as the threshold.

8.2.4 Validation

The voltage attacks is redone and the results are shown in Table 8.3. Almost all attacks are detected. This result is due to the large separation (high usefulness) of



(a) X22Y2

(b) X20Y2

(c) X8Y56

Figure 8.2: Relative frequency of occurrence of the STR frequency for each of the supply voltages and for every position.

Table 8.2: Thresholds for the two features and for every position.

Feature	X22Y2	X20Y2	X8Y56
CS count	21	32	35
STR frequency [MHz]	60.91	50.70	56.39

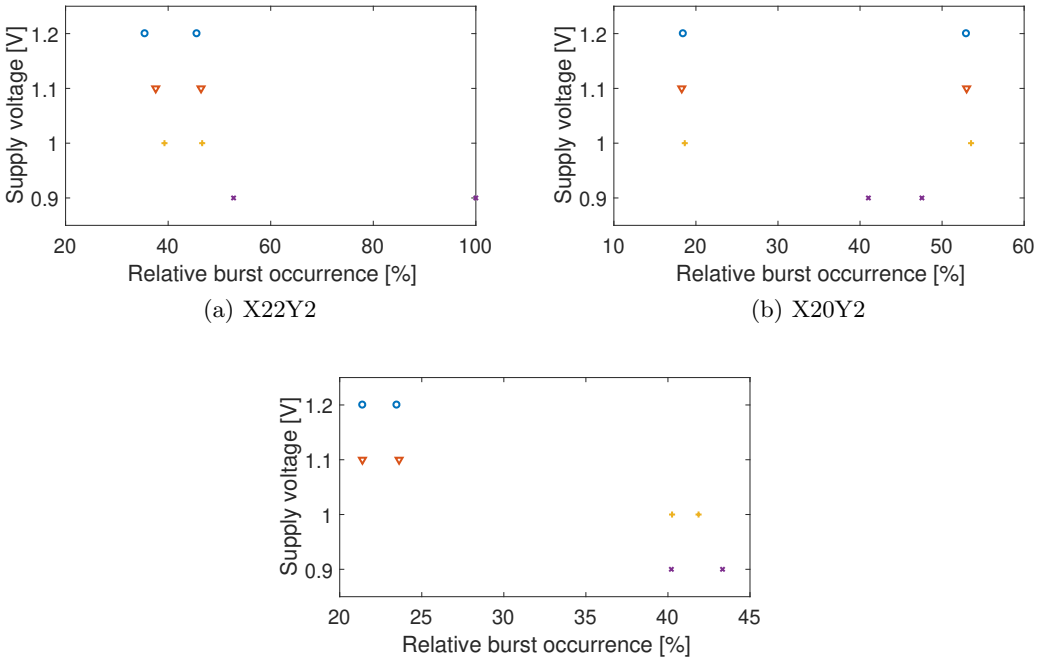
Table 8.3: Test alarm rate for every position.

Voltage [V]	X22Y2	X20Y2	X8Y56
1.2	0 %	0 %	0 %
1.1	0 %	0 %	0 %
1.0	100 %	0 %	2.12 %
0.9	100 %	100 %	68.24 %

the extracted features.

8.3 Conclusion

On-line testing turned out to be very effective thanks to the high sensitivity oft the features on the supply voltage. However the problem of finding a general threshold



(a) X22Y2

(b) X20Y2

(c) X8Y56

Figure 8.3: Percentage of the time that the burst detector raises an alarm at different supply voltages and for every STR at every position.

that would be valid for all positions and other attacks still exists. Some kind of dynamic threshold would solve the variability between different positions. Here, the threshold would be automatically adjusted if the TRNG behaves differently when implemented in another position. It however leads to a chicken and egg problem: to determine the threshold, the TRNG has to be sure that it is not under attack. However, that cannot be known with any certainty if there is no good threshold that would indicate if the TRNG is under attack or not.

With the advantages of on-line testing are also some additional costs related. The implementation of the tests requires area and increases the power consumption. Due to the simplicity of the tests however, these costs are kept to a bare minimum. Different attack strategies require different features to be detected. The STR frequency and the CS count are suitable for detecting power supply voltage variations, the STR frequency can also indicate changing surrounding temperature and the burst magnitude can be used to indicate an implementation defect.

The tests implemented in this section are not perfect. Especially at position X8Y56, at 1.0 V, they generated a false alarm rate of 2.12 %. This false alarm rate reduces the throughput of the generated output data because the data is discarded when a false alarm occurs.

The CS count test can be implemented with a very short latency of at most

eight bits. Every time a new byte is generated, a new CS count is available and tested. The STR frequency needs time to accurately calculate the frequency. These measurements have a rate of 0.122 MHz, which corresponds with a latency of 22 bits at most. The burst magnitude also needs some integration time to get an average result. This duration however has not yet been experimentally determined.

Chapter 9

Conclusion and future work

9.1 Conclusion

The goal of this thesis was to perform a robustness analysis of a TRNG that combines both the advantages of STRs and of the CS principle. Using the results of this analysis, interesting features could be selected to construct sensitive and efficient on-line tests.

First, a model was created that could predict the impact of important parameters on the behaviour of this TRNG. The most important conclusion was that careful placement and routing is very important to ensure that the rings evolve in an evenly spread mode. The amount of asymmetry between forward and backward delays in each ring should be kept at a minimum. When this placement and routing would be left over to the automatic synthesis tool, there is absolutely no guarantee at all that the rings behave as intended. It means that any stochastic model which assumes that the STRs are in evenly spread mode, does not hold and true randomness cannot be guaranteed any more. Another important result from the model is that although the output only contains pseudo randomness (in theory it is then perfectly deterministic and therefore predictable), it is generally very hard to almost impossible to detect this lack of true randomness. It was illustrated in the model by reducing the LUT output jitter (the only random component in the model) to zero and noticing that statistical tests are not able to detect it.

The second step was the actual design and implementation of the TRNG. The design was created using the *Xilinx* ISE design suite, version 14.7. This design is then implemented on two different FPGAs: a *Spartan 6 XC6SLX9-2CSG324C* and a *Spartan 6 XC6SLX16-2FTG256* FPGA. The second one was available on the HECTOR board, a specially designed board to perform physical attacks on TRNGs. Besides the essential building blocks: two STRs and eight CS units, additional components are added to constantly monitor the state of the TRNG. Frequency counters keep track of the STR frequency, CS counters check if the difference in frequency between the STRs is kept within limits, BRAM memory buffers the generated data before it is send to the computer and a UART interface handles this sending. The STRs and the CS units where manually placed to ensure a symmetric

topology. Care had to be taken that the loading of the CS units on the STRs was also symmetric. It was done by adding buffer in between them. The TRNG is implemented on five different positions inside the FPGA, to measure the effect placement has on the behaviour of the TRNG. Two out of these five positions where able to pass the NIST SP 800-22 and AIS 31 test suites without needing any additional post processing. The maximal XOR post processing filter order needed to pass the tests was equal to three.

The next step studies the effect of an imperfect implementation on the behaviour of the TRNG. All the guidelines, proposed after studying the results from the model where discarded and it is tried to get the TRNG out of its stable operating region. Both the amount of asymmetry and the amount of noise, created by neighbouring activity, is increased. To increase the asymmetry, additional buffers are placed to increase the forward delay at each stage of the STR. Measurements confirm the results from the model that predicted that the amount of bursty behaviour would drastically increase with routing asymmetry. To check the dependency on surrounding activity, small ROs are placed around the TRNG. These ROs add noise on the supply voltage and due to their large current consumption, the supply voltage also lowers locally. This lowering in supply voltage causes the STR frequency to go down. The success rate of the tests stayed within acceptable limits, but that can be caused by the pseudo random component of the output data. A last experiment measured the effect of placement on the TRNG. The results showed that the TRNG is certainly very sensitive to placement. This sensitivity can be devoted to the different types of slices that can reduce matching between the two STRs.

The following step was to apply physical attacks on the TRNG to assess its robustness against variations in operation conditions. Two attacks are performed: a surrounding temperature- and a power supply voltage attack. The first attack uses the *ESPEC SH-662* temperature and humidity chamber, it can change the temperature ranging from -60 °C to 150 °C and also regulate the humidity, which should be kept low when electronics are placed inside. Before each attack, an isolation test is executed. This test checks if the surrounding logic (BRAM, UART interface, frequency counters, etc.) still works correctly at the changed operating condition. A slight dependence of the measured randomness on the surrounding temperature was visible. It was however only clear after extensively testing the generated random data. Simple statistical tests such as bias and correlation could not detect this change. The voltage attack makes use of another FPGA on the HECTOR board. The power supply was ranged from 1.2 V (standard value) to 0.9 V (underpowering). Especially at 0.9 V, a clear reduction in randomness was visible. This change in voltage was reflected in the mean CS count as well as in the frequency of both STRs.

The last step is to take together the results from the previous experiments and use this knowledge to construct sensitive on-line tests. The selected features where mean CS count and STR frequency. The TOTAL methodology was followed and a on-line test module was successfully implemented. A threshold on the success rate of the NIST SP 800-22 test suite can be chosen and it defines the acceptable interval of both the selected features.

9.2 Future work

There are still a lot of ideas that did not get completely worked out yet. The important ones are listed below:

- A lot of other attacks can be performed on this TRNG. An EM radiation attack, where EM radiation, near the frequency of the movement of tokens through the STRs, can be injected via a probe onto the TRNG. The goal of this attack is to lock the STRs at exactly the same phase. Other attacks such as: power supply glitch injection, clock glitch injection or over-clocking (over-sampling) are also possible.
- As the TRNG is very sensitive to its implementation, the performance on other FPGA families and other synthesis tools still need to be examined.
- The thresholds for the on-line tests still need to be calculated manually for every implementation of the TRNG. Further investigation is needed to initialize this threshold automatically during start-up of the TRNG.
- When this design is to be used in larger systems, a more automatic way of implementation is needed. The users of the system do not know all the detail concerning TRNG design. A tool must be available that ensures a robust implementation.

Bibliography

[1] J.-L. Danger, S. Guilley, and P. Hoogvorst, "High speed true random number generator based on open loop structures in fpgas," *Microelectronics journal*, vol. 40, no. 11, pp. 1650–1656, 2009.

[2] P. Z. Wieczorek and K. Golofit, "Dual-metastability time-competitive true random number generator," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 1, pp. 134–145, 2014.

[3] M. Baudet, D. Lubicz, J. Micolod, and A. Tassiaux, "On the security of oscillator-based random number generators," *Journal of cryptology*, vol. 24, no. 2, pp. 398–425, 2011.

[4] M. Varchola and M. Drutarovsky, "New high entropy element for fpga based true random number generators," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2010, pp. 351–365.

[5] L. E. Bassham III, A. L. Rukhin, J. Soto, J. R. Nechvatal, M. E. Smid, E. B. Barker, S. D. Leigh, M. Levenson, M. Vangel, D. L. Banks *et al.*, "Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications," 2010.

[6] W. Killmann and W. Schindler, "A proposal for: Functionality classes for random number generators," *ser. BDI, Bonn*, 2011.

[7] A. T. Markettos and S. W. Moore, "The frequency injection attack on ring-oscillator-based true random number generators," in *Cryptographic Hardware and Embedded Systems-CHES 2009*. Springer, 2009, pp. 317–331.

[8] P. Bayon, L. Bossuet, A. Aubert, V. Fischer, F. Poucheret, B. Robisson, and P. Maurine, "Contactless electromagnetic active attack on ring oscillator based true random number generator," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2012, pp. 151–166.

[9] H. Martin, T. Korak, E. San Millán, and M. Hutter, "Fault attacks on strngs: Impact of glitches, temperature, and underpowering on randomness," *IEEE transactions on information forensics and security*, vol. 10, no. 2, pp. 266–277, 2015.

[10] A. Cherkaoui, V. Fischer, L. Fesquet, and A. Aubert, "A very high speed true random number generator with entropy assessment," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2013, pp. 179–196.

[11] H. Martin, P. Peris-Lopez, J. E. Tapiador, and E. San Millan, "A new trng based on coherent sampling with self-timed rings," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 1, pp. 91–100, 2016.

[12] A. Cherkaoui, V. Fischer, A. Aubert, and L. Fesquet, "A self-timed ring based true random number generator," in *Asynchronous Circuits and Systems (ASYNC), 2013 IEEE 19th International Symposium on*. IEEE, 2013, pp. 99–106.

[13] J. Hamon, L. Fesquet, B. Miscopein, and M. Renaudin, "High-level time-accurate model for the design of self-timed ring oscillators," in *Asynchronous Circuits and Systems, 2008. ASYNC'08. 14th IEEE International Symposium on*. IEEE, 2008, pp. 29–38.

[14] J. C. Ebergen, S. Fairbanks, and I. E. Sutherland, "Predicting performance of micropipelines using charlie diagrams," in *Advanced Research in Asynchronous Circuits and Systems, 1998. Proceedings. 1998 Fourth International Symposium on*. IEEE, 1998, pp. 238–246.

[15] B. Yang, V. Rožić, N. Mentens, W. Dehaene, and I. Verbauwhede, "Total: Trng on-the-fly testing for attack detection using lightweight hardware," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*. IEEE, 2016, pp. 127–132.

[16] Y. Liu, R. C. Cheung, and H. Wong, "A bias-bounded digital true random number generator architecture," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 1, pp. 133–144, 2017.

[17] A. Cherkaoui, V. Fischer, A. Aubert, and L. Fesquet, "A self-timed ring based true random number generator." IEEE, May 2013, pp. 99–106.

[18] J.-L. Danger, S. Guilley, and P. Hoogvorst, "Fast true random generator in fpgas," in *Circuits and Systems, 2007. NEWCAS 2007. IEEE Northeast Workshop on*. IEEE, 2007, pp. 506–509.

[19] F. Lozach, M. Ben-Romdhane, T. Graba, and J.-L. Danger, "Fpga design of an open-loop true random number generator," in *Digital System Design (DSD), 2013 Euromicro Conference on*. IEEE, 2013, pp. 615–622.

[20] A. Winstanley and M. Greenstreet, "Temporal properties of self-timed rings," in *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*. Springer, 2001, pp. 140–154.

[21] J. Yang, Y. Ma, T. Chen, J. Lin, and J. Jing, "Extracting more entropy for trngs based on coherent sampling," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2016, pp. 694–709.

[22] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer Science & Business Media, 2008, vol. 31.

[23] T. Kasper, D. Oswald, and C. Paar, "Em side-channel attacks on commercial contactless smartcards using low-cost equipment," in *Information Security Applications*. Springer, 2009, pp. 79–93.

[24] *ISO/IEC 18031 Random Bit Generation*, November, 2011.

[25] *Cryptographic Engineering*. Boston, MA: Springer US, 2009.

[26] A. Klein, *Stream ciphers*. Springer, 2013.

[27] J. Von Neumann, "13. various techniques used in connection with random digits," *Appl. Math Ser*, vol. 12, no. 36-38, p. 3, 1951.

[28] M. Dichtl, "Bad and good ways of post-processing biased physical random numbers," in *International Workshop on Fast Software Encryption*. Springer, 2007, pp. 137–152.

[29] S.-H. Kwok, Y.-L. Ee, G. Chew, K. Zheng, K. Khoo, and C.-H. Tan, "A comparison of post-processing techniques for biased random number generators," vol. 6633, 2011, pp. 175–190.

[30] P. R. Gray, P. Hurst, R. G. Meyer, and S. Lewis, *Analysis and design of analog integrated circuits*. Wiley, 2001.

[31] B. Jun and P. Kocher, "The intel random number generator," *Cryptography Research Inc. white paper*, 1999.

[32] Q. Li, Q. Liu, and J. Niu, "Chaotic oscillator with potentials in trng and adc." IEEE, July 2012, pp. 397–400.

[33] J. V. Evangelista, J. A. Artiles, D. P. Chaves, and C. Pimentel, "Emitter-coupled pair chaotic generator circuit," *AEUE - International Journal of Electronics and Communications*, vol. 77, pp. 112–117, July 2017.

[34] B. Sunar, W. J. Martin, and D. R. Stinson, "A provably secure true random number generator with built-in tolerance to active attacks," *IEEE Transactions on computers*, vol. 56, no. 1, 2007.

[35] J. D. Golic, "New methods for digital generation and postprocessing of random data," *IEEE Transactions on Computers*, vol. 55, no. 10, pp. 1217–1229, 2006.

[36] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720–738, 1989.

[37] A. Cherkaoui, V. Fischer, A. Aubert, and L. Fesquet, "Comparison of self-timed ring and inverter ring oscillators as entropy sources in fpgas," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*. IEEE, 2012, pp. 1325–1330.

[38] V. Fischer and M. Drutarovskỳ, "True random number generator embedded in reconfigurable hardware," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2002, pp. 415–430.

[39] 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.

[40] B. Valtchanov, V. Fischer, and A. Aubert, "A coherent sampling-based method for estimating the jitter used as entropy source for true random number generators," in *SAMPTA'09*, 2009, pp. Special–session.

[41] Xilinx, "Spartan-6 fpga data sheet: Dc and switching characteristics," 2011.

[42] *Spartan-6 Family Overview*, XILINX, 2011.

[43] *Spartan-6 FPGA Configurable Logic Block User Guide*, XILINX, 2010.

[44] H. Martin, P. Peris-Lopez, J. E. Tapiador, and E. San Millan, "A new trng based on coherent sampling with self-timed rings," *Industrial Informatics, IEEE Transactions on*, vol. 12, no. 1, pp. 91–100, February 2016.

[45] G. Timothy and D. Aaron. (2013) Documented verilog uart. [Online]. Available: https://github.com/cyrozap/osdvu/tree/276dd06ee5c01fc49e522272ceef088a15af3fe5

[46] (2015) Hardware enabled crypto and randomness. [Online]. Available: https://hector-project.eu/

Fiche masterproef

Student: Adriaan Peetermans

Titel: Attacking and Securing Hardware Random Number Generators

Nederlandse titel: Aanvallen en Beveiligen van Hardware True Random Number Generators

UDC: 621.3

Korte inhoud:
Een True Random Number Generator (TRNG) is een kritisch component in veel ingebedde veiligheidsapplicaties. Omdat ze als input dienen voor cryptografische algoritmes, kan het belang van hun beveiliging niet overschat worden. Het feit dat TRNGs dikwijls kleine analoge fenomenen (ruis) versterken, maakt ze een zeer kwetsbaar deel van het systeem. In dit werk, bestuderen we de effecten van zowel fysieke aanvallen als slechte implementaties op het gedrag van een Self-Timed Ring (STR)-gebaseerde TRNG. Om deze resultaten te ondersteunen, creëerden we een model van de STRs in *MATLAB* zodat de afhankelijk van kritische parameters ingeschat kan worden. Daarna ontwierpen we online testen als een tegenmaatregel en beoordeelden de performantie via experimenten.

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: elektrotechniek, optie Elektronica en geïntegreerde schakelingen

Promotor: Prof. dr. ir. Ingrid Verbauwhede

Assessoren: Prof. dr. ir. Bart Preneel
Prof. dr. ir. Wim Dehaene

Begeleiders: Dr. Vladimir Rožić
Bohan Yang
Miloš Grujić
Dr. Josep Balasch