
**Pedro Miguel Nunes Mendes**

Licenciado em Ciências da Engenharia Electrotécnica
e de Computadores

# Random Number Generator based on Ring Oscillators for IoT applications

Dissertação para obtenção do Grau de Mestre em

**Engenharia Electrotécnica e de Computadores**

Orientador: Doutor João Palma Goes, Professor Associado,
Faculdade de Ciências e Tecnologia
da Universidade Nova de Lisboa

Júri

| | |
|---|---|
| Presidente: | Doutor Rui Morgado Dinis, Professor Associado - FCT/UNL |
| Arguente: | Doutor Luís Santos Gomes, Professor Associado - FCT/UNL |
| Vogal: | Doutor João Palma Goes, Professor Associado - FCT/UNL |


FACULDADE DE
CIÊNCIAS E TECNOLOGIA
**UNIVERSIDADE NOVA** DE LISBOA

**Setembro, 2017**

**Random Number Generator**
**based on Ring Oscillators**
**for IoT applications**



Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "unlthesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/unlthesis [2] http://www.di.fct.unl.pt

*Aos meus Pais, que sempre me incentivaram a seguir aquilo que gosto...*

# Acknowledgements

I would like to acknowledge the contributions of a number of people who were very helpful and supportive in my research.

I extend my sincere appreciation to my principal advisor, Professor João Goes, whose wisdom and expertise enabled the successful completion of this research. Would like to extend my gratitude to Professors Anikó Costa, Filipe Moutinho, Ricardo Madeira and Raul Rato for the help on the development of this Thesis.

Thanks to Ihor Vasyltsov for the support and helpful advices, which significantly increased the quality of this project.

To my parents and my sister who always supported me during my journey and gave me all the tools without ever giving up on me.

A special thank you to Daniela Prudêncio, for all her love and support during the most difficult times.

Finally I wanted to thank all my friends who helped me during all these years and who always lifted me up during the academic journey.

To all of you, thank you very much.

# Abstract

Since the beginning of times that human beings were presented with the difficult of protecting their data. In all wars, cryptographic systems where essential to wining. Nowadays, fighting global terrorism makes cryptography of paramount importance in communication and security of critical systems. Cryptographic methods present a real challenge, so hard that it is as difficult to create as it is to crack.

Since the first appearances of Integrated Circuits, methods have been created to protect the data on them, the same happened when the first networks appeared and nowadays we still face the same challenge. Centuries have passed and the definition of the random word still creates confusion when trying to define it. This thesis proposes a random generation method which proved to be effective according to worldwide standards with the new emerging technologies of the IoT in mind.

A True Random Number Generator (TRNG) based on ring oscillators, implemented in a Field Programmable Gate Array (FPGA) platform is proposed and evaluated. Based on some new concepts on ring oscillators, jitter noise was analysed and used as a main noise source to create randomness.

The main aim of this project was to investigate and develop of a secure system that met the international requirements for IoT purposes.

**Keywords:** Ring oscillators, TRNG,FPGA , Random Number Generator,IoT ...

# Resumo

Desde o inicio dos tempos, os seres humanos defrontam-se com o desafio de proteger informação. Em todas as guerras, os sistemas criptográficos são essenciais para vencer. Hoje em dia, tendo em conta o terrorismo global que enfrentamos, a criptografia é um assunto de extrema importância na comunicação e na segurança de sistemas críticos. Os métodos criptográficos apresentam assim um desafio real, tão difícil de criar como de decifrar.

Desde o aparecimento dos primeiros Circuitos Integrados, foram criados métodos para proteger os dados que estes tratam, o mesmo aconteceu quando surgiram as primeiras redes e, hoje em dia, ainda enfrentamos o mesmo desafio com o aparecimento de novas tecnologias.

Os séculos passaram e a definição da palavra 'aleatório' ainda cria dificuldades em quem a tenta definir ou medir. Esta tese propõe um método de geração de números aleatórios que se mostrou efetivo de acordo com padrões mundiais e com o objetivo de aplicação nas novas tecnologias da IoT em mente.

Um verdadeiro gerador de números aleatórios (TRNG) baseado em osciladores de anel, implementado através de uma plataforma FPGA é proposta e avaliada. Com base em alguns novos conceitos sobre osciladores de anel, o ruído jitter foi analisado e usado como fonte principal incerteza para gerar aleatoriedade nos resultados.

O principal objetivo deste projeto foi investigar e desenvolver um sistema seguro que atendeu aos requisitos internacionais de segurança para aplicações em tecnologias da IoT.

**Palavras-chave:** Osciladores em anel, gerador de números aleatorios, FPGA, IoT ...

# Contents

# List of Figures

# List of Tables

# Listings

# Acronyms

| | |
|---|---|
| A/D | Analog to Digital. |
| ADC | Analog to Digital Converter. |
| ASIC | Application Specific Integrated Circuit. |
| BSI | Bundesamt in der Informationstechnik. |
| CASR | Cellular Automata Shift Register. |
| CCO | Current Controled Oscillator. |
| CLB | Configurable Logic Block. |
| DAS | Digitised Analog Signal. |
| DC | Direct Current. |
| DFT | Discrete Fourier Transform. |
| FFT | Fast Fourier Transform. |
| FIPS | Federal Information Processing Standards. |
| FIRO | Fibonnaci Ring Oscillator. |
| FPGA | Field Programmable Gate Array. |
| GARO | Galois Ring Oscillator. |

| | |
|---|---|
| HEX | Hexadecimal. |
| HPF | High Pass Filter. |
| I/O | Input/Output. |
| IC | Integrated Circuit. |
| IoT | Internet of Things. |
| JTAG | Joint Test Action Group. |
| LFSR | Linear Feedback Shift Register. |
| LUT | LookUp Table. |
| NIST | National Institute of Standards and Technology. |
| PDF | Probability Density Function. |
| PP | Post-Processing. |
| PRNG | Pseudo Random Number Generator. |
| PUF | Physical Unlonable Function. |
| RAM | Random Acess Memory. |
| RNG | Random Number Generator. |
| RO | Ring Oscillator. |
| RS232 | Recommended Standard 232. |
| S/H | Sample and Hold. |

SCH Schematic.

SOC System On Chip.

SR Shift Register.

TRNG True Random Number Generator.

UART Universal Asynchronous Receiver/Transmitter.

USB Universal Serial Bus.

VHDL VHSIC Hardware Description Language.

XST Xilinx Synthesizer.

Chapter 1

# Introduction

## 1.1 Introduction

The science of hiding any kind of information is called cryptography. This secret intel can make you loose a war for your enemy and, therefore, protecting data must be paramount importance. This document provides a tool for cryptographic systems, namely, a true random number generator.

Some of the most asked questions regarding this matter are

*'what is actually random?'*

or

*'what means to be random ?'*

, random can be considered as something that has always the same chance of happening. Consider a toss of a coin, if the coin is perfect and completely symmetric then every time it is tossed into the air there is a chance it could drop heads or tails. Always the same chance. But this 'random-ability' has been a widely discussed topic since many centuries and how it can be evaluated and distinguished as 'random' and 'not random'. Also, how can a result be quantified as more random than other. All of these examples and questions appeared when we were creating and developing this system.

Based on the fact that our system outputs binary sequences, '1' or '0', therefore, it can be considered as an analogy to the coin toss example, to be considered as random it has to present always the same change for each of the outcomes [19].

This tool provided by our document can be useful for IoT devices since lots of them use cloud based systems and a lot of data is constantly being transferred over networks and devices, hence, a system capable of being present in both cloud based servers and devices is a requirement.

Nowadays, most of the security systems available are encrypted relying on software, consequently, there are a lot of random number generation algorithms made available for different applications. These usually are the so called Pseudo Random Number Generator (PRNG), they are composed by complex algorithms which can generate 'almost' random numbers, they present a real challenge to decrypt however sometimes it is possible to predict values, those systems present results that seems to be random but they are not, creating the possibility to hack them. This possibility, is something that its mandatory to be eliminated.

Returning to the same coin example, imagine a coin being tossed 50 times into the air, the first toss dropped heads where all the others tosses dropped tails, the coin tosser claims the probability of the toss is the same for both coin faces and asks you to guess which face will drop next. What would you answer ?

Random is something in which we can not find patterns:

> *'... this way, random is different from arbitrary, arbitrary does not imply a determinable distribution of probability like random' [8]*

By its interpretation, random presents the need to a distribution, therefore, do we need arbitrary sequences or random sequences?

The meaning of random based in today's standards on TRNG systems can only be classified as sources of 'true random' if they comprise with several requirements based on many statistic analysis in the field of mathematics [38] [26]. This document proposes a fully implemented TRNG on an FPGA Platform with extraction methods as well as a detailed analysis on some standard methods to evaluate RNG's.

## 1.2 Problem Statement

Most of our electronic devices are always connected to some kind of network and most of our private intel are saved somewhere else than our devices, consequently, there is a true need to develop a secure and reliable system to encrypt our private data. Since software encrypting methods were already proven to be not reliable due to its weaknesses. Random generation methods based on physical systems and phenomenons came to reality because they operate on more secure systems.

Hence, a secure, reliable, easy implementation random number generation method for encryption key generation is needed.

## 1.3 Goals

In order to solve the problem depicted above, we aimed to develop a TRNG based on an easy and straightforward implementation, with size requirements and reliable properties. This system would have to be small enough to be implemented on IoT devices and cloud

based servers which encrypt data. Hence, it had to be classified as a TRNG instead of a PRNG.

The main following goals have been set:

- Reliable,
- Straightforward implementation,
- low die area/size,
- and the most important, having cryptographic quality.

This last topic can also be divided into several requirements:

- Unpredictability, random sequences being statistically independent,
- Scalability,
- Consistency.

[26]

The TRNG would have to meet all the requirements.

### 1.3.1 Unpredictability

All random number generators, either pseudo or true random must be unpredictable in order to be classified as 'random'. In the case of a PRNG, knowing the initial conditions and having the right amount of time, a forward sequence can be known or predicted based of accurate guesses, this is also known as forward predictability. Considering the possibility of not knowing initial conditions, using backward predictability with a long known sequence can return those conditions and also predict some sequences in accurate guesses.

In our case, having infinite data, no patterns can be found to get a prediction of the next sequence. That's why a classification of TRNG is important and different from PRNG because since PRNGs are also unpredictable, this only happens until a certain point.

Since a random bit sequence is composed by '1's and '0's, we can continue with the coin toss example. Therefore, for all the results on the coin tosses to be independent, the joint probability of all results needs to be equal to the multiplied probability of the independent results.

$$P(A_1, A_2, \ldots, A_n) = P(A_1) \times P(A_2) \times \ldots P(A_n)$$

Considering that for an independent event, a coin toss would have $P(A) = \frac{1}{2}$ then,

$$\frac{1}{2} \times \frac{1}{2} \times \ldots = \frac{1}{2^n}$$

This way, we prove that a sequence of coin toss results is statistically independent [19]. The physical representation of what is described above is represented in figure 1.1, where *A* and *B* are individual independent sets of bits in a sequence.

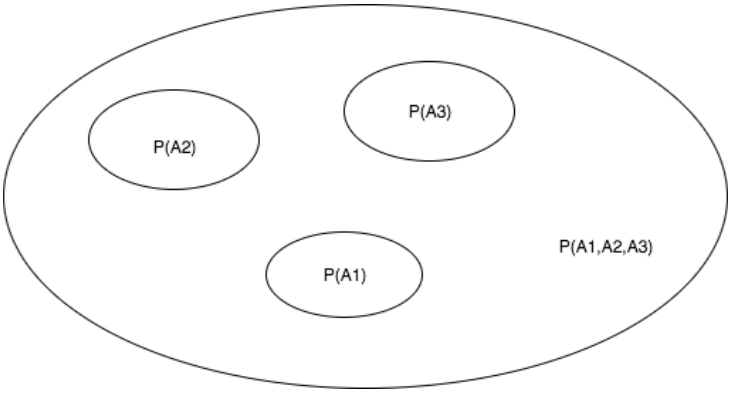



Figure 1.1: Statistical Independence

This is the kind of results we are looking to test on random sequences, these are tests that prove how random are sequences. Also, the majority of tests available come from NIST and FIPS which are described in chapter 4. The process of creating this random unpredictability has mostly been done based to analog noise sources, but new approaches were needed due to attacks on these devices [13].

### 1.3.2 Scalability

The random sequences generated must all present the same state of 'randomness'. It is only considered for something to be random if all the continuity of the sequence is evaluated as random. Hence, if only part of the sequence pass the tests, then the TRNG does not comply with the requirements.

### 1.3.3 Consistency

Just as well as scalability, consistency applies, a random sequence must be all random. Consistency evaluates that multiple 1 Mb streams are random and not derived from the same large set of data. Hence, consistency applies to multiple new random extractions whilst scalability evaluates multiples streams in a big extraction.

## 1.4 Thesis' Outline

Chapter 2 will introduce some of the basics about RNGs as well as previous configurations and techniques used to generate random numbers. Chapter 3 comprises the steps involved in the creation of the RNG. Chapter 4 comprises the statistical analysis on TRNG evaluation and some comparisons with previous implementation. Finally chapter 5 presents conclusions of the project and some future work proposal's.

## 1.5 Main contributions

The main contributions of this Thesis are:

- Based on physical phenomena, this random generation method provides truly random numbers with better results than previous approaches.
- Being the circuit designed in FPGA, it provides us a proof-of-concept as well as a straight-forward implementation.
- Lastly, the system was mainly developed to be user friendly, hence, extraction methods were implemented to obtain the random numbers.

# Chapter 2

# State of the Art

## 2.1 RNG Basics

In this chapter some RNG basic designs and conceptions will be introduced. RNG implementations and noise sources can be grouped into different categories.

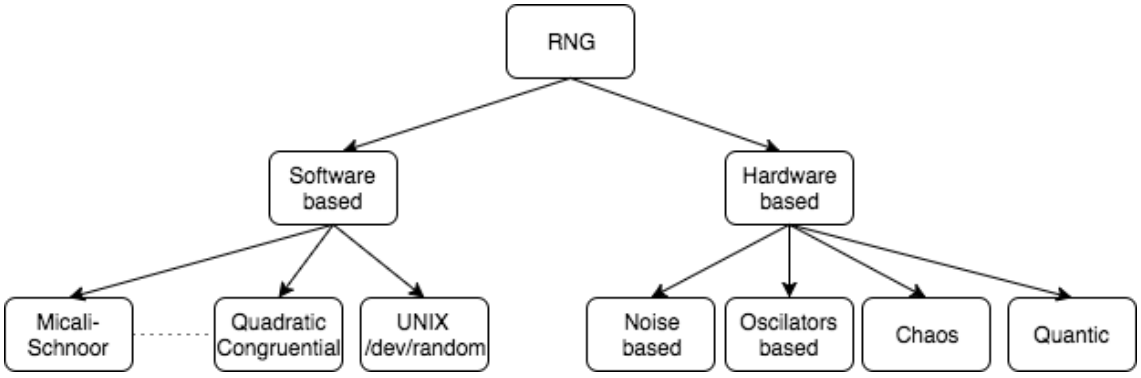



Figure 2.1: RNG type and implementation

Figure 2.1 shows a diagram with the representation of those categories, random numbers can be generated based on software, like /dev/random or /dev/urandom, which are two tools available on LINUX/UNIX environments, that will also be tested alongside our implementation, also, some other software can generate numbers based on calculations which appear to be random. Some other software algorithms are described in [26].

Hardware RNG's is where the magic happens and where simple apparently random strings appear to be random, most of hardware RNG's are TRNG's. They can be identified and categorised by its noise source.

TRNG's also can get its random source based physical phenomenons such as thermal noise, quantum changes, jitter from oscillators, some say this can be predicted and in

certain way it is, having the right amount of computer power for the calculation of all probabilities from all the external variables, it is possible. Hence, nothing is random in nature, only our prediction algorithms and technology are not enough.

Other form of categorisation on RNG's can be made by the deterministic type source, algorithmically, software created generators, are considered deterministic RNG's, the result involves no randomness associated if all conditions are known, hence if some conditions become a secret, then the sequence is apparently random.

Non-deterministic noise sources are what we are looking for, but this can also be further classified in two more categories, respectively, physical and non physical phenomenons.

Physical non deterministic sources are derived from effects such as electronic noise from resistive elements, noise from diodes, breakdown from capacitors or even radioactive decay on atomic elements. The non-physical non-deterministic can be associated with non physical processes in the computer environment, like RAM utilisation, hard disk seek time, defragmentation, full system time, normally this is associated with the LINUX random tool which accumulate entropy from various processes and then outputs it as a random sequence. Although this is classified as non deterministic source, it is hard to create TRNG using this methods.

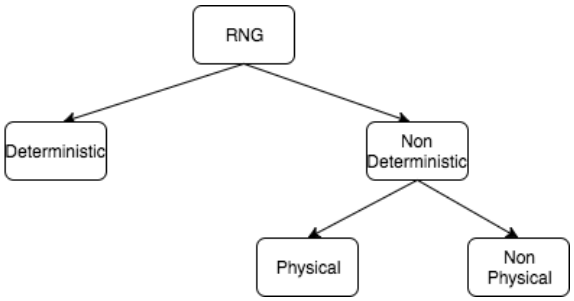



Figure 2.2: RNG Noise Classification

When searching about random numbers, one word that constantly shows up is entropy. Entropy is a fundamental property on a random generation process, entropy is the measure of the uncertainty associated to a random variable [41]. Usually this property present better performance results when associated with hardware generators.

## 2.2 Basic RNG Architecture

Architecture of generators can be varied but a simple chain is mainly used. It is not defined this way, but it was proved to be efficient and thats why it still stands. This chain usually comprises 3 steps:

- a noise source,
- a post processing block,

- an extraction block.

The first one is composed by a set of components which creates high entropy in the system. This signal is usually analog and then is then passed to the processing unit at a sample rate, the Digitised Analog Signal (DAS) is processed and extracted to the system which required the random number.

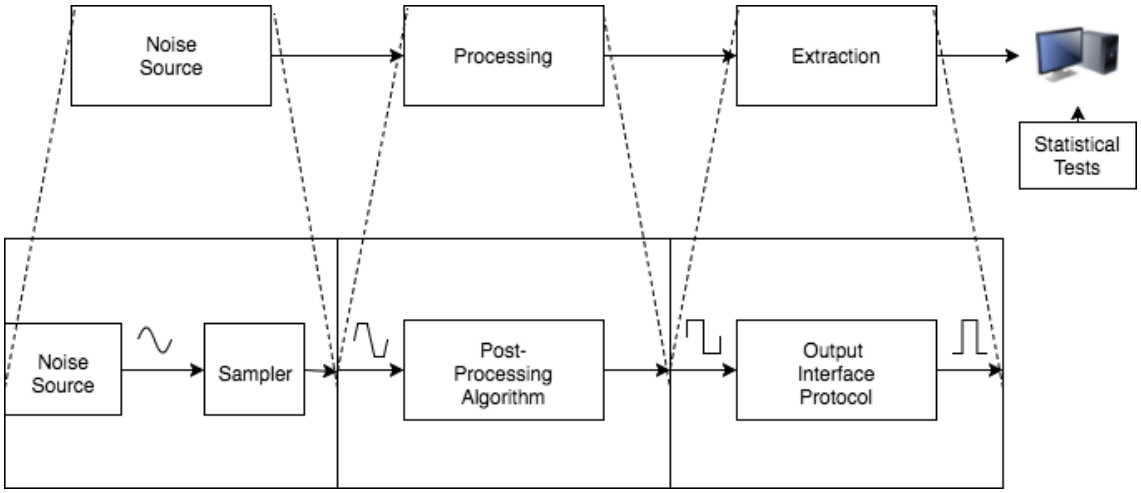



Figure 2.3: RNG general Architecture

Figure 2.3 represents the general architecture of a typical RNG building block. The noise source can be created using sources described previously, the entropy is then sampled at a rate in order to obtain the DAS, this sampling rate should be enough to capture the entropy of the system the most efficient as possible, i.e. if the systems evolves really slow, a high sampling rate would oversample the noise source and as so the post processing unit would have to process large sets of sequences only to produce a bit. The sampling rate must be either the ideal for the system or limited by technology.

### 2.2.1 Noise Sources

As described, there are many noise sources available, each one with its advantages and disadvantages. And as said in section 2.1, one the most important properties on noise sources is entropy [32].

Based on that, here are some examples of noise sources available:

- Electronic noise;
- Shot Noise;
- Metastability;
- Jitter;
- Quantum;

Electronic noise can also be considered thermal noise or johnson noise, since high/low temperatures makes the electrons move at higher/slower speed in a conductive material then this change makes subtle variations in the voltage of different components, quantum noise makes the most use of radioactive decay to its entropy source. Zenner and avalanche diodes can produce breakdown voltages, this creates random DC current fluctuations which is mostly known for shot noise. Lastly Metastability and jitter noise are considered the ideal digital noise on TRNG's, the metastability derives from the gates setup and hold times while jitter noise appears when rising and fall times are irregular, although jitter can be deterministic and non deterministic, in the random process only non deterministic jitter noise is considered.

### 2.2.2 Sampling

While some systems may require complex blocks to sample data like ADC's and equivalent. In TRNG's usually only a D-type Flip Flop is used to capture the random entropy output. The clock on the Flip Flop is it's sampling rate, just as described in section 2.2, this sampling rate should not be too fast neither too slow. The output of this Flip Flop is now a DAS that can be processed if required.

### 2.2.3 Post-processing

Post-processing sometimes is required because the output is not random due to some bias effects. This post processing aims to unbias the output from the sampler, like the oversampling example, if the output is oversampling entropy then the result is bias to '1' or bias to '0'. Post-processing is used to remove that effect from the sequences.

In order to accomplish this, many correctors can be used such as Von Neumann Corrector, XOR corrector, Resilient functions, etc. All of these correctors attempt to remove bias or deterministic bits in the random sequence.

### 2.2.4 Output interface

After all the previous processes done, the output can be observed in a oscilloscope and we would be amazed by the changing '1's to '0's and vice versa, but this does not give us output required or the random numbers to evaluate the string. We can output the oscilloscope captured signal to the computer and analyse that with MATLAB, join groups of 8 bits into 1 byte of data, post ward generate a HEX from that resultant number and apply that to generate our random numbers, but the aim of the project is not to create something to be difficultly analysed. The devices in which this system can be implemented do not have the same capabilities of an oscilloscope neither something near of that is required to the manufacturer. Hence, an extraction method needs to be implemented to provide a user-friendly tool.

## 2.3 Previous RNG's works

### 2.3.1 Noise Amplification RNG

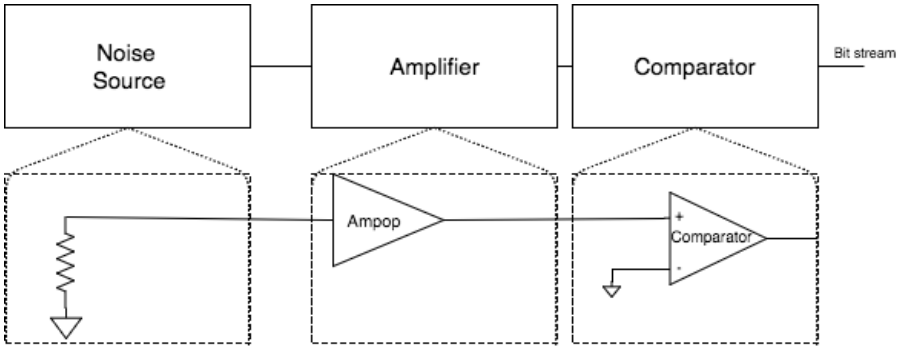



Figure 2.4: Noise Amplification RNG Circuit Schematic

Figure 2.4 presents a circuit schematic of a RNG block based on noise amplification techniques. The noise source is usually composed by a passive resistor, followed by an amplifier and sampled by a comparator.

This is simplest way to create a random analog generator. A bit sequence is generated based on a thermal noise source, the resistor changes its properties due to variations in the environment creating small voltage fluctuations, even small changes in the environment will create small voltage variations, it is mostly known as thermal noise.

This noise is really weak, therefore, an amplifier is needed to raise the noise to a voltage we can work on. The result of the amplification in then compared with a threshold voltage to assess if the output should be high or low logic level. Although this type of RNG appears to have no disadvantages because of its simplicity but in fact with a more careful look, this approach takes a lot of space in a IC and consumes a lot of power.

One of the major disadvantage of this design is due to the fact that these circuits when implemented need to be protected from the changes that occur from external sources like a power supply and substrate signals. Not having this protection, attacking this generator becomes easier since thermal changes can conduct to predictability in the bitstream. Afterwards it still needs a post processing unit that it's not represented in the figure. Since this design is presented as very simple, that shielding part comes as a major disadvantage because of the size it needs in order to be effective. Although this type of design it's still used in cryptographic systems, it's losing way to new technologies were some things need to be implemented digitally [23] [10].

### 2.3.2 Three Stage Noise Amplification RNG

Figure 2.5 shows a block a diagram of the circuit proposed on [23].

It comprises the previous block as noise source but it is composed by other additional two blocks, the oscillator sampling and discrete-time chaos. The resistor and the amplifier comprise the direct amplification of the noise, the adder, the Sample and Hold(S/H),

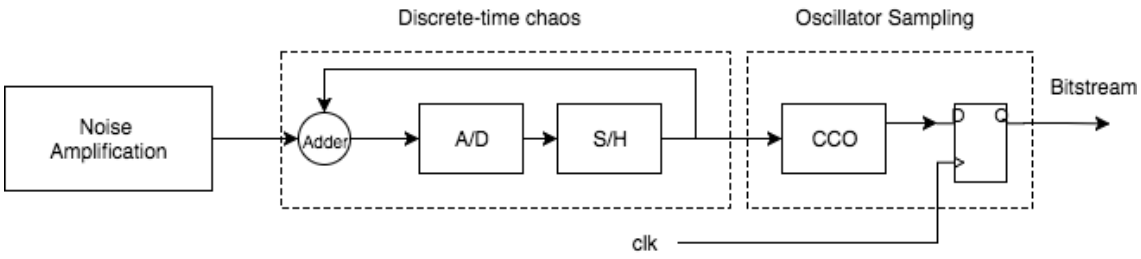



Figure 2.5: Noise Amplification RNG proposed by [23]

the Analog-to-digital converter (A/D) and limiter comprise the discrete-time chaos and finally the current controlled oscillator(CCO) and the sampler comprise the oscillator sampling. This RNG works by joining all of the strengths from each of the RNG techniques. The low power noise is amplified and the discrete-time chaos block measures the phase difference of this noise to create a current to drive the CCO, finally this CCO is then sampled the result can then be post processed as most of the RNGs. The solution proposed was the one on Figure 2.5. The A/D used was a single-stage (bitcell)[24] due to its simplicity and lack of a true high impedance node, a current mode S/H [29] and the CCO is a five-stage current-starved inverter architecture [40].

Circuit performance presents itself with good results but with a bigger size than most of the rest of the RNGs.

Table 2.1: Three Stage RNG Performance Table

| | |
|---|---|
| Maximum speed for 'near-random' behaviour | 4.7MHz |
| Typical I/O ratio | 0.4996 |
| Performance with substrate noise interference | unaffected |
| Maximum power supply variation | 1.8Vpp |
| Area | $1.5mm^2$ |
| Power dissipation at 1 MHz | 3.9mW |
| Supply voltage | 3 V |

One the disadvantages presented on the table 2.1 stand for the frequency at the behaviour is described as 'near-random', which is at 4.7 MHz and since the power dissipation and the presented working range are always described at 1 MHz, this presents a lack of information. Furthermore the chip die area of $1.5mm^2$ is considered as high area when compared to other RNG implementations.

### 2.3.3 SiN MOSFET Ph-RNG

Since the first problems of the RNGs are area on the circuit, the low noise signal and the fact that these generators will require thermal noise. This approach was designed to deliver a small IC and a high generation rate, which was one of the problems too, there always been some kind of trade off between IC size and generation rate.

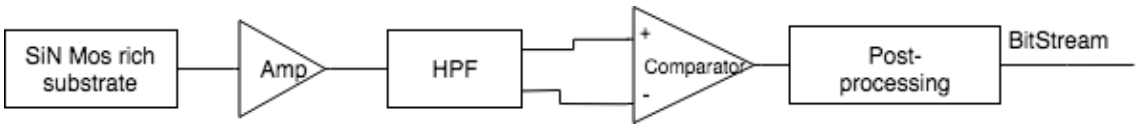



Figure 2.6: SiN MOSFET RNG proposed by [20]

Figure 2.6 presents the circuit schematic of a SiN noise based RNG that comprises a electron rich SiN substrate in order to create large magnitudes of noise, since this rich base creates a lot of noise, a sophisticated amplifier is no longer needed. A simple amplifier has been used followed by a high pass filter (HPF) designed to cut off low frequency signals originated by other sources than the SiN source. Just like the others RNG's a comparator and a post processing unit was used to deliver the output bitstream.

This type of implementation can be used in smart cards, due to its size. Using this type of RNG can complement the smart card by giving it a really secure system since they already have a CPU, random logic, RAM, etc.. The results are good and it shows a really small circuit footprint, and much faster output results than previous works [20].

Conventional analog thermal noise generators , usually comprise a weak source followed then by an amplifier, filter, comparator and a post-processing block. This was and still is the standard of most RNG, this approach, since it already uses a high power noise, the size can be reduced in the amplifier and filter block. This provided a standard RNG in a much smaller area.

The performance of the block presented good results as well, variating from a large range of temperatures, the generation rate stays constant. Although this is presented as a small circuit, it does not have the lowest power consumption.

One of the disadvantages of this approach is the method for creating the SiN substrate, which requires specific methods in IC manufacturing. Other circuits proposed are bigger however even lower power than this one [2]. The design uses a SiN electron rich base which is temperature independent to generate a high noise signal.

### 2.3.4 Linear Feedback Shift Register RNG

A linear-feedback-shift-register (LFSR) is a basically a simple shift register in which the input bit is a linear function of its previous state. With a predisposed set of output bits all in an XOR gate, can develop the linear shift. The initial value of the LFSR is called the seed, and because the operation of the register is deterministic, the sequence of values produced by the register is completely determined by its current (or previous) state.

This is what is called a pseudo RNG because it only can draw a predisposed set of numbers.

Figure 2.7 shows a $n$ bit LFSR which comprises $n$ D-type flip-flops(FF) and an XOR gate.

This simple circuit works as sequence generator of $2n-1$ different non-zero bit patterns (being $n$ the number of flip-flops used). At each clock, the signal will be shifted to the

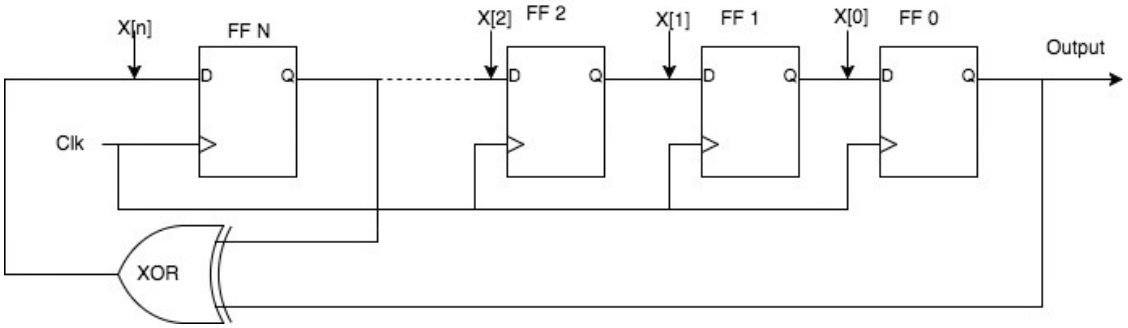



Figure 2.7: LFSR RNG $n$ bit

right and the output bit will be XORed with the bit that was previous in the $n-th$ FF output. This random sequence can be obtained by give a predisposed set of bits to the flip-flops, $X[i]$ inputs as a seed.

This implementation is as fast as technology can get the clock signal be generated. Some changes have been done to get more randomness out of this type of circuit, but most of them tried to get more longer cycles, meaning it would get more available patterns, still a Pseudo RNG but a lot harder to verify.

The main disadvantage on this type of implementation is its security. Since it is a PRNG, all of the results appear random but after a long cycle some patterns start to repeat. According to [27] this type of implementation produces sequences which are usable in a variety of applications like circuit testing, system simulation and Monte-Carlo method. All of them not needed for security but for analysis methods.

In Figure 2.7 we represent a simple LFSR RNG design, but such as Ring Oscillator RNG's can be designed with multiple techniques, also LFSR's present a lot of design techniques to obtain different results [5]. The more complex designs use mixed XOR techniques, where more XOR gates are added. Those modifications also change pattern cycles but speed remains constant.

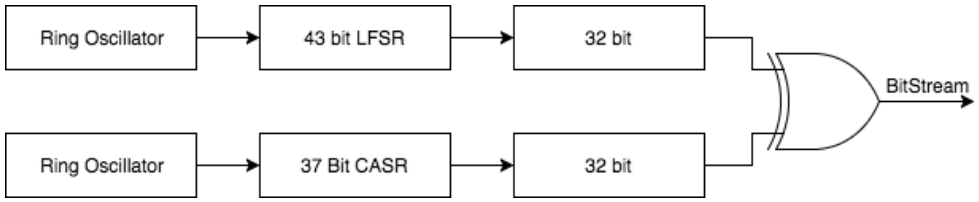



Figure 2.8: Tkacik LFSR CASR TRNG design

One of the TRNG implementations using LFSR's is the Tkacik design. In this design the XOR sample 32 bits of a 43 bits LFSR and 32 bits of a 37 bit Cellular Automata Shift Register(CASR) [3]. These two blocks, CASR and the LFSR, were then sampled by two independent ring oscillators as a clock signal [34]. Although this design presents good qualities on the other hand as the designer shows, although it passed some random tests, the entropy source is limited to these two oscillators and similarly to the previous design

it uses a LFSR, the attacker could decrypt the model in order to predict its output. In [6] this design is criticised for its potencial weaknesses.

### 2.3.5 Ring oscillator RNG's

One of the most reliable TRNG designs used are ring oscillators.

> *"A Ring oscillator is a cascaded combination of delay stages, connected in a closed loop chain. The ring oscillators designed with a chain of delay stages have created great interest because of their numerous useful features"[18]*

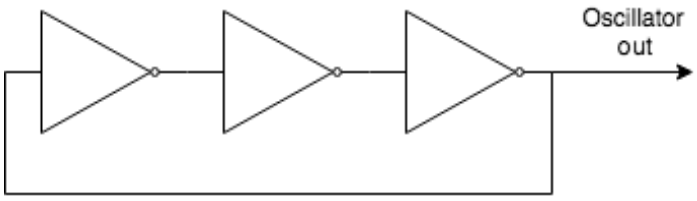



Figure 2.9: Ring oscillator RNG design

Figure 2.9 shows a simple ring oscillator with 3 inverter gates.

This is composed by a odd number of delay stages(inverters) and a loop feedback to start all over, this creates a free running oscillation between high and low logic level.

The ring oscillator RNG design was first proposed by [32] and comprised several simple ring oscillators XORed and sampled by a simple D-type Flip-Flop. This simplistic design presented very good results and several other designs were attempted using this as a basis.

#### 2.3.5.1 Ring oscillator TRNG design

The first design has been implemented in a FPGA and sampled at a frequency of 40 MHz. The ring oscillators had a fixed length of 13 inverters across 110 ROs. As post processing in this design resilient functions were developed. One of the major critics on this design are available in [6] and they rely on the statistical independence of the ROs on the original design. One characteristic of this design is the fact that the same RO but placed differently on the FPGA creates different entropy on the system.

One of the TRNG methods was proposed by [32] and it is represented by figure 2.10. By having *'m'* ring oscillators composed of *'n'* NOT gates the analog output signal of the circuit can be found at the output of the binary XOR-tree to be sampled by the D-type flip flop.

Parameters *'m'* and *'n'* used in this implementation can be determined for many different results. [32]

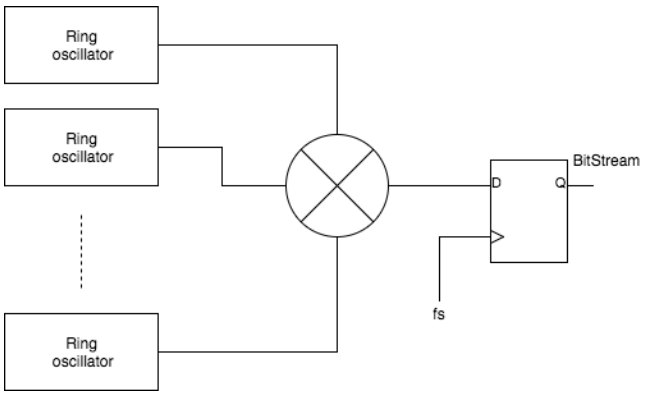



Figure 2.10: Ring oscillator TRNG design proposed by [32]

#### 2.3.5.2 Ring oscillator Golic Figaro design

This design is also known as Fibonacci ring oscillator (FIRO).

It has the advantage of the LFSR inputs do generate entropy but with inverters to create even more entropy. According to its developers, it creates more entropy than normal ring oscillator with the same length of cascaded delays [6].

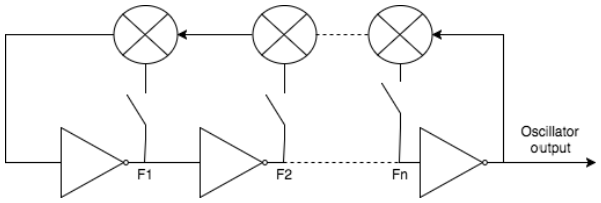



Figure 2.11: FIRO design

Like the LFSR, the $F_n$ values can be always changing in order to obtain higher entropies. This ring design work similarly like the LFSR but they operate asynchronously so, apparently the design have the properties to present random behaviour. The other hybrid ring that derived from this implementation is the Galois ring oscillator design (GARO)

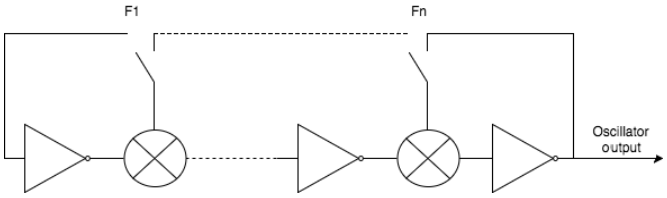



Figure 2.12: GARO design

The proposed from [6] is to create a regular design merging both designs. Results presented by the developers passed the proposed tests and seem to present good results. Although according to developer data, at each restart the behaviour does not appear to be random, randomness only appears after around, 40 ns.

#### 2.3.5.3 Metastable RO TRNG

This design based on [35] exploits one of the most important properties on the ring oscillators: the metastability.

Every circuit component has this physical phenomenon, when it is excited with a fixed threshold voltage, it enters in a metastable state, a dynamical system state which is basically composed of noise and uncertainty around the desired value that is expected for the component to have. This uncertainty is again entropy, so this RO can create entropy.

Metastability in this ring oscillator design occurs when a inverter gate is connected to its ouput and the voltage stochastically fluctuates around the value of the desired state.

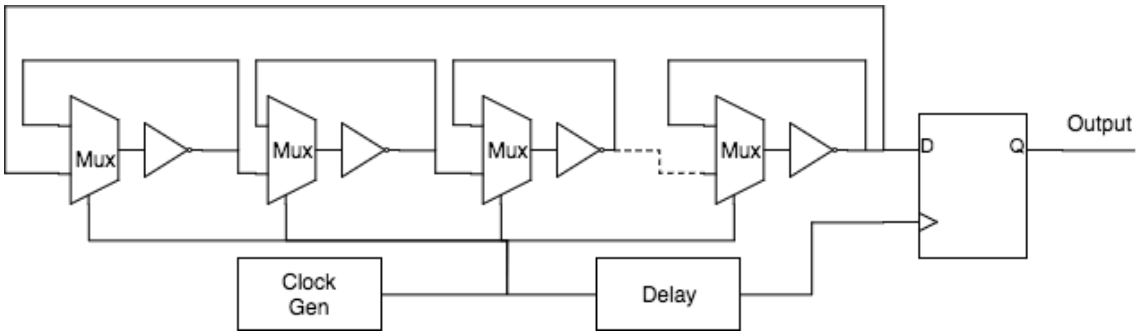



Figure 2.13: Metastability RO

Figure 2.13 presents the design of the ring, which comprises a Multiplexer and a inverter gate per ring stage to create the metastability effect, also a clock generator and a delay stage to capture the output of the RO. The testing was done using Cadence as well as an FPGA implementation. Tests to this RO topology proved that it performed well and that using a 65 nm technology, the total size of the ring was $1 \mu m^2$.

#### 2.3.5.4 Yang MultiMode TRNG

While most of the ROs produce high frequencies and create high jitter noise, there is one RO where this can be done with no additional inverters.

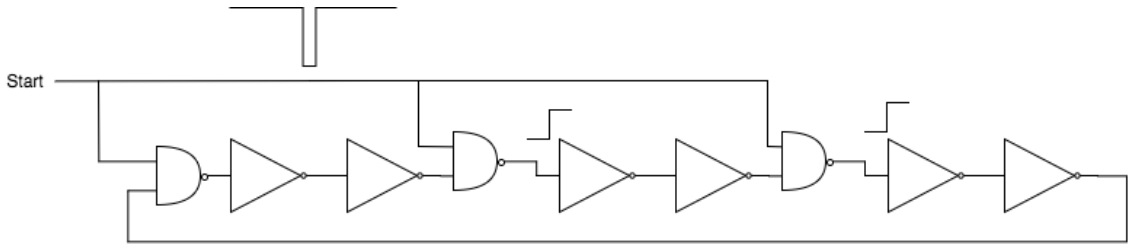



Figure 2.14: 3-edge RO proposed by [42]

> *"A conventional RO injects 1 edge that propagates through the ring to form pulses. The proposed 3-edge RO has 3 input nodes that inject 3 edges into the ring simultaneously". [42]*

Figure 2.14 presents a ring proposed by [42] which is composed by inverters and NAND gates.

This ring works by injecting a signal between the RO, this way, if a certain ring oscillator produce 10 MHz oscillation the same RO using this design can produce 30 MHz oscillation with the same amount of logic gates. Although this seems quite simple to implement, another issue is that, this RO may suffer from injection locking and collapse of the frequency injected, since not every single inverter gives the same delay, when a injection occurs, the frequency stays 3x higher than the nominal frequency until the first injection catches the second and then the third, and finally returning to the nominal frequency. The developer also has proved that longer rings tend longer to collapse while shorter rings collapse a lot faster.

This is how the RO proposed by [42] produces strong cumulative jitter which can also be seen as entropy. This RO design was not XORed with multiples ones but a different approach was made. The TRNG uses one ring and measures the number of cycles in the RO until the collapse of the frequency. Then the process restarts to obtain a new number. This design has some weaknesses in the measurements of this collapses but the results shown are good and passed the proposed tests.

Ring oscillator RNG implementations present good results for the area used, and they do not consume a lot of power.

### 2.3.6 Arbiter physical unclonable function

One way to develop RNG is by the arbiter method. This method generates two signals with different timings and delays travelling on a series of components with different characteristics, in the end where both arrive the 'arbiter' chooses who came first. Therefore the 'arbiter' is just a circuit that outputs the first signal to arrive. Since the way travelled by the signals is always different because of the different timings through the way, the arrive order is always different.

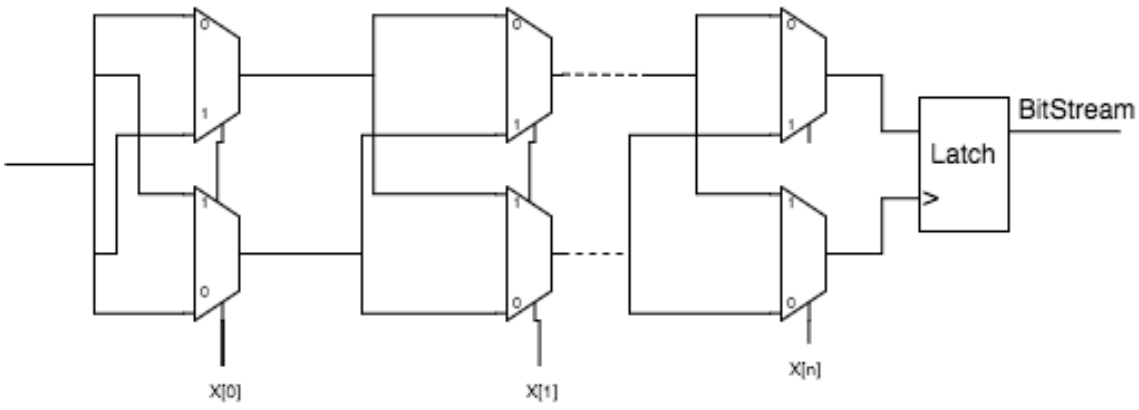



Figure 2.15: Delay implementation for Arbiter PUF's

Figure 2.15 presents a block diagram of a simple 'Arbiter' physical unclonable function (PUF), it complies the requirements, since it comprises a series of delays using multiplexers followed by an 'Arbiter' block at the end. The Multiplexers work as switching boxes where the input $X[i]$ controls the signal switching from top/bottom. This way the signals race with each other and finally the latch decides who got first. This type of circuit is the basic concept of the arbiter PUF, but can be 'hacked' if someone knows all the timings involved. Therefore, other type of techniques needed to be studied and pursued in order to get more secure systems. One of them is to obfuscate the signals output with an XOR.

In terms of performance, since it only relies on delay times, it is very robust against environmental variations.

> *"For realistic changes in temperature from 20° to 70° Celsius and regulated voltage changes of ±2% , the output noise is 4.8% and 3.7%, respectively. Even when increasing the temperature by 100°C and varying the voltage by 33%, the PUF output noise still remains below 9%."[31]*

This type of circuit can become a lot more complex in order to become more secure, higher output rates or even more randomness.

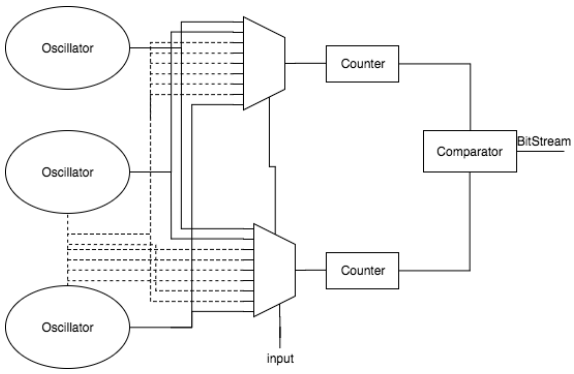



Figure 2.16: Arbiter PUF with Oscillators [31]

In section 2.3.5 we reviewed RO, they can also be implemented alongside with an 'Arbiter' PUF. Figure 2.16 shows a ring oscillator PUF implementation. It uses some concepts based on ROs and other based on the arbiter delay system. It is composed by several ROs in parallel to generate a signal and all of them joined together using two Multiplexers. Since all of the separate ROs oscillate at a different frequency, this creates variations on the output of each delay ring, in the previous arbiter implementation, every time a signal arrive earlier or later, the arbiter outputted a '1' or a '0' by the developer order of implementation respectively. In this case, every time a signal a arrive earlier than the other, a counter increments. When a separate clock ticks, a comparator compares the results of each counter. Then, it resets the counters and outputs a '1' or a '0' depending on the counter results. Although this is more secure due to the too many variables involved,

it is much slower than the previous arbiter implementation. The slower output rate is the only disadvantage when compared with the 'arbiter' PUF, since it can also give more 'random' sequences.

Since this design is completely digital, it can be implemented using an FPGA which will give some sort of freedom when designing the circuit. This presents another major advantage over the other designs previously referred.

CHAPTER 3

# TRNG DESIGN

After evaluating the prior work done and the state-of-the-art, it has been decided that the best implementation of the noise source should be based on ring oscillators.

All the experiments and designs are described in this chapter.

## 3.1 Hardware

Our design has been made using Xilinx ISE version 14.7 (nt64) and was mainly developed in VHDL. Xilinx platform provides an excellent environment to develop code and digital circuits for FPGA. The platform was created to code, syntethise, translate, map, place and route and finally implement the design on the desired FPGA. Basically, there is no need to interact with other platforms to implement designs. No simulations were made using the simulator incorporated in the platform due to timing restrictions and logic impossibilities.

### 3.1.1 Spartan 3 FPGA board UG130

Alongside with the Xilinx software, the project has been implemented in a Spartan 3 starter Kit UG130 FPGA with a X3S200FT256 device. Spartan 3 Family boards are well known for its precision, this specific board is known to be the best starter kit in FPGA systems development.

Many cloud based servers do have FPGA available as hardware security modules as well as some IoT devices which possess small FPGA to implement hardware functions. Therefore, they are the perfect way to prototype our RNG.

Some key features presented on this board are:

- 4,320 logic cell equivalents

- Twelve 18K-bit block RAMs (216K bits)
- Four Digital Clock Managers (DCMs)
- Up to 173 user-defined I/O signals
- 2 Mbit Xilinx XCF02S Platform Flash, in-system programmable configuration PROM
- 3-bit, 8-color VGA display port
- 9-pin RS-232 Serial Port
- 50 MHz crystal oscillator clock source
- JTAG port [30]...

### 3.1.2 Oscilloscope

All the measurements and signal testing were carried out using a Rhode & Schwartz RTO1022 Digital Oscilloscope. Some key features that worth mentioning are the 2 GHz bandwidth, 10 GSamples/s for 2 channels and a digital record rate for up to 40 M samples. The analog signal was read using a RT-ZP10 passive probe.

## 3.2 Cryptography using FPGA's

For quite some time FPGA are used to create cryptographic systems [41], as it is referred in section 3.1.1, one of the reasons FPGA are still in use rely on the fact that many systems require flexibility. Some systems require to be restructured without the need to physically replace all integrated circuits (ICs). FPGA's provide that advantage to users. In Cryptography, all the work can be done using software, but although that approach worked in the past, in recent years, that approach needed to change.

Our project aims to develop a new TRNG design using an FPGA for prototyping and for proof-of-concept. The final project can later be ported to an application specific integrated circuit (ASIC) or simply just ported to a different FPGA family device and implemented in the target device as a firmware.

An FPGA consists of a series Configurable Logic Blocks (CLBs), Input/Output pads (I/O) and routing channels connecting all of them together. These CLB's have flip-flops and lookup tables (LUT's) to implement the systems designed by the user. The netlist is implemented by the lookup tables and flip flops, multipliers, blocks of RAM, several IP cores and in the more recent FPGA boards, we have processor cores too in order to have a full SOC. Basically, the FPGA is a platform that allows the user to create digital circuits in a flexible and straight forward implementation. The design is coded in VHDL or via SCH on the ISE and then the same ISE reads that code and optimises that design into the target FPGA.

Table 3.1: XOR Truth Table

| A | B | A XOR B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Table 3.2: XOR Multi input Truth Table

| Input | Output |
|---|---|
| Odd number of 1's | 1 |
| Even number of 1's | 0 |

## 3.3 Ring oscillator TRNG design

The starting point for this project was as a first step to implement a ring oscillator RNG based on an XOR TRNG design [32], while attempting to improve performance and reducing the FPGA usage.

As described in section 2.3.5, ring oscillators (ROs) are composed by a cascade of inverter gates in a odd number with the last inverter output looped back to the input of the first inverter, acting as a feedback net. This creates a continuous oscillating sequence between high and low logic value.

Figure 2.9 shows the simplest approach to implement a RO, composing 3 NOT gates (inverter gates) connected in a feedback loop. This simple design is very susceptible to jitter noise which stands as a valuable property to increase entropy as a noise source.

To implement this as an RNG, several rings are XORed together and sampled by a D-type flip-flop at a frequency $F_s$, which will capture the jitter existent in the rings [32][29]. Since XOR is a special function represented by table 3.1 and mostly represented by exclusive OR, or EXOR, this function outputs '0' when signals are equal, and '1' when different. When there is more than two inputs on the gate, the output is as described in table 3.2. This data can be found on Xilinx ISE Help.

According to [2] for a better result, it is advised that several rings of different lengths should be used instead of a many rings with the same length.

## 3.4 Ring design and properties

Xilinx optimisation tools do not allow to create ring oscillators, as Xilinx synthesis tool (XST) attempts to generate components, it assumes that an odd number of inverters linked in a chain, consequently, timings apart, is equal as having just one inverter. To avoid this optimisation it was found necessary to enter in the XST properties and mark the option to 'keep hierarchy' of the design, this tells the XST that redundant information needs to be kept. Another issue with ring oscillators on the design is that the design will not be

synthesised if there is no input to the ring, figure 3.1 presents a solution to that problem. The first inverter gate was replaced with a NAND gate, in which one of the inputs is the enable signal that triggers the ring oscillation process.

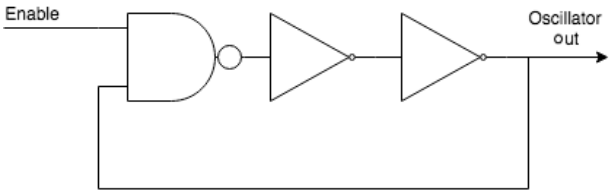



Figure 3.1: Ring oscillator with enable signal

Another problem that Xilinx presents to the developers, is due to the fact that several rings with the same length, XORed together and again, timings apart, they were all be the same if there were no jitter noise or different timings. The hierarchy on the synthethiser did not removed this issue, so the solution was to create a property on every net with the parameter KEEP set to TRUE, allowing it to not remove the oscillators.

Issues apart, several configurations and properties where explored and tested in order to prove some concepts.

### 3.4.1 Rings' length

The length of each RO was the first property tested when developing this system. Some conclusions were already been made on previous works regarding ROs' length [33].

According to [18], the resultant frequency of the oscillator depends on the propagation delay, $\tau_d$, per inverter stage, meaning that the length of the ring directly affects the output frequency. Therefore,

$$f_o = \frac{1}{2m\tau_d}$$

where the factor of delay, $\tau_d$, depends on many non linear variables and parasitics on the circuit. In order to test the properties on 'long' and 'short' ROs, two ring oscillators were designed and implemented. The two lengths used was 7 and 101 for short and long ROs, respectively.

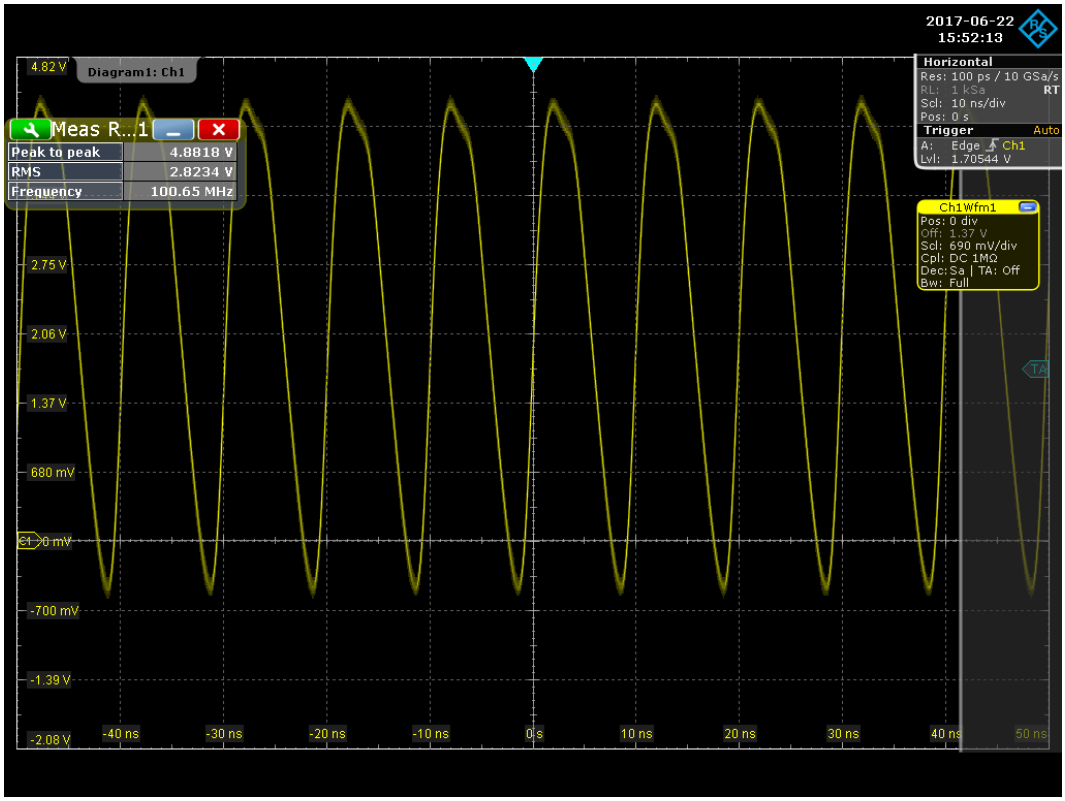



Figure 3.2: Ring oscillator 7 inverters

Figures 3.2 and 3.3 represent the output of the two ROs and the first thing that can be observed is the frequency associated with the oscillation. As expected, the longer RO presented a lower frequencies when compared to shorter one. As seen in previous figures, the RO with 7 inverter stages presented a frequency around 100MHz and the one with 101 inverters has only 7MHz. Applying the previous equation,

$$\tau_d \approx 0.7ns$$

Considering this value, is possible to estimate roughly the frequency of the oscillation, but just as described in this section, many factors can influence it.

In the next section will be discussed another property which can influence these values.

### 3.4.2 Placement of the rings

Since Xilinx offers many tools to manually place components as well as many other options to optimise the design placing on the FPGA Floor. We attempted to place the same RO on multiple sites in order to test the increase of uncertainty, reads entropy. Figure 3.2 shows the waveform resultant of a simple RO design with 7 inverters using the auto placement tool available on Xilinx for maximum delay.

With the auto placement set to maximum delay, a further test in order to accomplish a higher delay was put to proof.

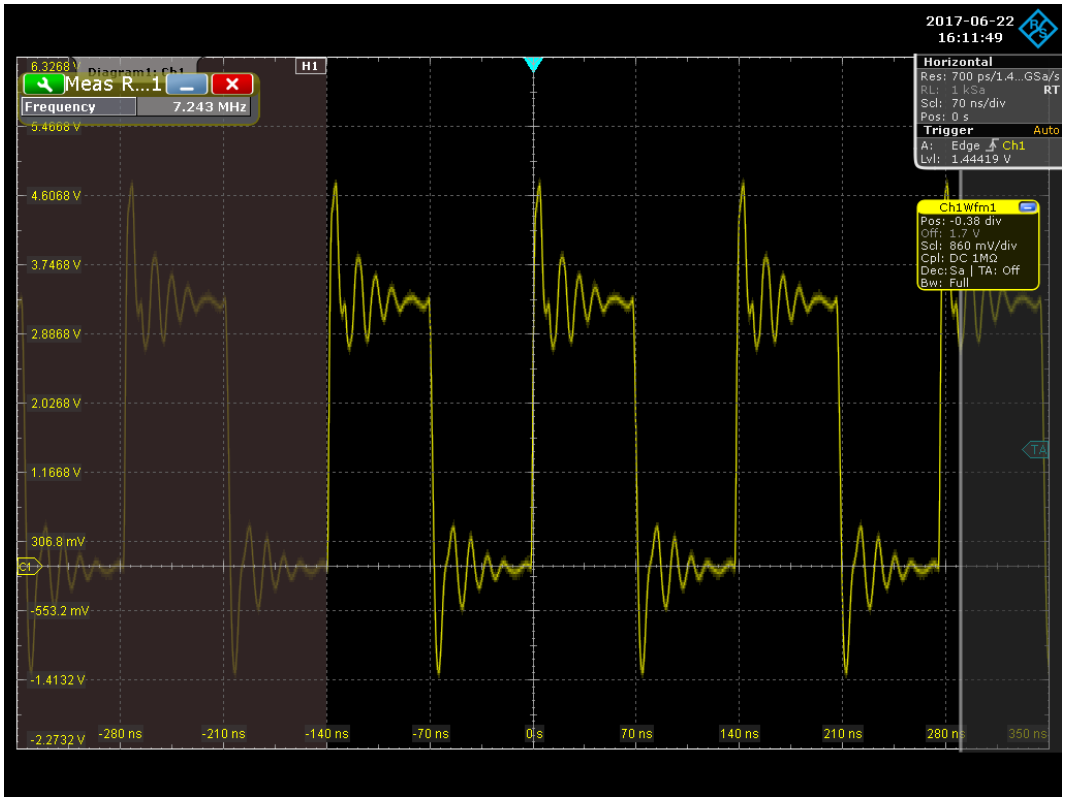



Figure 3.3: Ring oscillator 101 inverters

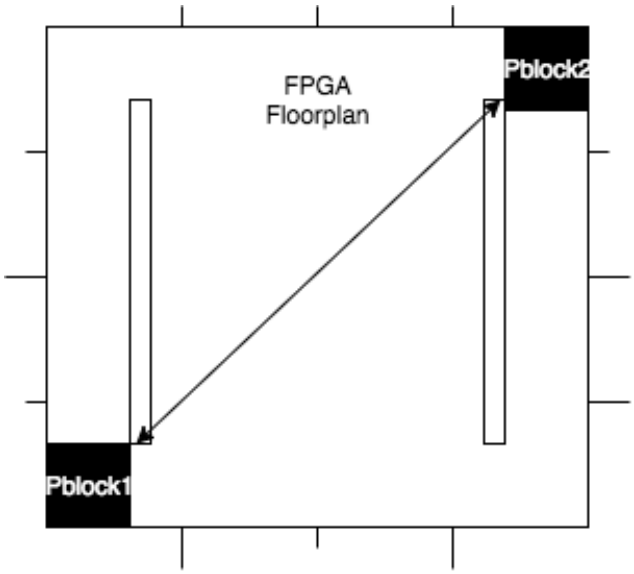



Figure 3.4: Manual placement test for higher delay

Figure 3.4 represents a test placement to achieve a higher delay on a RO. The first, third and fifth inverter gate where put in pblock1 while the rest was set on pblock2, this placement achieves a higher traveling path for the signals to run.

Figure 3.5 represents the output of the experiment. As can be seen this was proved

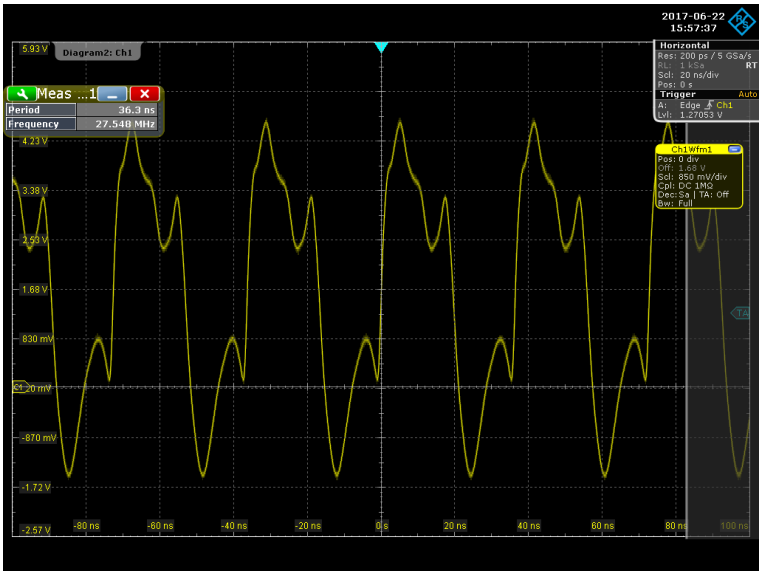



Figure 3.5: Ring oscillator with 7 inverters in a specific placement for maximum delay

to be correct, since the previous RO design with the same amount of stages presented an output of around 100 MHz and this one was reduced to about 27 MHz and a different waveform output. Applying the same equation, we obtained

$$\tau_d \approx 2,64ns$$

Hence, a specific placement will increase propagation delay as expected in a significant way.

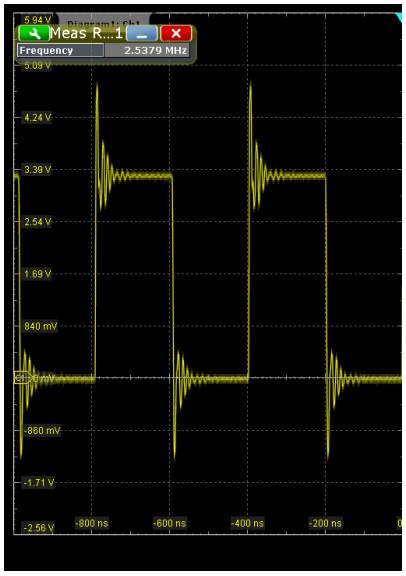



Figure 3.6: RO with 101 inverters manual placement

Again, the frequency on the manual placed design occurred but it is not so relevant like the smaller RO's. Figure 3.6 presents the waveform output of a long ring oscillator

with 101 inverter gates, again with the same placement as 3.4. And again the frequency decreased from 7 MHz to 2.5 MHz. The propagation delay,

$$\tau_d \approx 1.9ns$$

which stands for a significant change from the previous test. Another conclusion from these tests was cycle-to-cycle jitter which will be discussed in the next section, again as a property that can be used to further increase entropy.

### 3.4.3 Jitter and Phase noise analysis

Jitter uncertainty (noise) can be considered as an undesired property on most of our electronics due to its uncertainty on the signals time domain. It can be categorised as represented in Figure 3.7.

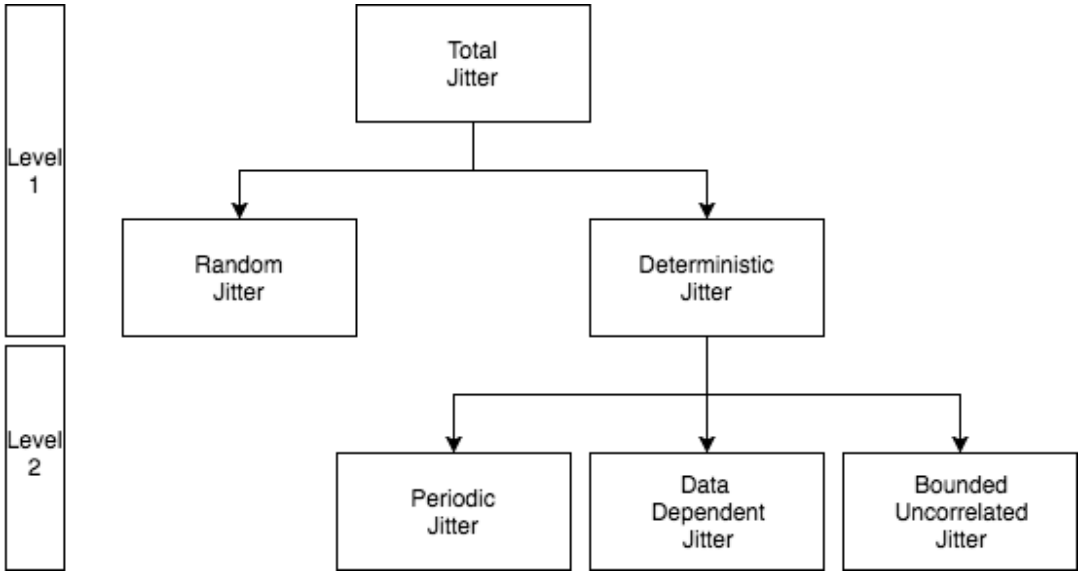



Figure 3.7: Jitter noise classification

It is considered as deviation from the expected time where the signal was supposed to happen, and since it is a deviation most electronics work based on perfection and small deviations affect performance a lot [22].

Random Jitter is the type of jitter we are looking for in our project, since it is considered non deterministic. Normally it is also called gaussian jitter because usually it follows a normal distribution but also stands as an unpredictable electronic timing noise [1].

Jitter on signals can be measured in three main ways:

- Period jitter,
- Cycle-to-cycle jitter,
- Time interval error jitter,

The first one attempts to measure the time deviation on each period, while the last one measures the deviation from the reference point of the ideal signal.

Cycle-to-cycle jitter (CCJ) is the measurement that applies to our calculations [18], since it will measure the differences in the period between adjacent cycles which are more meaningful for free running oscillators while others apply for example in PLL oscillators. Figure 3.8 illustrates this calculation and graphic representation of this measurement.

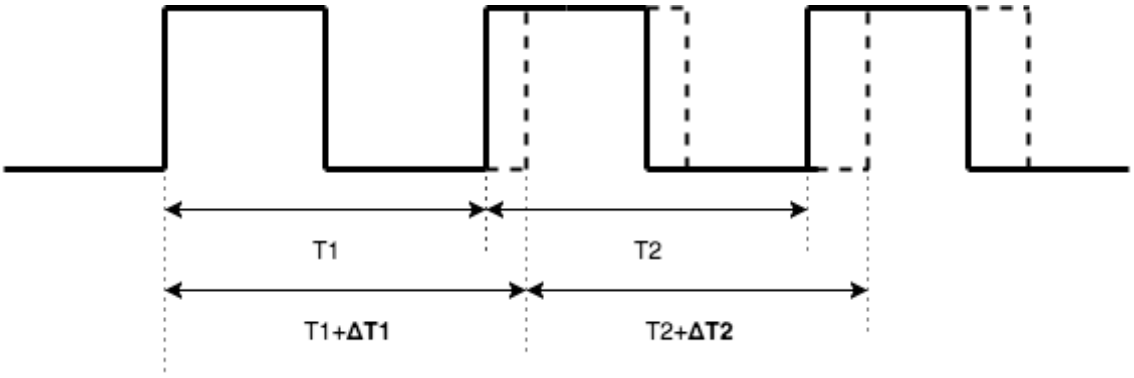



Figure 3.8: Cycle-to-cycle jitter

Cycle-to-cycle jitter is calculated by,

$$CCJ = \Delta T2 - \Delta T1$$

When this happens in real time for the infinite number of samples,

$$\Delta T_{cc} = \lim_{N\to\infty} \sqrt{\frac{1}{N}\sum_{n=1}^{N}(T_{n+1} - T_n)^2}$$

Phase noise also characterises the same noisy process, but this time using the power spectral density. It is estimated as [12] [15],

$$L(f_{off}) = \left[\frac{P_{sid}\left(f_o + f_{off}, 1Hz\right)}{P_{carrier}}\right]$$

with $P_{sid}$ as the single sideband power at the frequency offset $f_{off}$ from the carrier in the 1 Hz bandwidth, and $P_{carrier}$ being the power of the carrier as represented in figure 3.9. The units of this representation is Decibel below the carrier per hertz (dBc).

The jitter measurements on our project where carried out by jitter wizard available on the Rhode & Schwartz RTO1022 oscilloscope. Based on section 3.4.2 tests and using this tool we calculated how much CCJ existed on long and short ROs as well as how placement influenced these values.

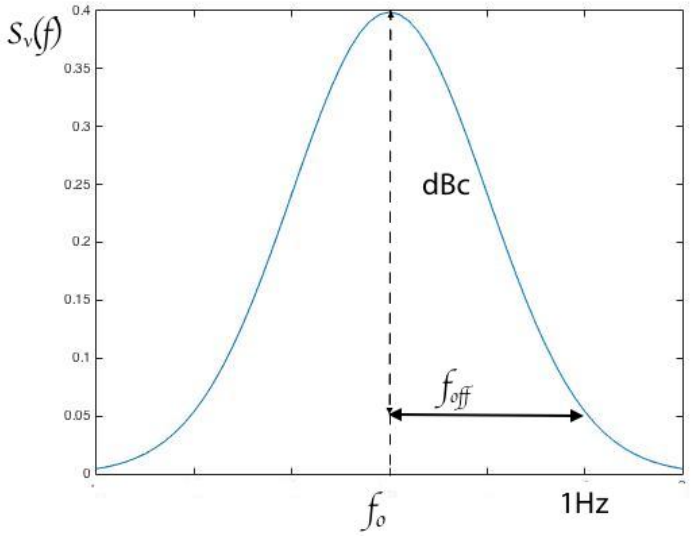



Figure 3.9: Phase noise per unit bandwidth

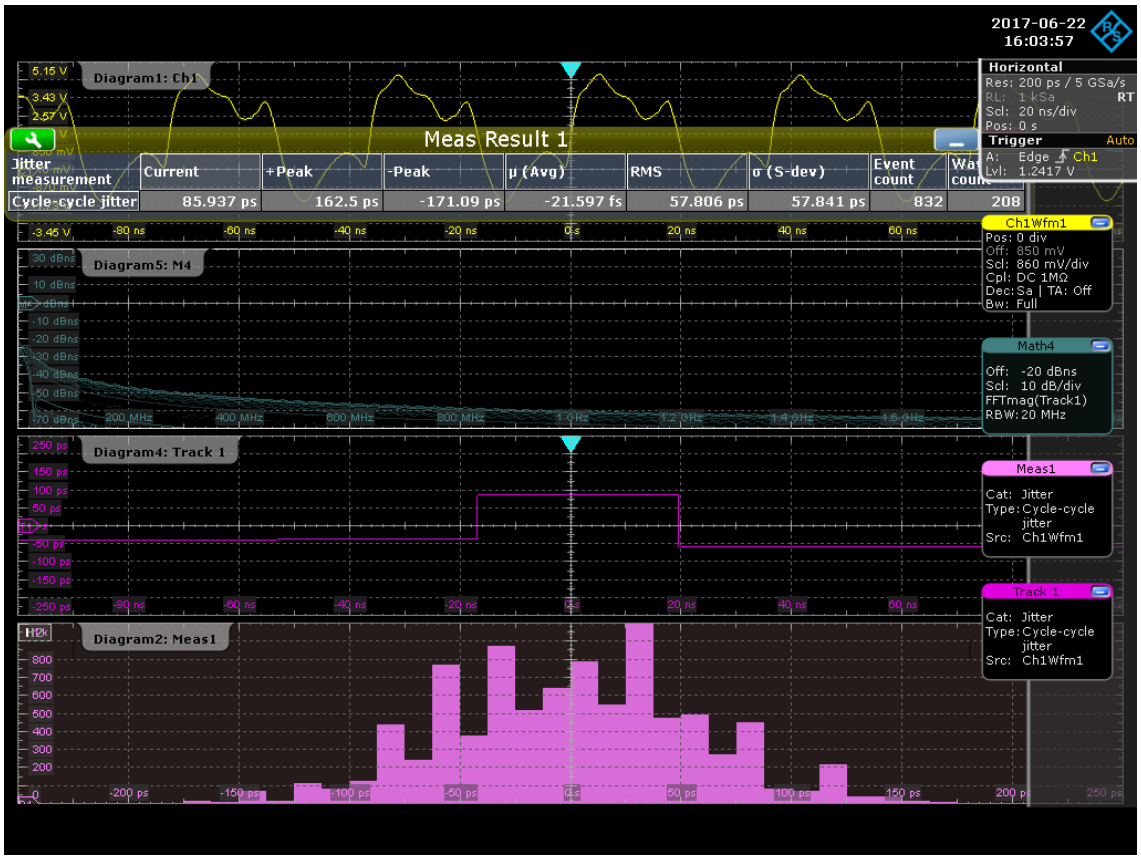



Figure 3.10: Jitter wizard on manual placed design with 7 delay stages

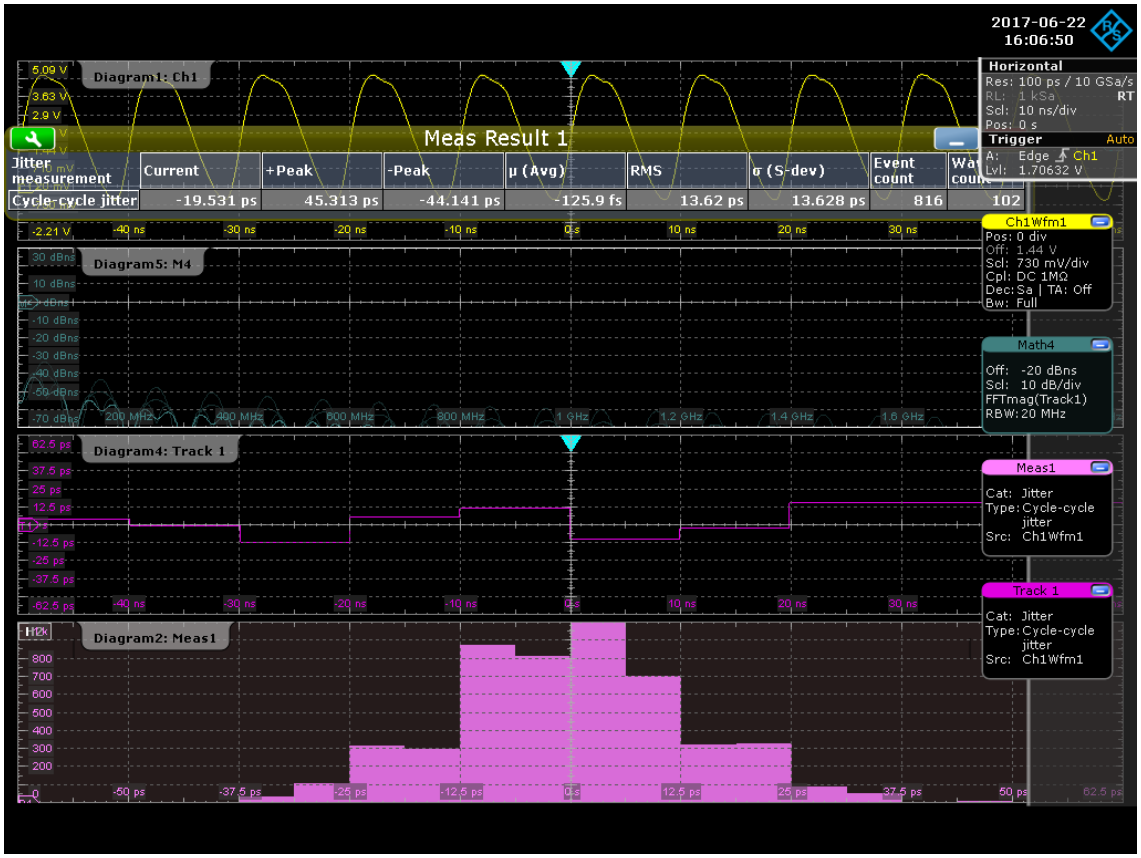



Figure 3.11: Jitter wizard on auto placed design with 7 delay stages

Figure 3.10 and 3.11 represents this comparison and as can be seen the standard deviation, $\sigma$, is about 13.6 *ps* in the auto placed design and about 60 *ps* in the manual placed. This means that the manual placed design accumulates more jitter as expected, due to the longer signal travelling path. The longer path will introduce more resistance on the signals which can be translated as an higher number of external variables affecting the signal which cannot be calculated, and again, increasing entropy.

The same test was carried out with the 101 cascaded inverter chain.

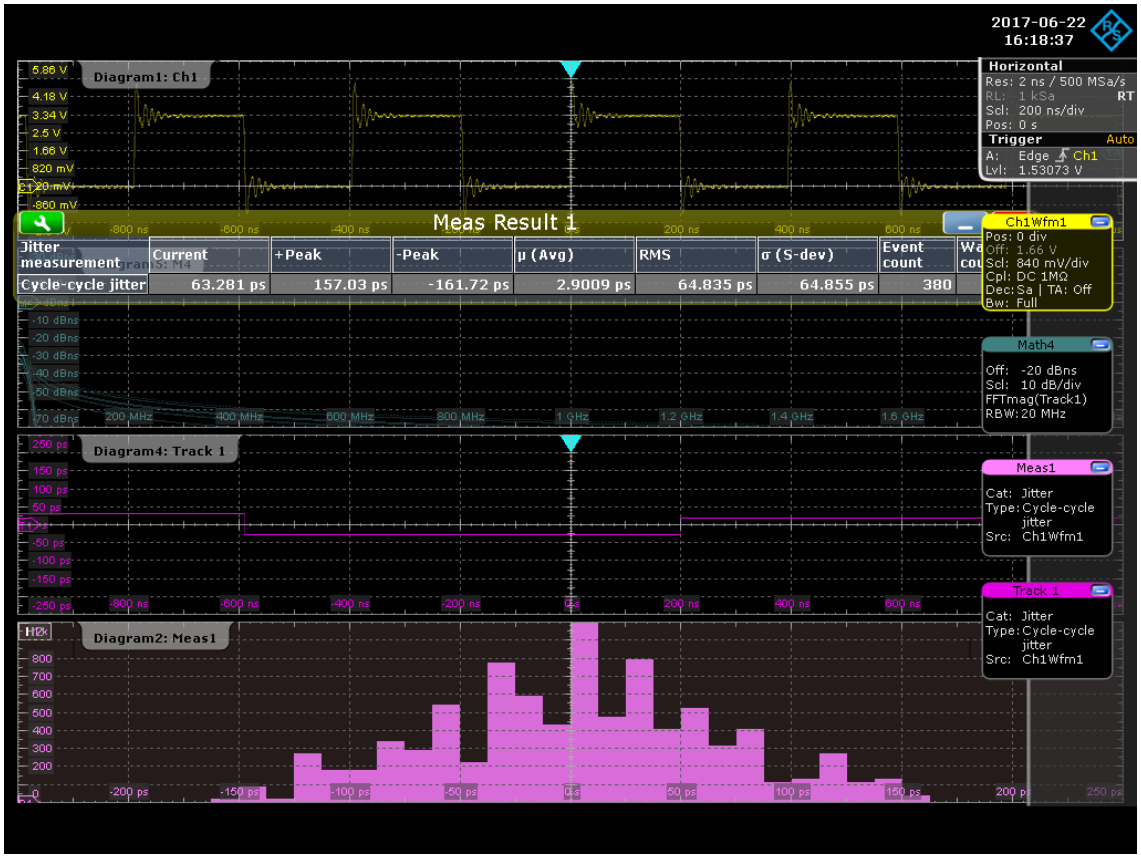



Figure 3.12: Jitter wizard on manual placed design with 101 delay stages

The placed design presented a standard deviation of around 65 *ps*, as the non placed presented around 53 *ps* as figure 3.12 and 3.13 represents. It would be expected that the placed design presented a higher CCJ but as this happens in the shorter RO, the same does not happen on the longer one. The conclusion on this test is that shorter ROs, accumulate more jitter when placed and longer rings present the same high jitter even when they are not manually placed, meaning that the manual placement to increase jitter does not worth the increase in die area/size, in contrast, shorter rings do worth it.

### 3.4.4 Ring oscillator topology

The RO topology to implement on the XORed method was the Multimode ring oscillator [42].

Just as presented in chapter 2, this design is based on a multi injection RO which can improve the jitter accumulation and oscillate at much as $m$ times the number of injections on the RO when compared to the same amount of gates. Although this design has been used to a create a TRNG with a different method, our project will attempt to create a XOR ring oscillator TRNG [32] with multimode ring oscillators [42]. This design for not being as simple as the first approach, provided a problem According to the developer, each RO tend to collapse to its nominal frequency, which is the frequency at which the

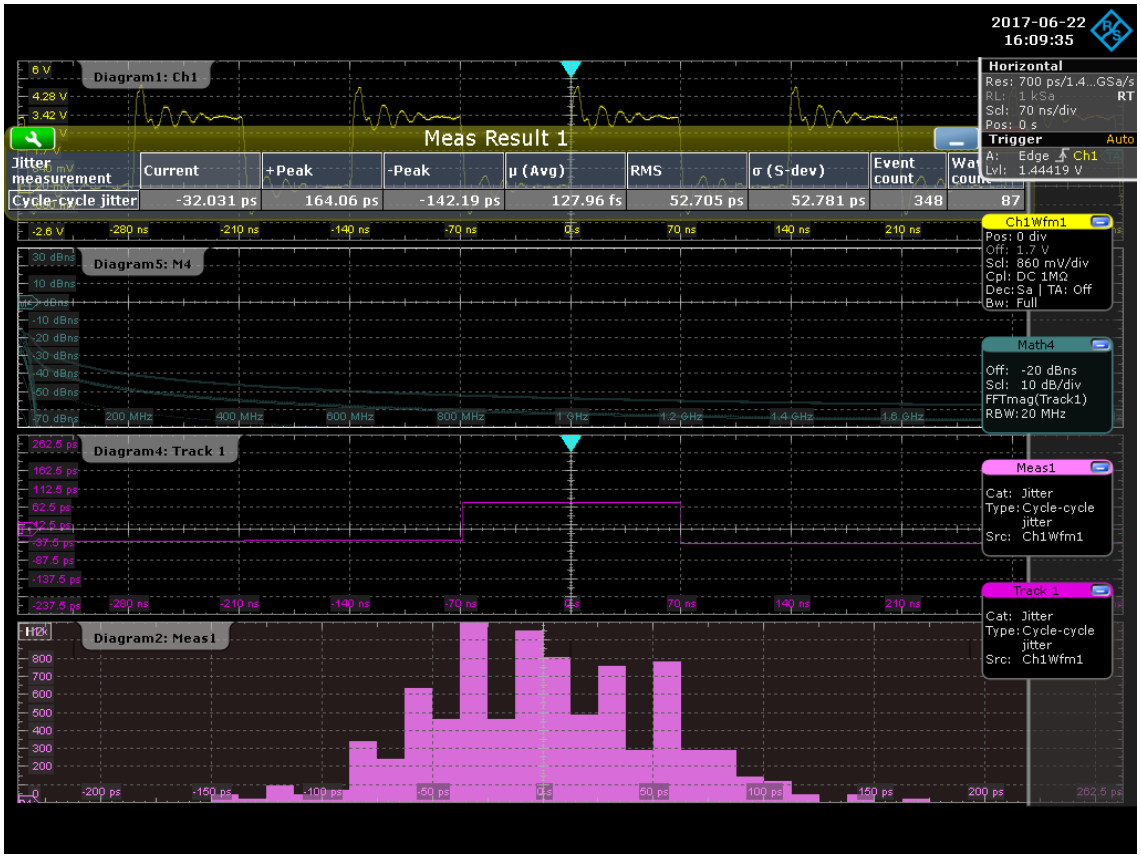



Figure 3.13: Jitter wizard on auto placed design with 101 delay stages

same amount of inverter gates on a ring would oscillate, this happens when the multi injections get caught on each other.

Figure 3.14 presents the output of two ring oscillators with 9 cascaded inverters, as can be seen the two outputs are oscillating at almost the same frequency, which means that the multimode implementation already as collapsed to its nominal frequency. A solution was then required to take advantage of the RO topology.

Another test has been carried out following the concerns on placement. Distance and interference between ROs seemed a good theory to test. Figure 3.15 represents the placement scheme used to test this theory.

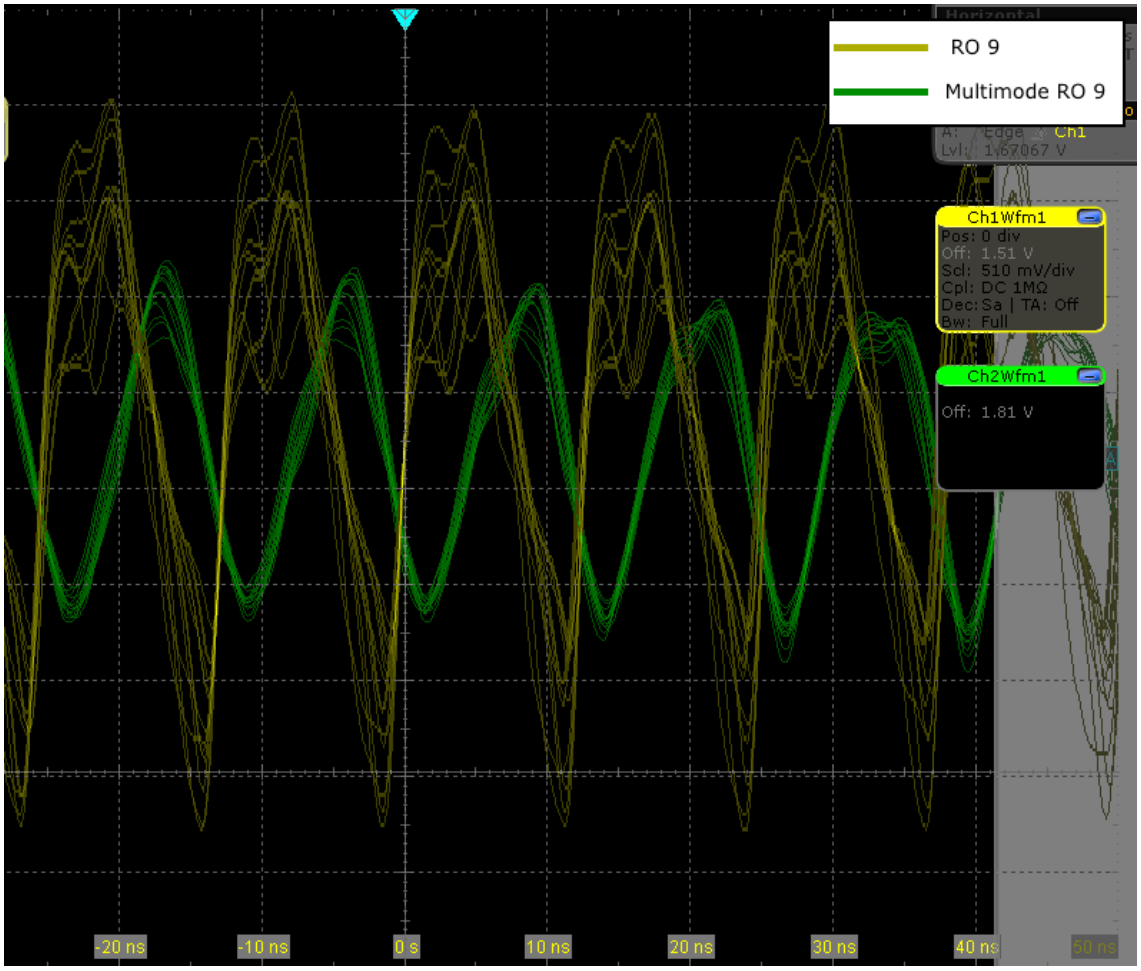



Figure 3.14: Ring oscillators with 9 inverter gates proposed by [42] and [32]

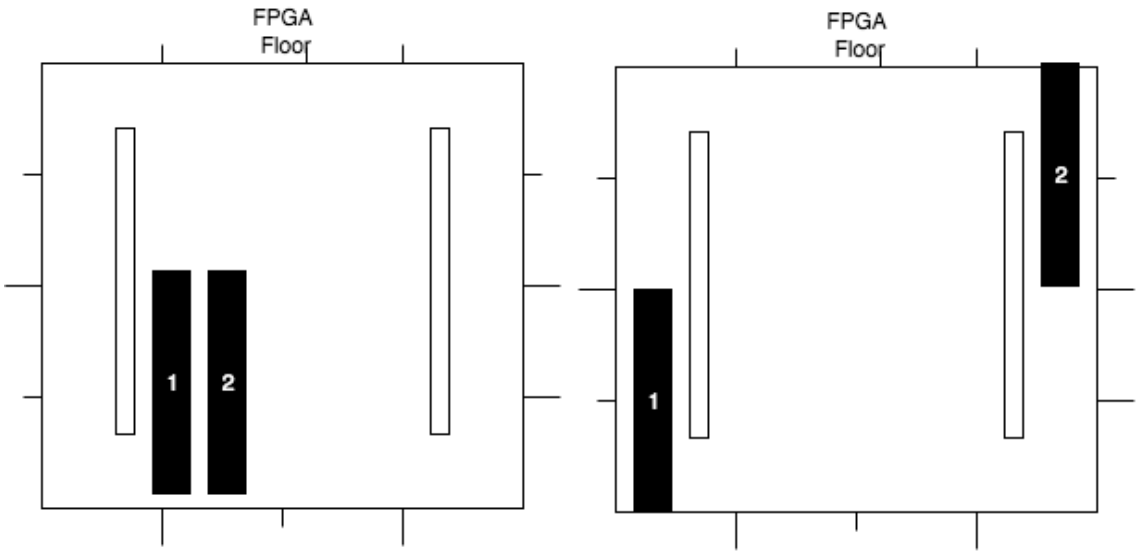



Figure 3.15: Ring placement on far apart and close interaction

a Test ROs close to each other

b Test ROs far apart

Figure 3.16: Waveform comparison between RO spacing

Figure 3.16 which represents the output waveforms associated with the two tests, the only influence present on the result was a specific harmonic which derived from the injection locking of the two RO since they are really close to each other, the same was concluded on jitter wizard [25].

One of the main conclusions from this test was that as long as there is a small prohibited gap between the rings, not much influence would be sensed.

## 3.5 Entropy source and sampler

The entropy source of the proposed RNG is composed by different length multimode ring oscillators all XORed together [32][42] as described earlier on this document. These rings presented a challenge when it comes to maintaining its properties. Hence, an auto reset block was designed.

The entropy source was designed with rings having lengths 9,13,15 and 21 with manual placement in order to increase the jitter on each RO.

Figure 3.17 represents the final entropy block diagram, which is comprises 9 RO for each ring' length, XORed and sampled by a D-type flip-flop at 50 MHz and having all the rings restarted with a pulse slow enough to prevent the rings from collapse. Tests shown that anything below 500 kHz worked, but ended up being in 25 kHz for best performance on our design.

Figure 3.18 represents the schematic block which comprises the design of the pulse generator. To reset the rings at the desired frequency, it converts the 50 MHz clock signal from the FPGA board into a slower clock at the desired pulse frequency, that clock is then put through a long chain of inverter gates and finally a NAND gate with the original signal to create a pulse at each clock signal.

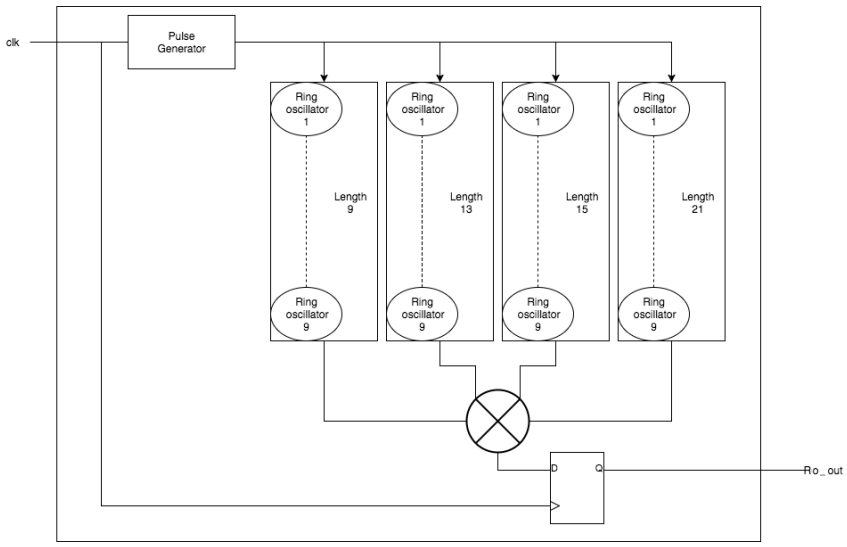



Figure 3.17: Final entropy source block and sampler

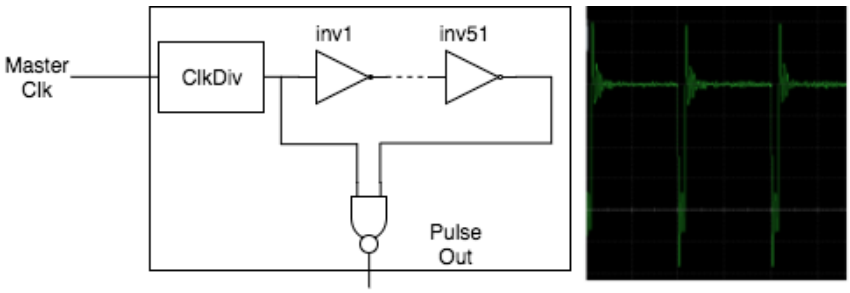



Figure 3.18: Pulse generator

As seen in sections 3.4.2 and 3.4.1, with manual placement and ring' length tests, the ROs used have a length of 9, 13, 15 and 21, these lengths do not represent nothing in particular.

According to previous conclusions, the shorter rings which oscillate at a higher frequency were placed to achieve a higher signal traveling path, and the two longer ROs were put to auto placing in a specific region of the FPGA floor. This placement has been achieved having the maximum possible jitter in mind.

Figure 3.19 represents the final placement scheme of the ROs on the FPGA floor. The RO with length 9 has been split in groups of 3 inverters, having the first group been placed on pblock1, the second on pblock3 and the third on pblock2. On ROs with length 13, have been also split in groups of 3 inverters having the first, the third group and the last inverter on pblock5 and the rest on pblock4. Finally the ROS with length 15 have been auto placed on pblock7 and ROs with length 21 on pblock6.

The reddish areas are defined a prohibited areas for the Synthesiser.

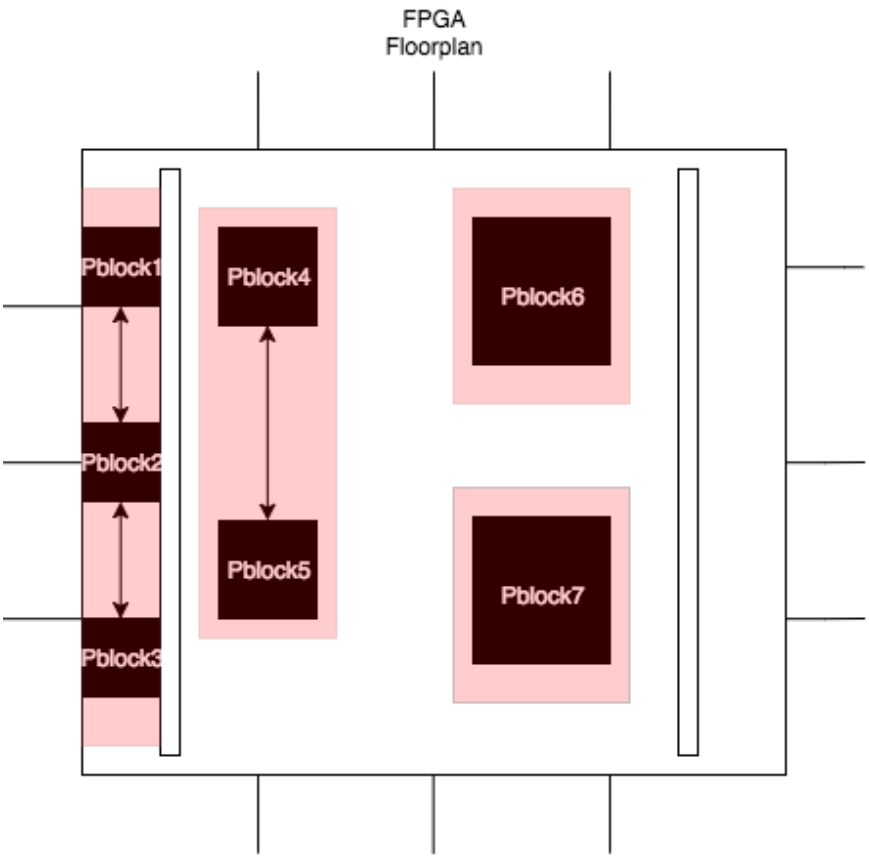



Figure 3.19: Final ROs placement

### 3.5.1 ROs data analysis

All the ROs with the same length have been XORed together and the ouput has been recorded using the oscilloscope and processed in Matlab. One of the basic signal analysis that can be made on the output of the rings is the Fast Fourier Transform(FFT). The FFT is a signal processing method that allows us to obtain the harmonic frequencies that compose a signal.

Figure 3.20 represents the signal on time domain as well as the corresponding FFT on the frequency domain. We can conclude that most of the signals oscillate at around 60 to 80 MHz and not having notably valued harmonics expect on RO with a length of 15 inverters presenting many harmonics. The same analysis has been carried out with the outputs of all the XORed ROs with lengths of 9 and 13.

Figure 3.21 represents the same test and as expected, it shows no evidence of frequency components on the signal output, which can be considered as a positive result, since the jitter presented in the rings is high enough to mask the frequency at which the ROs are oscillating.

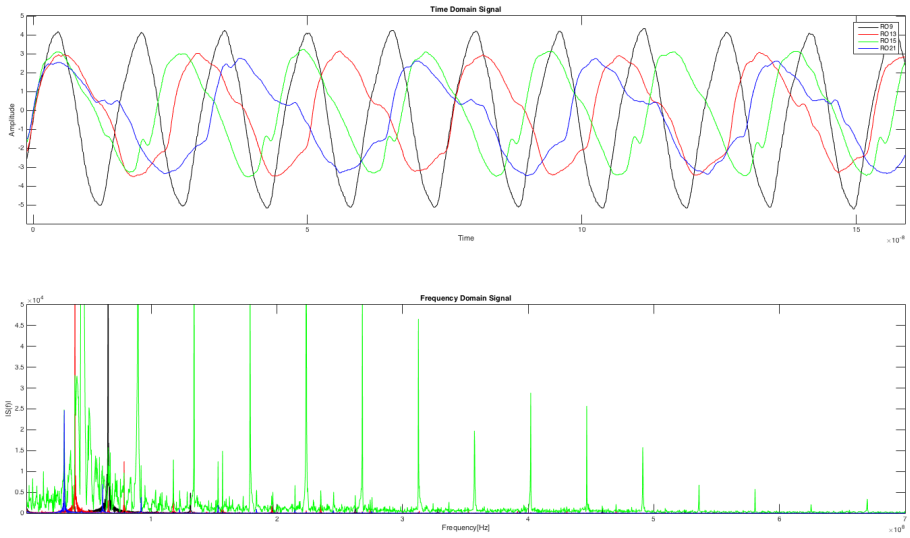



Figure 3.20: FFT of the RO output

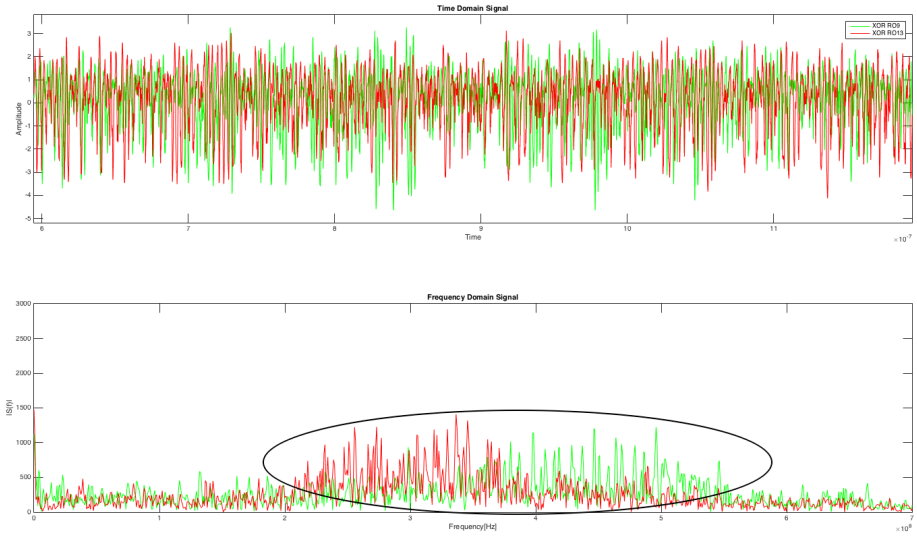



Figure 3.21: FFT of the XOR RO output

Also, in the same figure and highlighted, we can see a zone of the spectrum where there is a slight influence from the ROs, this presents no problem because all the outputs from all of the different ROs will be XORed and these frequencies will be masked as expected. Figure3.22 shows that no frequency components can be found on the output from all the ROs.

In addition to the FFT analysis, an Autocorrelation test was also carried out. Again, this analysis has been done using Matlab.

Autocorrelation and crosscorrelation are two properties that attempt to measure the

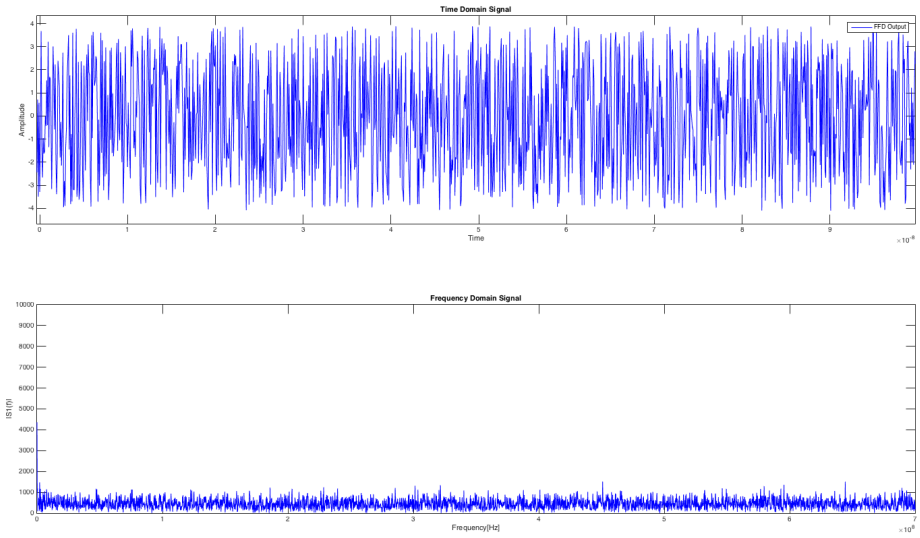



Figure 3.22: FFT of the Entropy source output

degree of similarity between two signals, whereas, the crosscorrelation measures the similarity between two signals and the autocorrelation is the same as a crosscorrelation of the signal with himself, which can be represented the following equation,

$$f \star f = \int_{-\infty}^{\infty} f(u+\tau) f(u) du$$

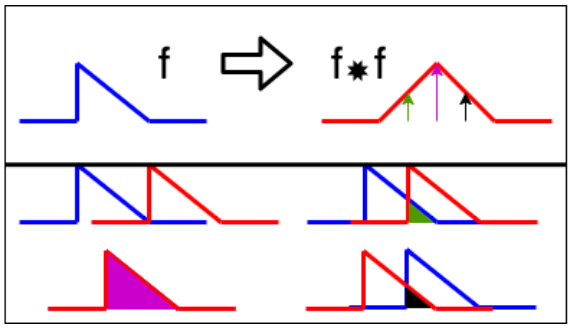
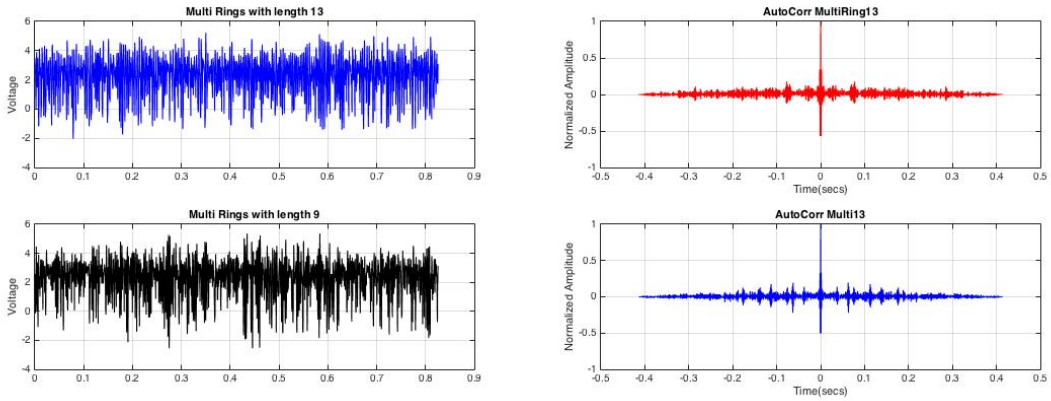



Figure 3.23: Graphical representation of the autocorrelation

Figure 3.23 represents the graphical representation of the same equation.

Furthermore, we carried out this in Matlab and we performed an autocorrelation on the signals from the XOR output of ROs with length of 9 and 13.

Figure 3.24 presents the results of the autocorrelation, as can be seen, the signals degree of similarity is only high when they are in total overlap, other than that only small similarities are appearing. Once again, this analysis showed that we facing a good random behaviour.




a Signals representation

b Autocorrelation graphic results

Figure 3.24: Autocorrelation on XOR output of two ROs of different lengths

### 3.5.2 Jitter analysis on multimode ring oscillators

To analyse the jitter of the ROs, again has been used the jitter wizard of the oscilloscope, which proved to be a very useful tool on the development of the TRNG.

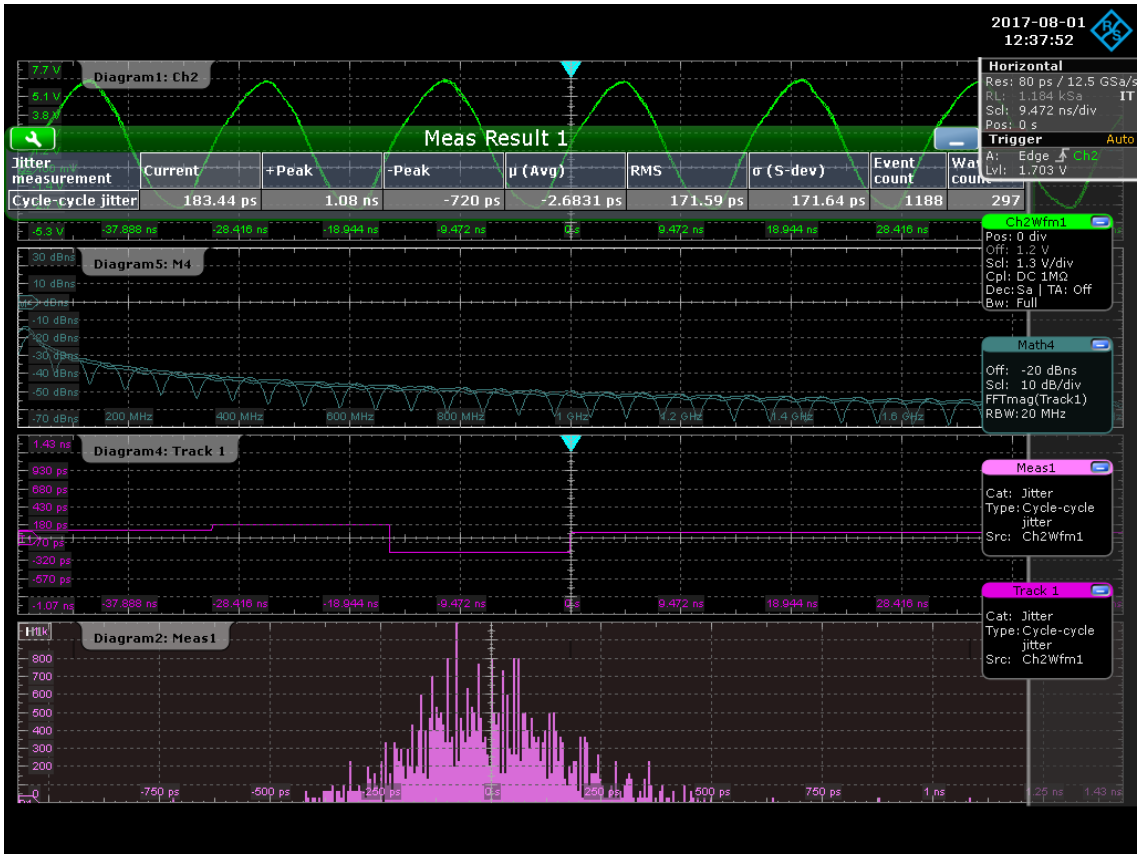



Figure 3.25: Jitter analysis on the length 9 Multimode Ring


Figure 3.26: Jitter analysis on the length 13 Multimode Ring


Figure 3.27: Jitter analysis on the length 15 Multimode Ring

Table 3.3: Standard deviation on multimode RO design

| Length | $\tau$ |
|---|---|
| 9 | $171.64ps$ |
| 13 | $1.85ns$ |
| 15 | $811.85ps$ |
| 21 | $843.73ps$ |


Figure 3.28: Jitter analysis on the length 21 Multimode Ring

The standard deviation on the ROs increased when compared to the previous tests, which once again is a valuable property when trying to achieve true randomness.

Table 3.3 represents the standard deviation from the ROs on the jitter wizard. This increase in uncertainty gives us a good entropy indicator since this ROs will be XORed together with other 8 equal ROs and post ward all with each other, making it close to impossible to trace back the source of the signals or calculate a prediction of the results. The entropy source will be tested in chapter 4 according to worldwide standards in order to check if it complies with the expectations.

Table 3.4: Von Neumann Corrector Rule

| Input | 00 | 01 | 10 | 11 |
|---|---|---|---|---|
| Von Neumann | X | 0 | 1 | X |

## 3.6 Post-processing techniques

Since the first attempts on this RNG design were oversampling the XOR output probably due to high frequency generated by the multimode ROs, a solution was required to remove the bias output.

Several Post Processing techniques where available at the time, between them, the most effective ones were the XOR Corrector [11], Von Neumann Corrector [14], Truncation of defective bits [21], Hash-Functions [17] and finally Linear Code Correctors [16]. After research about this topic, the conclusion relied on 3 methods that were suitable for our system, the XOR Corrector, the Von Neumann and the Truncation of defective bits.

### 3.6.1 XOR Corrector Vs Von Neumann Vs Truncation of defective bits

All of these modules could be implemented to unbias/correct the output of the sampler, each one of them has its own advantages and disadvantages. The XOR Corrector stands for the simplest one and fastest, but not the most efficient. The Von Neumann is efficient but not the fastest. Finally, the Truncation of defective bits, which presents the best results but it has a much larger area than any of these options.

In the first tests of the system, the output we were getting was bias due to some oversampling effect, and for that reason we have decided to implement a corrector. In the final RNG design with fine tuning on the ROs, the implemented corrector worked as just a security measure against some bias effects on the resultant output.

The final choice of the corrector was the Von Neumann because security is important as well as performance, the truncation method comes with much more performance than required, and die area is important on our case. The XOR solutions also presented good results comparing to the Von Neumann approach but was also outperforming the system. The Von Neumann technique, although it appear to be the worst of these comes with good results simply because it will not output any bit if not needed. Therefore, in terms of performance, the Von Neumann seemed to be the most adequate to this RNG.

As can be seen in table 3.4 the corrector takes two consecutive bits and checks if they are equal, if this is true then the corrector does not output nothing, if they are different, then it will apply the rule on the table.

Again, the XOR takes two consecutive bits and submits it to the XOR function, resulting in the output on table 3.5.

The fact that the Von Neumann Corrector does not output anything in some conditions, reduces the original sequence to almost one fourth of it, this does not represent a problem to our system since it produces good random bit sequences most of the time.

Table 3.5: XOR Corrector Rule

| Input | 00 | 01 | 10 | 11 |
|---|---|---|---|---|
| XOR | 0 | 1 | 1 | 0 |

Moreover, it comes as a security measure, as referred. Due to the fact that the output bit rate will be data dependent, if no good data arrives at the corrector then no output will be sent.

## 3.7 Interface

Since this particular board has a RSR232 port available, a interface block has been developed to extract the bit sequences in a practical way for the user. Instead of just looking to waveforms in an oscilloscope or analysing signals in Matlab. This extraction method aims to capture the sequences to the computer in the form of HEX sequences. The most practical way to carry out this task was to implement a Universal Asynchronous Receive and Transmit (UART) Block. This UART block can be found on [4], yet, only the Transmitter block was used since there is need to implement a receiver block due to not having the need to receive data.

Figure 3.29 represents the full interface block implementation.

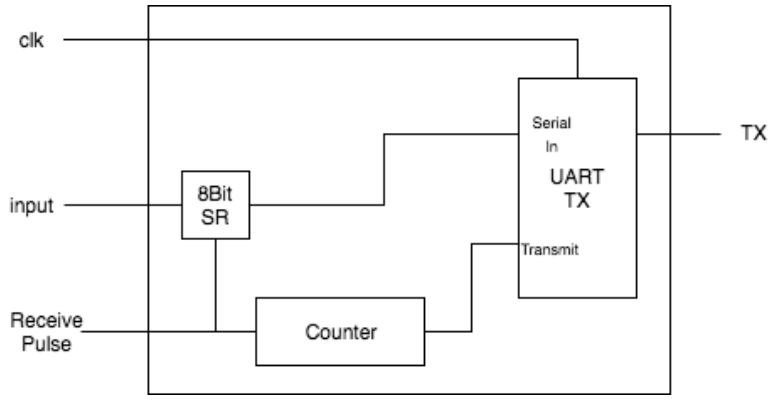



Figure 3.29: Block diagram of a full interface block

### 3.7.1 UART block

UART is one the most widely used protocols in serial communications. Most of IoT devices use this protocol to communicate with sensors. Hence, this is another advantage when implementing this method on our RNG, since most of them already have it implemented. It is also known as 'two wire' communication since it uses two separate lines to communicate, the receiver and the transmitter.

#### 3.7.1.1 UART protocol

UART protocol usually consists of a transmitter and a receiver, in which, one is made to send a pre made frame and the other to decode it, respectively. UART Frames consist of 8 data bits with option to use parity bits to verify if the frame has been received correctly and start, stop bits to indicate the start and the end of the data frame. In our project a UART frame with 8 data bits and no parity was used. Figure 3.30 represents the UART Transmitter frame used for the serial device communication with the computer.

| Start Bit | Data 0 | Data 1 | Data 2 | Data 3 | Data 4 | Data 5 | Data 6 | Data 7 | Stop Bit |
|---|---|---|---|---|---|---|---|---|---|

Figure 3.30: UART Frame

#### 3.7.1.2 UART Transmitter Block

Since there was no need to receive data from the computer to the device, only the UART Transmitter block has been used. This block has been developed according to [4]. The baudrate is a rate of periodic pulses at which the data will be sent in order to achieve a correct communication this baud must be equal on both sides (receiver and transmitter). The block has been set to transmit at a serial rate of 115200 and some parameters were adjusted to obtain this baud. Hence, a baudRate generator was required to obtain the 115200 rate, this generator counts the number of clock signals until a pulse is supposed to happen.

This number of clocks respects to a rule depending on the frequency of the board.

$$ClkCount = \frac{clkFreq}{DesiredBaud} = \frac{50 \times 10^6}{115200} = 434$$

The baudrate generator will then generate a pulse at every 434 clocks.

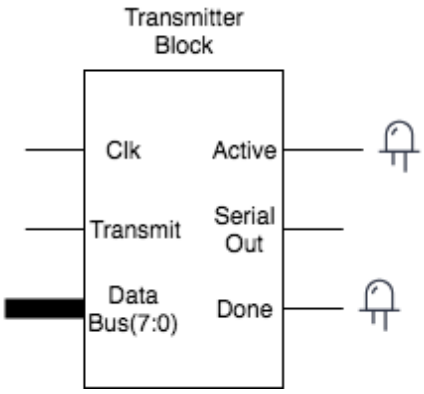



Figure 3.31: UART Transmitter

Figure 3.31 represents the UART Transmitter block used which comprises the input clock to generate the baud rate internally, the data bus where the 8-bit words from the

sampling and correction process will be received, and the output signals which are connected to the board LED's for visual observation of the block functionality and finally the serial output to the RS232 board output.

### 3.7.2 Counter

One of the inputs observed in the previous section comprises a input called 'Transmit', this input is the signal that triggers the UART block that the bus is ready to transmit the data. Hence, a counter had to be developed in order for the system to count when did the correction block outputted the 8-bits.

### 3.7.3 8-bit shift register

In order to save the values while the data bus had not been filled, a Shift Register has been implemented.

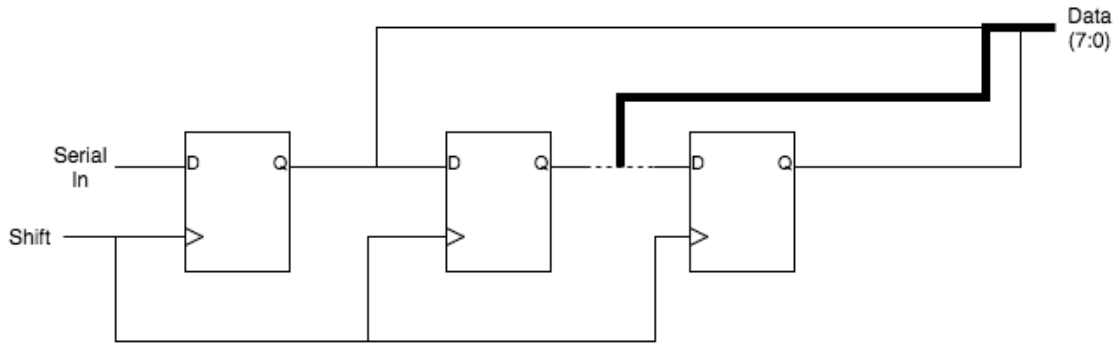



Figure 3.32: 8Bit Shift Register

The shift register will hold the bit values until it is fully refreshed of its old values. When it shifts 8 times, the counter reaches the end and triggers the UART to transmit the data, the process starts all over again, reentering the values on the shift register once again.

## 3.8 Final design

Connecting all of these circuit blocks together and we have a fully integrated TRNG prototype on an FPGA. Figure 3.33 represents the final schematic of the TRNG which comprises all the previous blocks.

The same design can be implemented without post-processing. Figure 3.34 represents the schematic block of that design, in which the post-processing block was removed and the clock signal was connected to the 'ReceiveImpluse' port, which stands for the previous signal that came from the Von Neumann to indicate that another correct bit was delivered. Since this time there are no correct bits, only bits, then at every clock a bit is delivered to the extraction block.

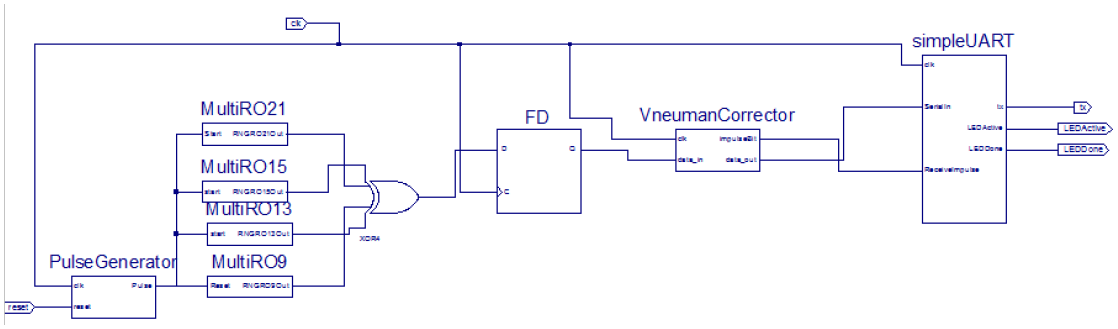



Figure 3.33: Final TRNG schematic

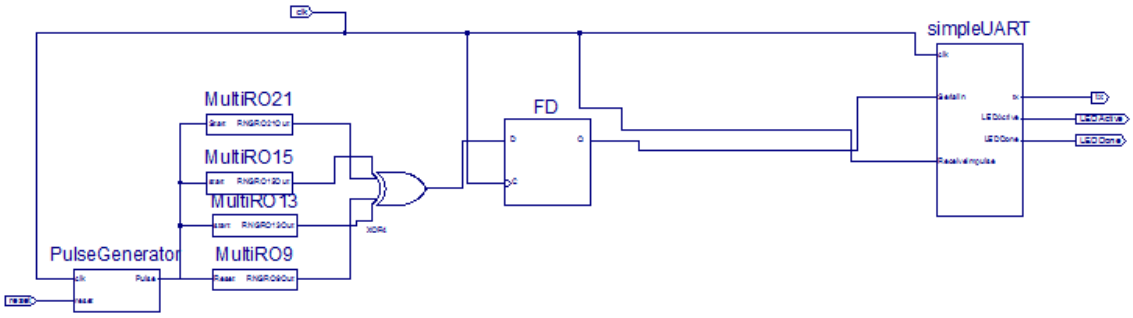



Figure 3.34: Final TRNG schematic without PP

Figure 3.35 represents the final placement on the FPGA, where the ROs have been placed according to what is described in section 3.5. Blue, magenta, navy and red represents the placement of the ROs with length 9,13,15 and 21 respectively. The green represents the corrector as well as the UART Transmitter and, finally, the rose represents the pulse generator.

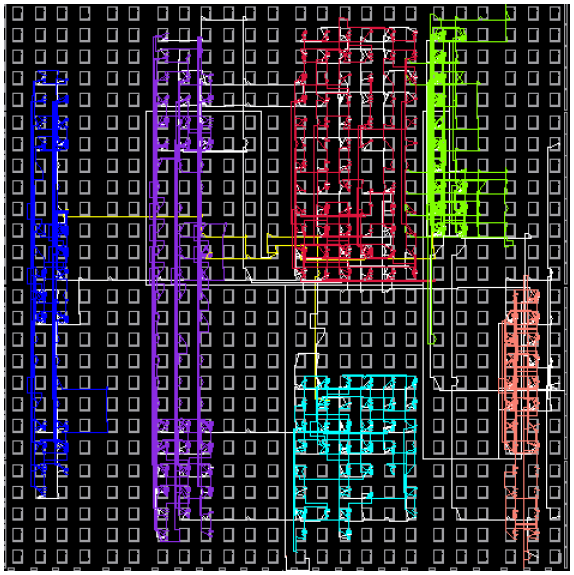


Figure 3.35: Final system placement on the FPGA

Table 3.6: Device utilisation summary with PP

| Logic Utilisation | Used | Available | Utilisation |
|---|---|---|---|
| Total number of Slice Registers | 106 | 3 840 | 2% |
| FlipFlops | 105 | – | – |
| Latches | 1 | – | – |
| Number of 4 input LUT's | 685 | 3 840 | 17% |
| Number of occupied slices | 679 | 1920 | 35% |
| Only related logic | 679 | 679 | 100% |
| Only unrelated logic | 0 | 679 | 0% |
| Total number of 4 input LUTs | 747 | 3 840 | 19% |
| Used as logic | 685 | – | – |
| Used as route-thru | 62 | – | – |
| Number of bonded IOBs | 5 | 173 | 2% |
| Number of BUFGMUXs | 2 | 8 | 25% |

Table 3.7: Device utilisation summary without PP

| Logic Utilisation | Used | Available | Utilisation |
|---|---|---|---|
| Total number of slice registers | 101 | 3 840 | 2% |
| Number of 4 input LUT's | 682 | 3 840 | 17% |
| Number of occupied slices | 672 | 1920 | 35% |
| Only related logic | 672 | 672 | 100% |
| Only unrelated logic | 0 | 672 | 0% |
| Total number of 4 input LUTs | 744 | 3 840 | 19% |
| Used as logic | 682 | – | – |
| Used as route-thru | 62 | – | – |
| Number of bonded IOBs | 5 | 173 | 2% |
| Number of BUFGMUXs | 1 | 8 | 12% |

Table 3.6 presents the device utilisation summary of the FPGA with the post-processing block and Table 3.7 the same design without post-processing. Although these changes are not so significant in terms of FPGA slices, in ASIC design this could be a good indicator when reducing size because the TRNG performed good even without the use of the Corrector, meaning that only the entropy source presented good TRNG results.

Chapter 4

# Statistical tests and analysis

Quality of random numbers are assessed by a set of mathematical and statistic tests. Therefore, international associations of standards provide some tests which are required to achieve security parameters on generators. If the random algorithm or hardware platform passes these tests, it implies that the generator is considered secure.

The most important test suite available is the NIST Test suite from National Institute of Standards and Technology (NIST). One of the compliments found is the standard certification of FIPS140-1, FIPS140-2 and the AIS.31 available by the Bundesamt fur Sicherheit in der Informationstechnik (BSI) Institute in Germany.

As stated before in chapter 1, random number evaluation is based on unpredictability of events, all of these tests and certifications were designed so this unpredictability could be classified and quantified in a measurable way.

To run these tests some suites were developed and distributed, NIST Test suite was used to evaluate on its own methods, cryptool which is a great application to run cryptography methods, available on FIPS140-1 battery tests. Moreover, many documents referred a suite named DIEHARD but this tool seems to be no longer available on the official sources. In this document 3 suites were explored and used:

- NIST
- BSI
- Cryptool

Some changes were implemented in the BSI Test Suite in order to increase performance tests.

## 4.1 FIPS140-1 tests

This certification were carried by Cryptool and has been evaluated in 4 tests.

- Long run Test
- Mono-bit Test
- Poker Test
- Runs Test

### 4.1.1 Long run

The first test of this sequence gets a sample of 20 000 bits and the sequence passes if no long run happen. A long run is defined by maximum length of 34 bits with only 0's or 1's. Not passing the long run test represent a bias result that can be reduced with post processing.

### 4.1.2 Mono-bit test

The mono bit test is based on

$$x = \sum_{j=1}^{20000} b_j$$

This equation will count the number of ones in a 20 000 bit universe. The sequence will pass if the result is $9654 < x < 10346$.

### 4.1.3 Poker test

According to [26] not only unpredictability sequences are considered random but they also need to be scalable and consistent. The sequence is divided into 5 000 consecutive 4 bit segments and it will count and store the number of repetitions happening in the same 4-bits (16 combinations). Consider $F(i)$ as the number of occurrences of each value $i$, where $0 <= i <= 15$

$$X = \frac{16}{5000} \times \sum_{i=0}^{15} [F(i)]^2 - 5000$$

The test will pass if $1.03 < x < 57.4$. The main aim of the test is to record patterns, if one pattern is recursive then one sequence will appear repeated times and the poker result will be too high due to the $F(i)^2$ factor.

### 4.1.4 Runs test

This test is available in a lot of applications for statistical and mathematical analysis, for example, Matlab, Mathematica or Octave. A run is defined like a long run, a sequence

Table 4.1: FIPS140-1 Runs test approval

| Length | Minimum | Maximum |
|---|---|---|
| 1 | 2267 | 2733 |
| 2 | 1079 | 1421 |
| 3 | 502 | 748 |
| 4 | 233 | 402 |
| 5 | 90 | 223 |
| 6 or more | 90 | 233 |

Table 4.2: Changes between FIPS140-1 and FIPS140-2

| Test | Change |
|---|---|
| Long Run | Length 34 to 26 |
| Mono-Bit | $9725 < x < 10275$ |
| Poker | $2.16 < x < 46.17$ |

Table 4.3: Changes between FIPS140-1 and FIPS140-2 on Runs Test

| Length | Minimum | Maximum |
|---|---|---|
| 1 | 2343 | 2657 |
| 2 | 1135 | 1365 |
| 3 | 542 | 708 |
| 4 | 251 | 373 |
| 5 | 111 | 201 |
| 6 or more | 111 | 201 |

of all 1's or 0's, in a bit sequence universe. While the long run had only one maximum length, this set of runs with multiple lengths must lie between a set of intervals to pass.

Consider a sequence $a_1, a_2, a_3, ..., a_2 0000$, a run with length $k$ must be between the $x$ to $y$ interval. To pass this test, the lengths and the intervals must comply with the ones on Table 4.1.

The FIPS140-1 was then updated to FIPS140-2 and some specifications were narrowed. Hence not every sequence may now be random. The changes from FIPS140-1 to FIPS140-2 are the ones described in Table 4.2 and 4.3 .

Althought this changes made some of the random generators to not be considered random, FIPS140-1 has still been used in the process as a preliminary test.

## 4.2 NIST test suite

The U.S. Government and the Institute of Standards developed a set of statistical tests and standards that define the level of security and a secure evaluation for randomness. The last revision of this suite [26], comprises a battery of 15 tests and it complies with the FIPS140-2 specifications as well as its own certification. The tests are:

- Frequency
- Frequency Test within a Block
- The Runs Test
- Test for the longest run of ones in a Block
- The Binary Matrix Rank Test
- The Discrete Fourier Transform (Spectral) Test
- The Non-overlapping Template Matching Test
- Maurer's "Universal Statistical" Test
- The Linear Complexity Test
- The Serial Test
- The Approximate Entropy Test
- The Cumulative Sums Test
- The random excursions test
- The random excursions variant test

Before entering on an overview of the most important tests on this suite, there are a few concepts which need to be introduced in order to fully understand some of the processes involved in the statistical testing. Although there are 15 available tests, we will only give a brief overview over the most important tests of this suite.

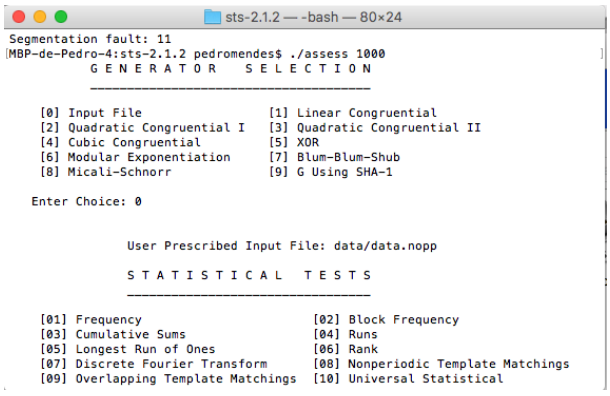



Figure 4.1: NIST Test suite terminal application

### 4.2.1 NIST Test Concepts

These concepts are used in many of the NIST tests as its main functions. Some are used as functions and others as parameters to functions, concepts which are important in order to understand the meaning of the tests as well as what is achieved with them.

Most of these functions come in a package supplied with the suite. They are implemented in ANSI C and can be found by the code in *'math.c'* and consulted by the corresponding header file *'math.h'*.

#### 4.2.1.1 Complementary error function

It is represented by the equation below

$$erfc(z) = \frac{2}{\sqrt{\pi}} \int_z^{\infty} e^{-u^2} du$$

and it graphically it represents the integral of the function on figure 4.2 at a input $z$.

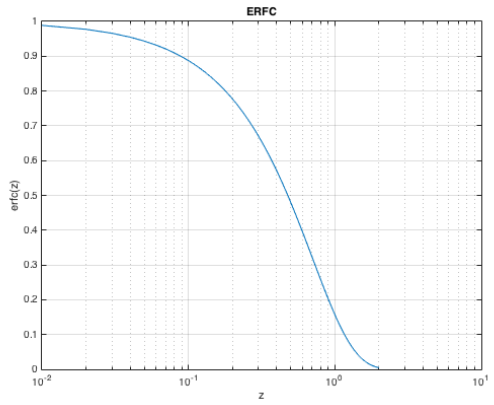



Figure 4.2: Graphical representation of ERFC function

#### 4.2.1.2 Incomplete Gamma function

This function is well known function in mathematic fields, it is mostly used to calculate approximations to some PDF integrals. It is defined as follows [28][39],

$$Q(a,x) \equiv 1 - P(a,x) \equiv \frac{\Gamma(a,x)}{\Gamma(a)} \equiv \frac{1}{\Gamma(a)} \int_x^{\infty} e^{-t} t^{a-1} dt$$

The domain of this function is $Q \in \{[0,1], \forall a \in \Re \mathfrak{e} \wedge 0 \leq x < \infty\}$ , for any value of $a$ and $x$ higher than 0, the value of the function $Q(a,x)$ can only be between 1 and 0.

#### 4.2.1.3 Normal and $\chi^2$ Distribution

These distributions are the most used during the tests performed.

The first one can also be called gaussian distribution, known for its representation in figure 4.3. It is described by:

$$f(x,\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

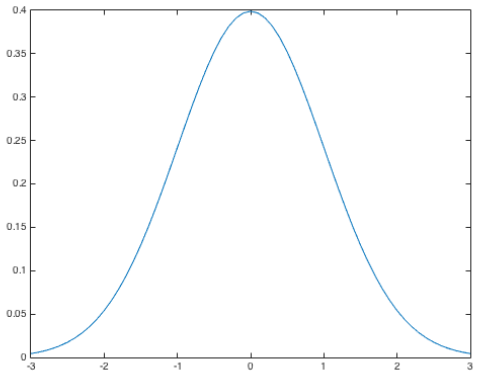



Figure 4.3: Graphical representation of normal distribution

The $\chi^2$ distribution is also a form of PDF which is described by the following equation,

$$f(x,k) = \frac{x^{\frac{k-2}{2}} e^{\frac{-t}{2}}}{2^{\frac{k}{2}}\Gamma(\frac{k}{2})}$$

The graphical representation of the function as well as the influence of the degrees of freedom, $k$, that the function receives as an argument are represented by figure 4.4.

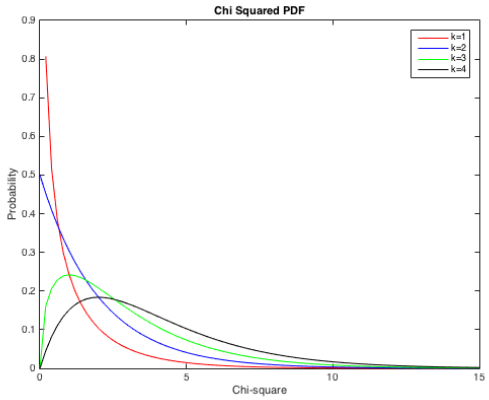



Figure 4.4: Graphical representation of $\chi^2$ distribution with the different degrees of freedom

#### 4.2.1.4 $p_{value}$

This is the dominant parameter discussed alongside this section. It is considered the final parameter to assess the tests results in the suite. It is frequently called the 'tail probability' [26].

The definition of this variable is considered as being the probability, under de null hypothesis, $H$, of obtaining a result equal or more extreme than what was actually observed [7].

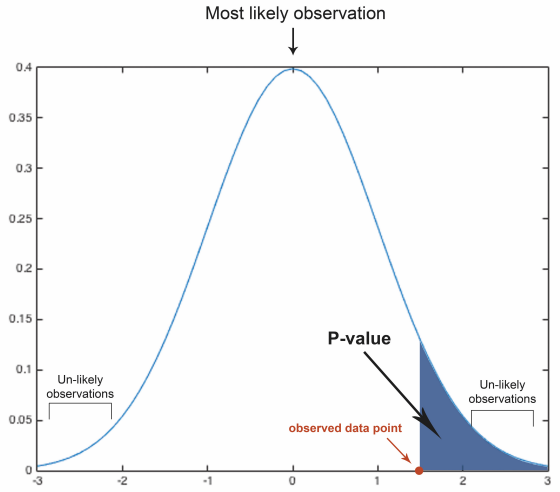



Figure 4.5: Graphical representation of $p_{value}$

Figure 4.5 represents the graphical representation of $p_{value}$, in which it is represented by the area under the gaussian PDF from the observed data point to the limit of the distribution where is the probability of the unlikely events. The $p_{value}$ is calculated using the concepts on sections 4.2.1.4, 4.2.1.2, 4.2.1.1.

### 4.2.2 Frequency test

This test is also known as mono bit. Its application differs from the last one, it still aims to count the number of ones in the sequence but also the number of zeros and assure that its distribution follows the reference statistic of $z = \frac{s_{obs}}{\sqrt{2}}$. This test will assure between a certain interval that the results are all likely to happen. Consider the sequence $\epsilon = 10111010101...$ in which the first bit is $a_1, a_2, ...a_n$, then consider the following sum $S_n = \sum_{i=1}^{n} x_i$ , where $x_i = 2a_i - 1$. Hence, each bit will appear on the sum as +1 or -1 if it was 1 or 0 respectively. The next step is to obtain $s_{obs}$,

$$s_{obs} = \frac{|S_n|}{\sqrt{n}}$$

Finally the $p_{value}$ is calculated through the next equation

$$p_{value} = erfc\left(\frac{s_{obs}}{\sqrt{2}}\right)$$

For the sequence to pass this test and to be considered as random, the $p_{value}$ must be higher than 0.01. If the value is less than that, it is caused by the factor $|S_n|$ or $|s_{obs}|$ being too high meaning that there are long sequences of ones or zeros.

It is recommended that the value $n$ to be higher than 100 for true performance on the test.

### 4.2.3 Frequency test within a block

Another test to count the number of ones and zeroes to assess that the probability of each event to be as close as possible to $\frac{1}{2}$.

This test aims to the same as the previous one, but this time testing scalability of the sequence. This test partition the sequence in $M$-bit blocks. Therefore, a sequence with length $n$, having 4 bit-blocks and being $N$ the number of blocks,

$$N = \frac{n}{M}$$

If $n = 10$ then the number of blocks that result is 2 blocks of 4 bits and the other 2 bits are discarded. Next step on this test is to calculate the proportion $\pi_i$ through the equation below,

$$\pi_i = \frac{\sum_{j=0}^{M} \epsilon_{i-1} M + 1}{M}$$

with $\epsilon_i$ being the bits in each subsequence and for $1 \leqslant i \leqslant N$. This equation will count the proportion of ones in the $N$ blocks. Last step of the test is to compute the $\chi^2$ statistics, where

$$\chi^2(obs) = 4M \sum_{i=1}^{N} \left(\pi_i - \frac{1}{2}\right)^2$$

and compute the $p_{value}$ through the incomplete gamma function (igamc) found on section 4.2.1.2 .

$$p_{value} = igamc\left(\frac{N}{2}, \frac{\chi^2(obs)}{2}\right)$$

For example, consider a sequence $\epsilon = 1001110100$ , once again with $M = 4$, since $n = 10$ then $N = 2$ and 2 bits are discarded, therefore it will end up with $\epsilon_1 = 1001$ and $\epsilon_2 = 1101$, where $\pi_1 = \frac{2}{4} = \frac{1}{2}$ and $\pi_2 = \frac{3}{4}$, and $\chi^2(obs) = 4 \times 2 \times [(\frac{1}{2} - \frac{1}{2})^2 + (\frac{3}{4} - \frac{1}{2})^2] = \frac{1}{2}$ Hence, the $p_{value}$ associated with this sequence is given by

$$p_{value} = igamc\left(1, \frac{1}{4}\right) = 0.7788$$

Again, to pass this test the $p_{value}$ of the sequence must be over 0.01, and since $0.7788 \geq 0.01$ then the sequence is considered random. If $p_{value}$ gets too low ($< 0.01$), this indicates a deviation from the normal proportion of ones in at least one block.

This test is good to evaluate sequences since it calculates the proportion of ones and presents bad results even if one block deviates from the normal distribution. According to [26], the test sequence has to be at least a 100 bits and $n \geq M \times N$, $M < 0.01n$, a maximum $N$ of 100 blocks and recommended bits per block must be over 20.

### 4.2.4 Runs test

This test aims to the same as the runs test from FIPS140, while the methodology is a bit different.

Instead of calculating the length of the runs, this test will compute a test statistic based on a few rules,

$$V_n(obs) = \left[\sum_{k=1}^{n-1} r(k)\right] + 1$$

where $r(k) = 0$ if $\epsilon_k = \epsilon_{k+1}$ else r(k)=1 and $\pi = \frac{\sum j\epsilon_j}{n}$ (which stands for the count of ones divided by the number of bits on the sequence). This equation will then be used to calculate the associated $p_{value}$ with the following,

$$p_{value} = erfc\left(\frac{|V_n(obs) - 2n\pi(1-\pi)|}{2\sqrt{2n}\pi(1-\pi)}\right)$$

Again, for a simple example, having a sequence $\epsilon = 1001110100$ then n=10 and $V_{10}(obs) = [1+0+1+0+0+1+1+1+0]+1 = 6$ then

$$p_{value} = erfc\left(\frac{\left|6 - 2\times 10\times \frac{1}{2}\times(1-\frac{1}{2})\right|}{2\sqrt{2\times 10}\times\frac{1}{2}\times(1-\frac{1}{2})}\right)$$

$$= 0.5271018$$

Once again, if $p_{value} \geqslant 0.01$ then the sequence is considered random by this test. In this case, the value of $V_n(obs)$ is very important to determine the value oscillations in the sequence, with a high oscillating sequence (0101010101010101010..) the $V_n(obs)$ will have a high value, while a slow oscillating sequence (0011001100110011001100...) will have a lower one.

It is recommended that this test consists of a minimum of 1000 bits. This test will stand for the same conclusions as previous runs test, because high oscillating sequences will have less 'runs'.

### 4.2.5 Discrete Fourier Transform (spectral) test

One of the major properties on a random sequence is that all numbers present almost the same occurrences, so, no bias in the sequences. If a number occurs too many times then the sequence is bias to that number. Therefore, this test takes advantage from the Discrete Fourier Transform (DFT) to analyse and detect periodic events. These periodic events represent a bias in the sequence.

Figure 4.6 represents two examples of DFT on random sequences. The first one presents an unbiased output and the second presents some bias values which can be represented by peaks in the FFT. In order to implement the test, the first approach is to convert all the sequence binary values to -1 and 1. Hence, a sequence $\epsilon = 01001000111$ becomes $S = -1,1,-1,-1,1,-1,-1,-1,1,1,1$ The FFT is then applied to S. Next step is to calculate the Modulus of X', where X is

$$X = DFT(S)$$

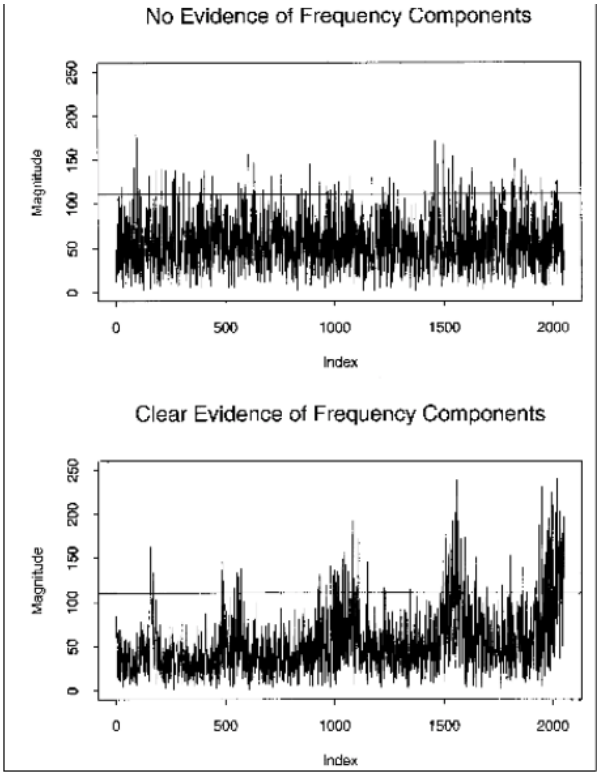



Figure 4.6: DFT Example in [26]

, and X' is the first $\frac{n}{2}$ elements of X. A value T must be computed,

$$T = \sqrt{log\left(\frac{1}{0.05}\right) \times n}$$

which stands for the 95% peak height threshold assuming the randomness of the sequence. Two values needs to be extracted, $N_0$ and $N_1$ ,which stand for the theoretical number of peaks in M below T and the real number of peaks in M below T respectively.

$$N_0 = 0.95 * \frac{n}{2}$$

The last steps aim to obtain the $p_{value}$ in order to decide if the sequence is considered random. First, the factor $d$ is calculated, which stands for a comparison between the theoretical expectations versus the real obtained values.

$$d = \frac{N_1 - N_0}{\sqrt{n(0.95)(0.05)/4}}$$

and finally,

$$p_{value} = erfc\left(\frac{|d|}{\sqrt{2}}\right)$$

Again, the sequence is considered random if $p_{value} \geq 0.01$. Hence, by observing 4.1 where a simple sketch of this test has been developed with Matlab as a preliminary test for periodic events on random sets. The sequence present a $p_{value} = 0.5336 \geq 0.01$, then, it is considered random.

Listing 4.1: Matlab Sketch Spectral Test

```
%%SEQUENCE
ep=[0 1 0 0 1 0 0 0 1 1 1];
X=[ ];
%%%
c= length(ep);
n=1;
for n = 1:c
  if ep(n)== 0
     X(n)= -1;
  else
    X(n)=ep(n);
  end
end
S= fft(X);
S1=S(1:(n/2));
M=abs(S1);
T= sqrt(2.9957*n);
N0=0.95*n/2;
N1=0;
a= length(M);
j=1;
for j = 1:a
  if M(j)< T && M(j) ~= 0
   N1= N1+1;
  end
end

%%%
d=(N1-N0)/(sqrt([n*0.95*0.05/4]));
Pvalue=erfc(abs(d)/sqrt(2));
```

Conclusions on the results are based on $d$ value, if it is too low, there were few peaks below T and many above, which represent that inside the sequence, there are lot of repeatable sequences. This happens in many PRNG with long sequences, in which some patterns begin to appear when long sequences are evaluated.

It is also recommended to apply to a minimum of 1000 bit length.

The presented tests where the most important for our RNG. While all of them where used, only the described are most important to assess if our data was considered as truly random.

## 4.3 AIS.31 in BSI test suite

The last tests carried out for on our RNG has been the AIS.31 certification. This certification is presented as a testing proposal for PRNG and TRNG, through AIS.20 and AIS.31, respectively. For our project, since we are testing it for TRNG, only the AIS.31 certification has been carried out. This certification proposes a series of tests divided by classes

and verifies if the entropy source presents random generation properties [9] [37]. The test suite used to carry this set of tests was the BSI Test Suite which was designed by the Federal Office of Information Security from Germany. This application was developed by this office and it is available on their official website.

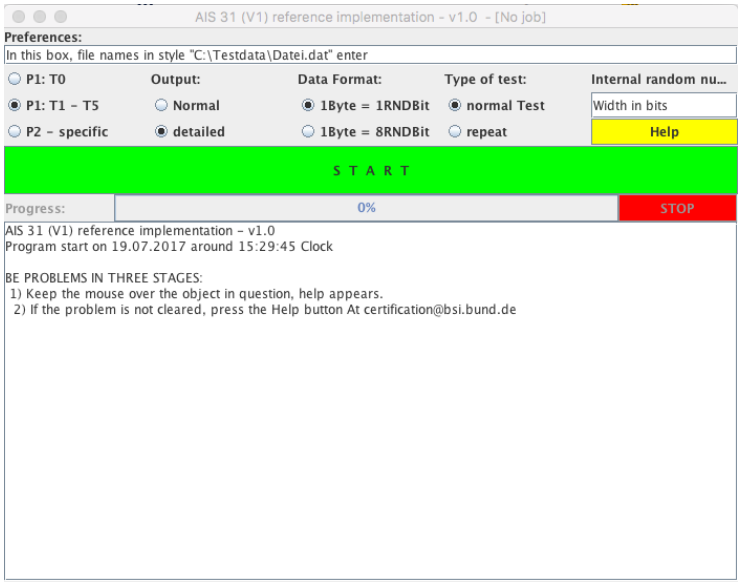



Figure 4.7: BSI Test Suite Application

### 4.3.1 AIS.31 certification

Just like the other sets of tests, this also as its own classes of tests. The classical BSI Test suite with AIS.31 certification come with all the tests from FIPS140-1 Tests while joining some other interesting tests and it is divided by classes of security parameters. Some changes were implemented in order to modify the parameters from FIPS140-1 to FIPS140-2 already described in section 4.1.

The AIS.31 certification is divided into 2 class groups, P1 and P2.

The tests presented by this certification are the following:

- T0 - Disjointness Test
- T1 - Monobit Test
- T2 - Poker Test
- T3 - Runs Test
- T4 - Long Run Test
- T5 - Auto Correlation Test
- T6 - Uniform Distribution Test

- T7 - Multinomial Distribution Test
- T8 - Entropy Test

Where test T0 to T5 are the tests for P1 certification and T6 to T8 are the test for P2 class.

### 4.3.2 Class P1

Class P1 also divides into two groups, a preliminary Test (T0) and the consecutive tests T1 to T5, where T1 to T4 are the same tests as FIPS140-1 presented in section 4.1 and are available in the original suite. Some changes on the application were made in order to achieve the FIPS140-2 approval. This class aims to prove that the RNG is statistically inconspicuous [37].

#### 4.3.2.1 T5 -Autocorrelation test

Considering $\tau \in 1, 2, ....., 5000$, the sequence $\epsilon = a_1, a_2, .....a_{20000}$ and the following equation

$$Z_\tau = \sum_{j=1}^{5000} b_j \oplus b_{j+\tau}$$

the sequence will pass the autocorrelation test if $2326 < Z_\tau < 2674$ Again, all the tests from this class require a minimum sequence length of 25000 bits.

### 4.3.3 Class P2

This next class can be considered as the specific class test, this test has a all new approach. The P2 class aims to guarantee that its impossible to predict the sequence at every time even knowing all its possible predecessors or other sequences from the same generator. Tries to prove the true randomness of sequence again.

One of the major changes in this test is that it will only run with a minimum of 7 200 000 bits and it is composed by the tests T6,T7 and T8.

#### 4.3.3.1 T6 - Uniform distribution test

As the name suggests, this test aims to prove that the sequence follows a uniform distribution. So, by considering the sequence $b_1, b_2, ..., b_n \in 0, 1^k$, it proves the uniform distribution if

$$\frac{1}{n} \times \left| j \leq n | w_j = x \right| \in [2^{-k} - a, 2^{-k} + a] \forall x \in \{0, 1\}^k$$

where $n$ is the length of the string and $a$ is the confidence interval, which in our case was 0.5 [37]

### 4.3.4 T7 - Multinomial distribution test

Having a sequence composed by $w_{i1}, w_{i,2}, ..., w_{i,n}$ where $i \in 1, 2, ...h$ , according to the null hypothesis, a statistical model where the hypothesis that an event alone is responsible for the results, the samples are identical for multinomial distributions. Calculating

$$p_t := \frac{f_1[t] + f_2[t] + ...f_h[t]}{h \times n}$$

, where

$$f_i[t] := \left|\left\{j : w_{ij} = t\right\}\right|$$

, and t stands for the number of samples $t \in 0, 1, ..., s-1$ Again, by the null hypothesis,we calculated the test variable

$$\sum_{i=1}^{h} \sum_{t=0}^{s-1} \frac{(f_i[t] - np_t)^2}{np_t}$$

which is approximately $\chi^2$ distributed with (h-1)(s-1) degrees of freedom.

### 4.3.5 T8 - Entropy Test

The last test on this section aims to the same objective as the approximate entropy test in NIST suite. Having a bit sequence that is segmented in several non overlapping sequences with length $L$. It will count the distance $(A_n)$ from the previous subsequence.

$$A_n = \begin{cases} n, if\ no\ i < n\ exist\ with\ w_n = W_{n-i} \\ min\{i | i >= 1, w_n = w_{n-i}\}, if\ others \end{cases}$$

The accepted distances for a random sequence approval are derived from the Coron Test Table [9] This test can also be compared to Maurer's Test from on NIST suite.

## 4.4 Extraction method

With the UART communication protocol and relying on the RS232 port available on the Xilinx Spartan 3 Board. This port was connected to the computer using a RS232 to USB cable, which is recognised in the computer as a serial port. The numbers were extracted using a terminal application from 'Braypp' which turned out to not be so accurate as expected. The terminal application used has been TeraTerminal version 4.95 (SVN 6761). The application opened the serial port to communicate at 115200 baud with 8bit data, no parity and 1 stop bit.

The terminal presented the random sequences in HEX format, which are then pasted on a .bin file to later analysis.

Figure 4.8 represents the extraction process of random numbers.

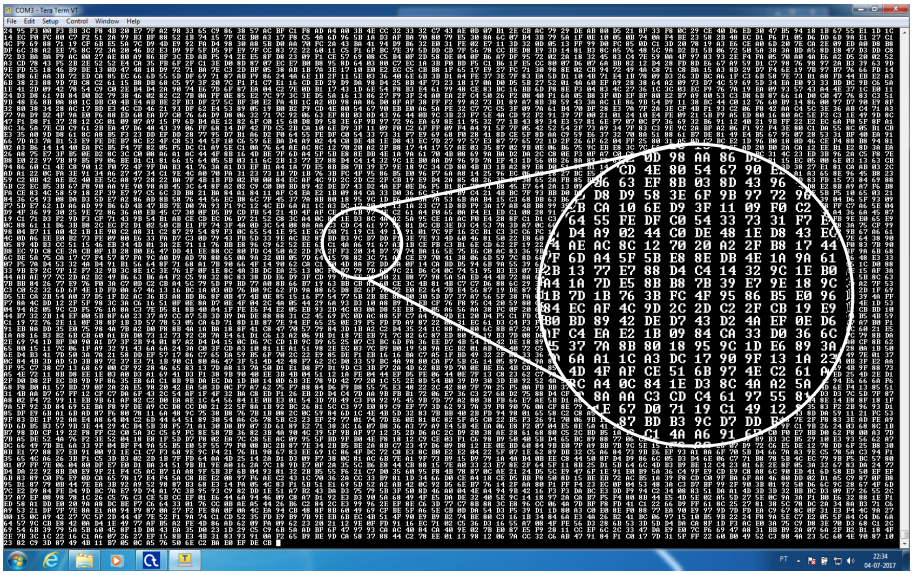


Figure 4.8: Extraction of Random Numbers with TeraTerm through UART

## 4.5 Test Results

In this section we submitted our sequences to the tests described earlier.

To ensure that the sequences are according to the expected, we drawn out 4 histograms in Matlab with 4 Long sequences from the TRNG.

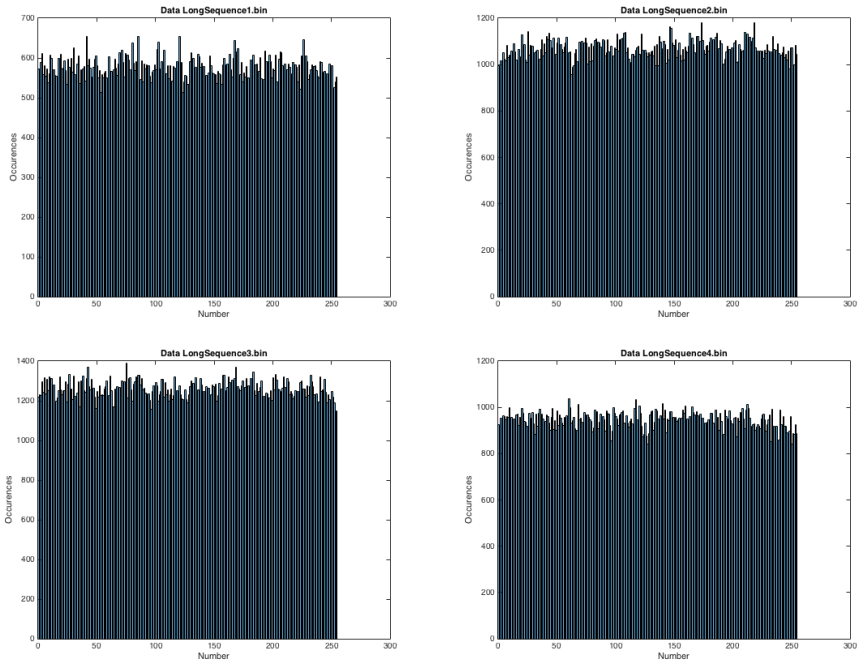
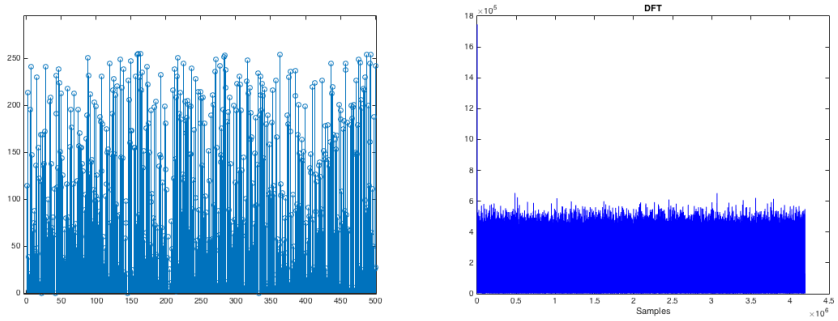



Figure 4.9: Histogram on multiple Long Sequences

Figure 4.9 represents those histograms and we can observe that there are no trends

on results, meaning that there are no bias on the output, the histograms represent a uniform distribution as expected. Also, using the important properties of the DFT already discussed on sections 4.2.5 and 3.5.1. The DFT was applied in a sequence using Matlab. Figure 4.10 represents a sample of the stemplot and the DFT. Again, the DFT shows no frequency components on the sequence, which is a good indicative that we are in the presence of a TRNG.




a Stem Plot of a part of the Long Sequence b DFT on Long Sequence

Figure 4.10: Long Sequence DFT Analysis

### 4.5.1 FIPS140-1 on the Cryptool

Just as described, these tests where carried out as a preliminary tests to check if the TRNG output was eligible to continue with the evaluation. For this we used CrypTool 1.4.31 Beta 6b.

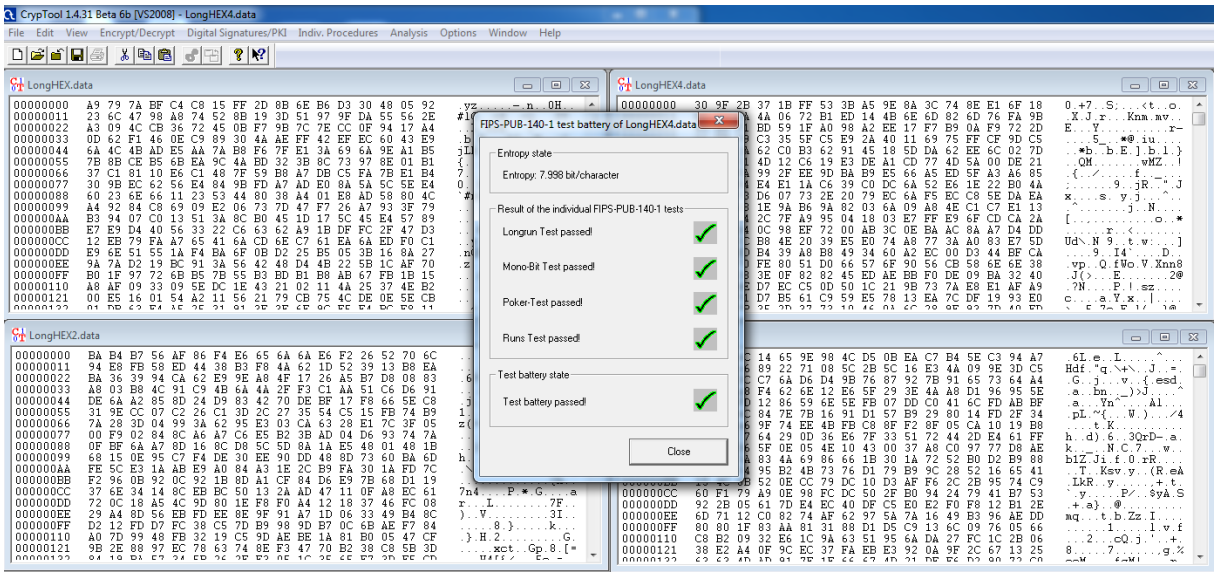



Figure 4.11: FIPS140-1 Tests with Cryptool V1.2.31

The TRNG passed in all the tests as can be seen in figure 4.11, averaging a entropy of 7.998 bit/character. Cryptool also has another tools which provided us some data

Table 4.4: NIST statistical tests

| Test | Bitstream Length | Success Rate | Average $p_{value}$ | Result |
|---|---|---|---|---|
| Frequency | 1000 | 498/500 | 0.5271 | PASSED |
| Block Frequency | 100 | 499/500 | 0.4865 | PASSED |
| CSums | 1000 | 500/500 | 0.5621 | PASSED |
| Runs | 1000 | 496/500 | 0.5260 | PASSED |
| L.Run of 1's | 200 | 495/500 | 0.4952 | PASSED |
| Rank | 60 000 | 496/500 | 0.5027 | PASSED |
| FFT | 1000 | 498/500 | 0.5617 | PASSED |
| Universal | 400 000 | 98/100 | 0.3978 | PASSED |
| Approximate Entropy | 100 | 497/500 | 0.4760 | PASSED |
| Serial | 1 000 000 | 40/40 | 0.4564 | PASSED |
| Linear Complexity | 1 000 000 | 40/40 | 0.6082 | PASSED |
| Overlapping Template | 1 000 000 | 39/40 | 0.2717 | PASSED |

about the sequence, one of them is the periodicity tool, which will search for a repetition of a certain sequence of characters with length $k(k >= 1)$ from a certain position in the document.

This analysis to the document presented no periodicity in any part of the long sequence.

### 4.5.2 NIST test suite

The most important test on this TRNG was carried out by the NIST test suite. As described in section 4.2, this suite presents one the most complete test on random specifications and standards accepted worldwide.

Every TRNG/PRNG available on consumer applications and products must be accepted by the standards created by this Institute.

In table 4.4 , 500 bitstreams were tested, the request to pass the tests where that the minimum 488 streams should pass, and we are above that in all of them.

The Approximate entropy test was done with a bitstream length of 100 and the block length $m = 1$, the Serial test was done with $10^6 bits$ and $m = 12$, Linear Complexity test was done again with $10^6 bits$ and $m = 1000$, and finally Block Frequency was carried out with block Length $m = 10$, all values of the block lengths where determined according to [26].

Since the NonPeriodic Overlapping Templates matching, Random excursions and random excursions variant where more exhausting tests in which several streams where required.

The NonOverlapping Templates Matching the test has been done with 500 bitstreams of 40 000 bits each, also the block length $m$ was defined with the value 9 which returned 148 test results, the results averaged a pass of about 494/500 with the lowest results being 4 times 487/500 and one 486/500. The Random Excursions test was carried out with 17

Table 4.5: BSI statistical test results on FIPS140-1

| Test | Average | Result |
|---|---|---|
| T0 | – | PASSED |
| T1 | 9975 [x] | 257/257 |
| T2 | 14.1760 [x] | 257/257 |
| T3 | — | 257/257 |
| T4 | — | 257/257 |
| T5 | 2496 $[z_\tau]$ | 257/257 |
| T6 | – | PASSED |
| T7 | – | PASSED |
| T8 | 7.9959 $[A_n]$ | PASSED |

Table 4.6: BSI statistical test results on FIPS140-2

| Test | Average | Result |
|---|---|---|
| T0 | – | PASSED |
| T1 | 9969 | 257/257 |
| T2 | 16.596 | 257/257 |
| T3 | – | 256/257 |
| T4 | – | 257/257 |
| T5 | 2531$[z_\tau]$ | 257/257 |
| T6 | – | PASSED |
| T7 | – | PASSED |
| T8 | 7.998 $[A_n]$ | PASSED |

bitstreams with 1 000 000 bits each, getting 17/17 in 7 excursions of the test having one 16/17, the variant on this test was carried out with the same parameters, but achieving 13 excursions with a approval of 17/17 and the rest having 16/17.

### 4.5.3 AIS.31 on BSI test suite

In this section, we will present the results associated with the BSI Test suite. which compromises the tests available on FIPS140-1 and FIPS140-2 alongside with a new ones. Although FIPS140-1 was already applied to the bitstreams, it was not extensively used on a long set of data, the BSI test suite provided us tools to do that.

Table 4.5 and 4.6 presents the results on the test results from the BSI Test suite. The RNG passed all of the tests from all the classes in both FIPS140-1 and FIPS140-2 parameters except from one run with the RunsTest with FIPS140-2 parameters and 8bit numbers. where a run of ones had a maximum interval of [2343; 2657] and the run got a value of 2662 which came a little high than the previous runs. Again, the tests confirmed that we are in the presence of a TRNG system and this also will serve as proof for true randomness results.

Table 4.7: NIST statistical tests on [33] with post-processing

| Test | Success Rate | Average $p_{value}$ | Result |
|---|---|---|---|
| Frequency | 493/500 | 0.4866 | PASSED |
| Block Frequency | 498/500 | 0.0487 | PASSED |
| CSums | 493/500 | 0.5086 | PASSED |
| Runs | 487/500 | 0.02553 | PASSED |
| L.Run of 1's | 494/500 | 0.0462 | PASSED |
| Rank | 497/500 | 0.2596 | PASSED |
| FFT | 498/500 | 0.5617 | PASSED |
| Universal | 99/100 | 0.9240 | PASSED |
| Approximate Entropy | 497/500 | 0.000012* | PASSED |
| Serial | 36/40 | — | PASSED |
| Linear Complexity | 40/40 | 0.5852 | PASSED |
| Overlapping Template | 39/40 | 0.4846 | PASSED |

## 4.6 Comparison

As a comparison with this project, we went with the XOR design from [32] with same length rings with auto placement. All the parameters associated with the rings where the same found on [33] and tests were carried out with the VonNeumann corrector.

One of the main changes on the proposed design is based on the number of each ring. Since our FPGA platform doesn't have the same amount of free slices then the ring lengths stayed the same but the amount of rings was reduced from 40 to 25 each.

Table 4.7 represents the results when the TRNG was submitted to the NIST Tests. The results proved that the previous design also stands as a TRNG and the results proved that when passing all of the tests available. Althought the results didn't prove so much difference from our project, since both being considered as TRNG, some of the tests perform better on our RNG than this one but not significantly better.

Cryptool also has some other tools who provided us a little more data about this tests.

One of those tools is an analysis based on the triangle problem of the Heilbronn type [36], and assumes that the values on the sequence are considered as co-ordinates of points in a unit square. These points are then linked to random triangles in which the surface area is considered. It is expected that a uniform distributed sequence will rise and the non uniform will decline.

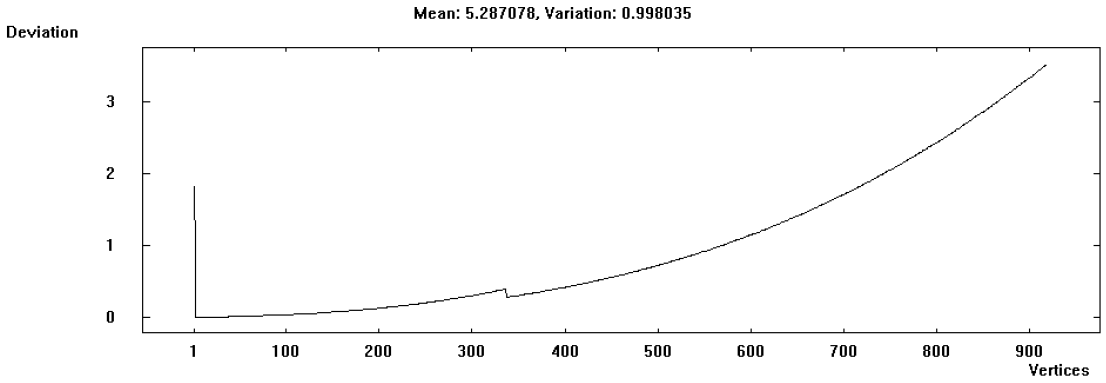



Figure 4.12: Vitanyi Graph on multimode XOR design with Cryptool

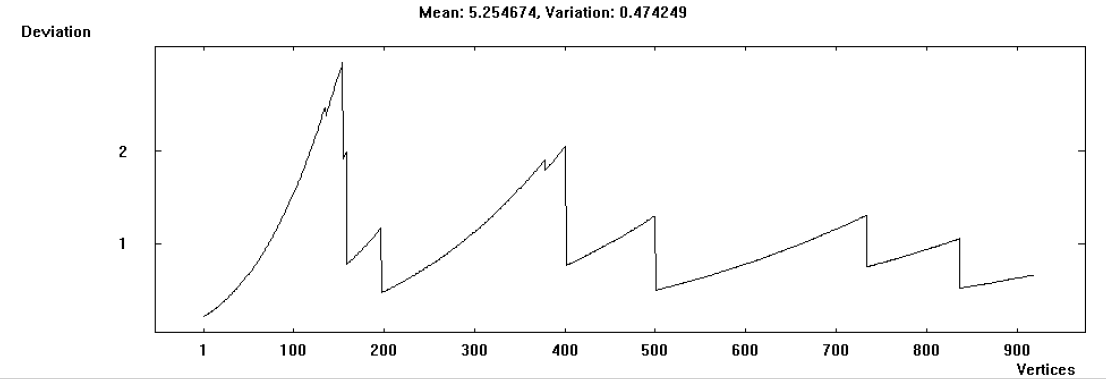



Figure 4.13: Vitanyi Graph on [33] with Cryptool

Figure 4.12 and 4.13 presents the graphic output of this analysis where we can see that the [33] approach presented a result that declined more that our approach, which according to [36], presents that our sequence is 'more' uniform than the previous approach.

Another comparison that was carried out was the importance of the Post Processing block on both designs.

As can be seen on table 4.8 our TRNG does not put all of the importance of the output on the Post Processing Block. The post processing block is a secure measure against external influences which tend to bias the output.

Also, using cryptool FIPS140-1 Battery tests, our design passed all of the tests while classic XOR design[33] only passed long run test.

There is no absolute definition on how random a sequence should be, consequently, all we can comment on is how much guesses we can do and how many of those proved to be right.

Another comparison carried out was with the Linux RNG /random. Just like discussed in chapter 1, one of the main safety tools used to encrypt data on our devices is based on complex algorithms to obtain entropy. Table 4.9 presents the result of a random generation method widely used based on linux random entropy when submitted to NIST

Table 4.8: NIST statistical tests on [33] and our project without post-processing

| Test | Classic XOR [33] | Our TRNG design |
|---|---|---|
| Frequency | 184/500 | 497/500 |
| Block Frequency | 496/500 | 499/500 |
| CSums | 194/500 | 497/500 |
| Runs | 426/500 | 493/500 |
| L.Run of 1's | 484/500 | 494/500 |
| Rank | 495/500 | 494/500 |
| FFT | 494/500 | 494/500 |
| Universal | 0/100 * | 97/100 |
| Approximate Entropy | 487/500 | 497/500 |
| Serial | 26/40 | 39/40 |
| Linear Complexity | 40/40 | 40/40 |
| Overlapping Template | 0/40* | 39/40 |

test suite.

Table 4.9: NIST statistical tests on Linux

| Test | Success Rate | Average $p_{value}$ | Result |
|---|---|---|---|
| Frequency | 49/500 | 0.000504 | FAILED |
| Block Frequency | 500/500 | 0.7442 | PASSED |
| CSums | 77/500 | 0.000795 | FAILED |
| Runs | 496/500 | $2e^{-6}$ | FAILED |
| L.Run of 1's | 483/500 | 0.2780 | FAILED |
| Rank | 496/500 | 0.5027 | PASSED |
| FFT | 454/500 | 0.2457 | FAILED |
| Universal | 98/100 | 0.3978 | PASSED |
| Approximate Entropy | 454/500 | 0.1678 | FAILED |
| Serial | 0/5 | — | FAILED |
| Linear Complexity | 5/5 | 0.6496 | PASSED |
| Overlapping Template | 0/5 | — | FAILED |

Most of the tests presented to this algorithm failed, having just one of the tests where it even performed better than our project which was the Block Frequency test.

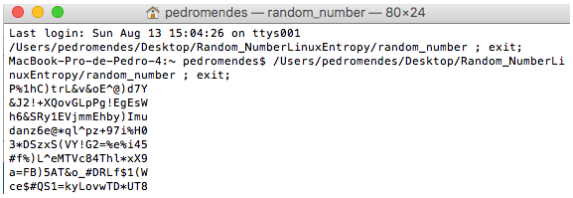



Figure 4.14: Linux Entropy RNG

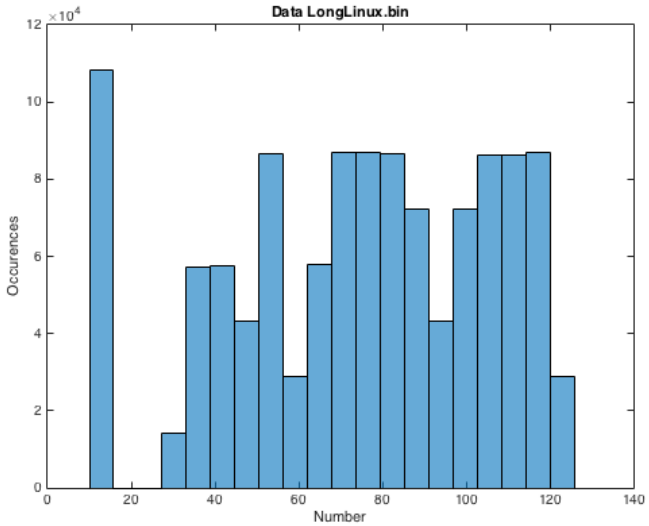



Figure 4.15: Linux Entropy RNG Histogram

Figure 4.15 presents a histogram of that RNG output, which presents a different distribution from the TRNG that has been developed in this document.

Chapter 5

# Conclusions

According to all the work developed on this document, we can conclude that we designed , implemented and evaluated a full TRNG system with extraction methods for straightforward implementation on FPGA devices.

The RNG developed presented good results on all the standard certifications for worldwide security, so we can claim that our project is secure against many of the attacks happening today on devices. Since this design implies that a physical phenomenons are happening, than, this implementation due to its 'physicality' stand as a major drawback for hackers that rely on software hacking techniques.

One of the major challenges on the implementation has been the environment optimisation tools, XST, which stood out a true challenge when trying to implement some redundant logic.

Although all of the issues, the TRNG has also been studied in some of its major properties, which proved some of the concepts associated with ring oscillators but some had not been tested yet in FPGA targets.

The statistical suites used performed according to all the documentation provided and came as a major learning to the developers to understand all of the concepts behind randomness and how can it be quantified.

Although there are some designs which will fit better on small and portable IoT devices, this design stands a tested secure design which will fit the high standards on cloud servers, while some of those small designs can't.

Finally, we can also conclude that random generators present a foremost importance topic and a extremely interesting technology to work with. Also, FPGA's present once again the proof that they are not the tech for yesterday but a technology of the future.

## 5.1 Future Work

As future work on this project we suggest the next developer three major challenges:

- To port the design to ASIC and since the ASIC design will perform faster and possibly having better results, some changes on the lengths of the rings as well as the number of rings to develop.
- Attempt new ring designs on this method and develop an intensive study using the test suites we used to assess the best ring design on this method.
- As a final task, to test with the specific tools for the required work to develop an intensive study on the strengths of the design against the majority of the attacks on physical RNG's.

# Bibliography


[1] Anritsu. *Jitter Analysis Basic Classification of Jitter Components using Sampling Scope*. Tech. rep.

[2] M. Bucci, L. Germani, R. Luzzi, A. Trifiletti, and M. Varanonuovo. "A high-speed oscillator-based truly random number source for cryptographic applications on a smart card IC". In: *IEEE Transactions on Computers* 52.4 (2003), pp. 403–409. ISSN: 00189340. DOI: 10.1109/TC.2003.1190581.

[3] J. Cartagena, H. Gomez, and E. Roa. "A fully-synthesized TRNG with lightweight cellular-automata based post-processing stage in 130nm CMOS". In: *NORCAS 2016 - 2nd IEEE NORCAS Conference* (2016). DOI: 10.1109/NORCHIP.2016.7792898.

[4] P. P. Chu. *FPGA PROTOTYPING BY VHDL EXAMPLES*. John Wiley & Sons, Inc., 2008. ISBN: 9780470185315.

[5] P. P. Chu and R. E. Jones. "Design Techniques of FPGA Based Random Number Generator". In: *Quantum* (1999), pp. 1–6.

[6] M. Dichtl and J. Golic. "High-Speed True Random Number Generation with Logic Gates Only". In: (2007).

[7] F. Dorey. "The P Value: What is it and what does it tell you?" In: *Clinical Orthopaedics and Related Research* 468.8 (2010), pp. 2297–2298. ISSN: 0009921X. DOI: 10.1007/s11999-010-1402-9.

[8] J. Eusébio. "Geradores de Sinais Aleatórios baseados em Caos para Aplicações em Telecomunicações". MA thesis. NOVA School of Science and Technology, 2017.

[9] "Evaluation of random number generators". In: *Bundesamt fur Sicherheit in der Informationstechnik* 0.10 ().

[10] G. Evans, J. Goes, A. Steiger-Garção, M. Ortigueira, N. Paulino, and J. Lopes. "Low-voltage low-power CMOS analogue circuits for Gaussian and uniform noise generation". In: (2003). DOI: 10.1109/ISCAS.2003.1205521.

[11] V. Fischer and M. Drutarovsky. "True Random Number Generator Embedded in Reconfigurable Hardware". In: *LNCS* (2003), pp. 415–430.

[12] A. Hajimiriand and T. H. Lee. "A general theory of phase noise in electrical oscillators". In: *IEEE J. Solid-State Circuits* 33 (Feb. 1998), pp. 179–194.

[13] H. Handschuh, P. Paillier, and J. Stern. "Probing Attacks on Tamper-Resistant Devices". In: *Igarss 2014* (2014). ISSN: 0717-6163. DOI: 10.1007/s13398-014-0173-7.2.

[14] J.V.Neumann. *Various Techniques used in connection with Random Digits*. 1951.

[15] W. Kester. "Converting Oscillator Phase Noise to Time Jitter". In: *App Note*. 2009, pp. 1–10.

[16] Y.-s. Kim, J.-w. Jang, and D.-w. Lim. "Linear Corrector Overcoming Minimum Distance Limitation for Secure TRNG from (17, 9, 5) Quadratic Residue Code". In: *ETRI J.* 32.1 (2010), pp. 93–101. DOI: 10.4218/etrij.10.0109.0141.

[17] H. Kong, B.-j. Wang, H.-j. Cao, Y.-h. Wang, and H.-g. Zhang. "Random number generator of bp neural network based on sha-2 (512)". In: *Int. Conf. Machine Learning Cybern.* 2.1 (2007), pp. 2708–2712.

[18] M. Mandal and B. Sarkar. "Ring oscillators : Characteristics and applications". In: *Indian Journal of Pure & Applied Physics* 48.February (2010), pp. 136–145. ISSN: 0019-5596.

[19] H. B. Mann. "Statistical Independence in Probability, Analysis and Number Theory". In: *SIAM Review* 5.3 (1963), pp. 292–293. ISSN: 0036-1445. DOI: 10.1137/1005084.

[20] M. Matsumoto, S. Yasuda, R. Ohba, K. Ikegami, T. Tanamoto, and S. Fujita. "1200um2 physical random-number generators based on SiN MOSFET for secure smart-card application". In: *Digest of Technical Papers - IEEE International Solid-State Circuits Conference* 51 (2008), pp. 44–46. ISSN: 01936530. DOI: 10.1109/ISSCC.2008.4523233.

[21] M.A.Zidan, A.G. Radwan and K.N.Salama. "Random Number Generation Based on Digital Differential Chaos". In: *Int.Midwest Symp. Circuits Syst.* 2 (2011), pp. 2–5.

[22] M. S. Mcclure. "Digital Jitter Measurement and Separation". PhD thesis. Faculty of Texas Tech University, 2005.

[23] C. S. Pétrie and J. Alvin Connelly. "A noise-based ic random number generator for applications in Cryptography". In: *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 47.5 (2000), pp. 615–621. ISSN: 10577122. DOI: 10.1109/81.847868.

[24] P. Pouliquent, K. A. Boahent, and A. G. Andreou. "A Gray-Code Current-Mode Analog-to-Digital Converter Design". In: *Proc. ISCAS'91* 2 (1991).

[25] B. Razavi. "A study of injection locking and pulling in oscillators". In: *IEEE Journal of Solid-State Circuits* 39.9 (2004), pp. 1415–1424. ISSN: 0018-9200. DOI: 10.1109/JSSC.2004.831608.

[26] A. Rukhin, J. Soto, J. Nechvatal, S. Miles, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo. "A statistical test suite for random and pseudorandom number generators for cryptographic applications". In: *National Institute of Standards and Technology* 800.April (2010), p. 131. DOI: 10.6028/NIST.SP.800-22r1a.

[27] M Sahithi, B Muralikrishna, M Jyothi, K Purnima, A. J. Rani, and N. N. Sudha. "Implementation of Random Number Generator Using LFSR for High Secured Multi Purpose Applications". In: 3.1 (2012), pp. 3287–3290.

[28] I. W. Selesnick. "Laplace random vectors, Gaussian noise, and the generalized incomplete gamma function". In: *Proceedings - International Conference on Image Processing, ICIP* (2006), pp. 2097–2100. ISSN: 15224880. DOI: 10.1109/ICIP.2006.312821.

[29] M. Song, Y. Lee, and W. Kim. "A Clock Feedthrough Reduction Circuit for Switched-Current Systems". In: *IEEE Journal of Solid-State Circuits* 28.2 (1993), pp. 133–137. ISSN: 1558173X. DOI: 10.1109/4.192044.

[30] "Spartan-3 FPGA Starter Kit Board". In: 130 (2008), pp. 1–64.

[31] G. E. Suh and S. Devadas. "Physical unclonable functions for device authentication and secret key generation". In: *Proceedings - Design Automation Conference* (2007), pp. 9–14. ISSN: 0738100X. DOI: 10.1109/DAC.2007.375043.

[32] B. Sunar, W. Martin, and D. Stinson. "A Provably Secure True Random Number Generator with Built-In Tolerance to Active Attacks". In: *IEEE Transactions on Computers* 56.1 (2007), pp. 109–119. ISSN: 0018-9340. DOI: 10.1109/TC.2007.250627.

[33] P. S. Sundaram. "Development of a FPGA-based True Random Number Generator for Space Applications". PhD thesis. Linkoping Institute of Technology, 2010.

[34] T. E. Tkacik. "A Hardware Random Number Generator". In: *Cryptographic Hardware and Embedded Systems* (2002), pp. 450–453. DOI: 10.1007/3-540-36400-5_32.

[35] I. Vasyltsov, E. Hambardzumyan, Y. S. Kim, and B. Karpinskyy. "Fast digital TRNG based on metastable ring oscillator". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 5154 LNCS (2008), pp. 164–180. ISSN: 03029743. DOI: 10.1007/978-3-540-85053-3_11.

[36] P. Vitányi, T. Jiang, and M. Li. "Kolmogorov Complexity and a Triangle Problem of the Heilbronn Type". In: ().

[37] W. Killmann, W. Schindler. "A proposal for : Functionality Classes and evaluation methodology for true (physical) random number generators". In: *Bundesamt fur Sicherheit in der Informationstechnik* Ver 3.1 (2001).

[38] J. Walrand. "Lecture Notes on Probability Theory and Random Processes". In: *Department of Electrical Engineering and Computer Sciences* (2004).

[39] E. S. Watson. "Applications of Incomplete Gamma Functions to the Incomplete Normal Distribution Incomplete Normal Distribution". In: 1.1 (2015), pp. 1–6.

[40] N. H. Weste and K. Eshraghian. "Principles of CMOS VLSI Design". In: *Reading, MA: Addison-Wesley* (2010), pp. 231–237.

[41] K. W.Paul. "The Design and Analysis of a True Random Number Generator in a Field Programmable Gate Array". In: *Proc. of International Symposium on FPGAs* (2004).

[42] K. Yang, D. Fick, M. B. Henry, Y. Lee, D. Blaauw, and D. Sylvester. "16.3 A 23Mb/s 23pJ/b fully synthesized true-random-number generator in 28nm and 65nm CMOS". In: *Digest of Technical Papers - IEEE International Solid-State Circuits Conference* 57 (2014), pp. 280–281. ISSN: 01936530. DOI: 10.1109/ISSCC.2014.6757434.

ANNEX I

# ANNEX 1 - SCH AND VHDL FILES

Listing I.1: LogicGate.vhdl

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity logic_gate is
5     Port (
6       logic_in : in STD_LOGIC;
7       logic_out : out STD_LOGIC);
8   end logic_gate;
9
10  architecture Behavioral of logic_gate is
11      begin
12        logic_out <= NOT logic_in;
13  end Behavioral;
```

Listing I.2: Ro3instance.vhdl

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity RO3Block is
6    Port (
7        EN_in : in STD_LOGIC;
8        ro3_in : in STD_LOGIC;
9      ro3_out : out STD_LOGIC);
```

```
10      end RO3Block;
11
12    architecture Behavioral of RO3Block is
13       component logic_gate is
14           Port (
15             logic_in : in STD_LOGIC;
16             logic_out : out STD_LOGIC);
17       end component;
18
19     constant stage_nr : integer := 2;
20      signal pulse_buff : STD_LOGIC := '0';
21   signal stage_array : std_logic_vector(0 to stage_nr);
22   signal temp : STD_LOGIC;
23     begin
24       gen: for i in 0 to stage_nr-1 generate
25         logic_gate_inst: logic_gate port map(stage_array(i),
              stage_array(i+1));
26       end generate gen;
27       temp<=NOT (ro3_in AND EN_in);
28       stage_array(0) <= temp; --EN_in;
29       pulse_buff <= stage_array(stage_nr);
30       ro3_out <= pulse_buff;
31
32
33   end Behavioral;
```

Listing I.3: RO9.vhdl

```
1   library ieee;
2  use ieee.std_logic_1164.ALL;
3  use ieee.numeric_std.ALL;
4  library UNISIM;
5  use UNISIM.Vcomponents.ALL;
6
7  entity RO9 is
8     port ( start   : in    std_logic;
9            Rng_out : out   std_logic);
10 end RO9;
11
12 architecture BEHAVIORAL of RO9 is
13    signal linha1        : std_logic;
```

```
14    signal linha2        : std_logic;
15    signal Rng_out_DUMMY : std_logic;
16    component RO3Block
17       port ( EN_in   : in    std_logic;
18              ro3_in  : in    std_logic;
19              ro3_out : out   std_logic);
20    end component;
21
22 begin
23    Rng_out <= Rng_out_DUMMY;
24    ro3_ins1 : RO3Block
25       port map (EN_in=>start,
26                 ro3_in=>Rng_out_DUMMY,
27                 ro3_out=>linha1);
28
29    ro3_ins2 : RO3Block
30       port map (EN_in=>start,
31                 ro3_in=>linha1,
32                 ro3_out=>linha2);
33
34    ro3_ins3 : RO3Block
35       port map (EN_in=>start,
36                 ro3_in=>linha2,
37                 ro3_out=>Rng_out_DUMMY);
38
39 end BEHAVIORAL;
```

Listing I.4: Vneuman.vhdl

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity VneumanCorrector is
5      Port ( clk : in std_logic;
6          data_in : in std_logic;
7        impulseBit: out std_logic;
8          data_out : out std_logic);
9  end VneumanCorrector;
10
11 architecture Behavioral of VneumanCorrector is
12
```

```
13      signal counter : integer range 0 to 1;
14      signal reg : std_logic_vector(1 downto 0);
15      signal res : std_logic;
16
17  begin
18
19      proc_corr : process(clk)
20      begin
21          if rising_edge(clk) then
22              reg(counter) <= data_in;
23          impulseBit <='0';
24          end if;
25
26        if rising_edge(clk) then
27              if counter = 0 then
28                  counter <= 1;
29              else
30                  counter <= 0;
31             impulseBit <='0';
32              end if;
33          end if;
34
35        if counter = 1 then
36        if reg(0) = reg(1) then
37          impulseBit<='0';
38        else
39          res<=reg(0);
40          impulseBit<='1';
41        end if;
42      end if;
43
44      end process;
45
46      data_out <= res ;
47
48  end Behavioral;
```

Listing I.5: LongChain51.vhdl

```
1
2   library IEEE;
```

```
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity LongChain51 is
6      Port (
7          clk_in : in STD_LOGIC;
8          pulse_out : out STD_LOGIC);
9   end LongChain51;
10
11 architecture Behavioral of LongChain51 is
12     component logic_gate is
13         Port (
14             logic_in : in STD_LOGIC;
15             logic_out : out STD_LOGIC);
16     end component;
17
18     constant stage_nr : integer := 51;
19     signal pulse_buff : STD_LOGIC := '0';
20    signal stage_array : std_logic_vector(0 to stage_nr);
21 begin
22     gen: for i in 0 to stage_nr-1 generate
23         logic_gate_inst: logic_gate port map(stage_array(i),
              stage_array(i+1));
24     end generate gen;
25     stage_array(0) <= clk_in;
26     pulse_buff <=NOT(stage_array(stage_nr) AND clk_in);
27     pulse_out <= pulse_buff;
28
29
30 end Behavioral;
```

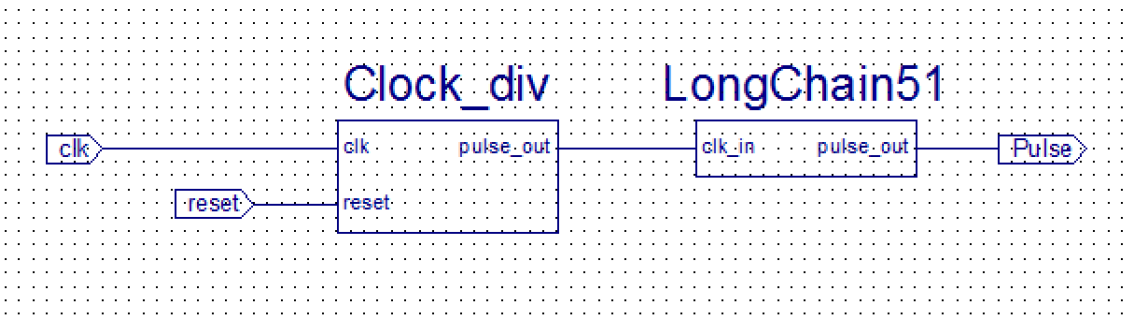



Figure I.1: Pulse Generator SCH

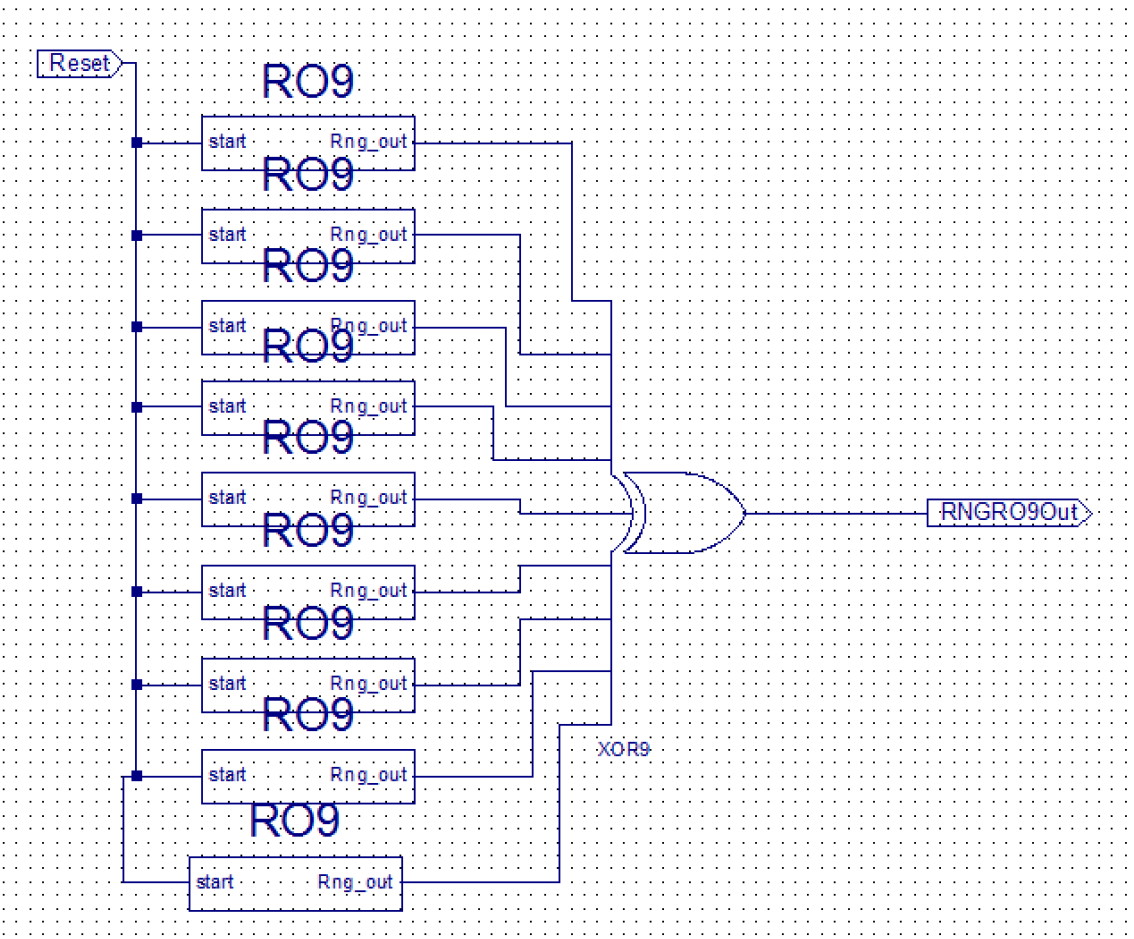
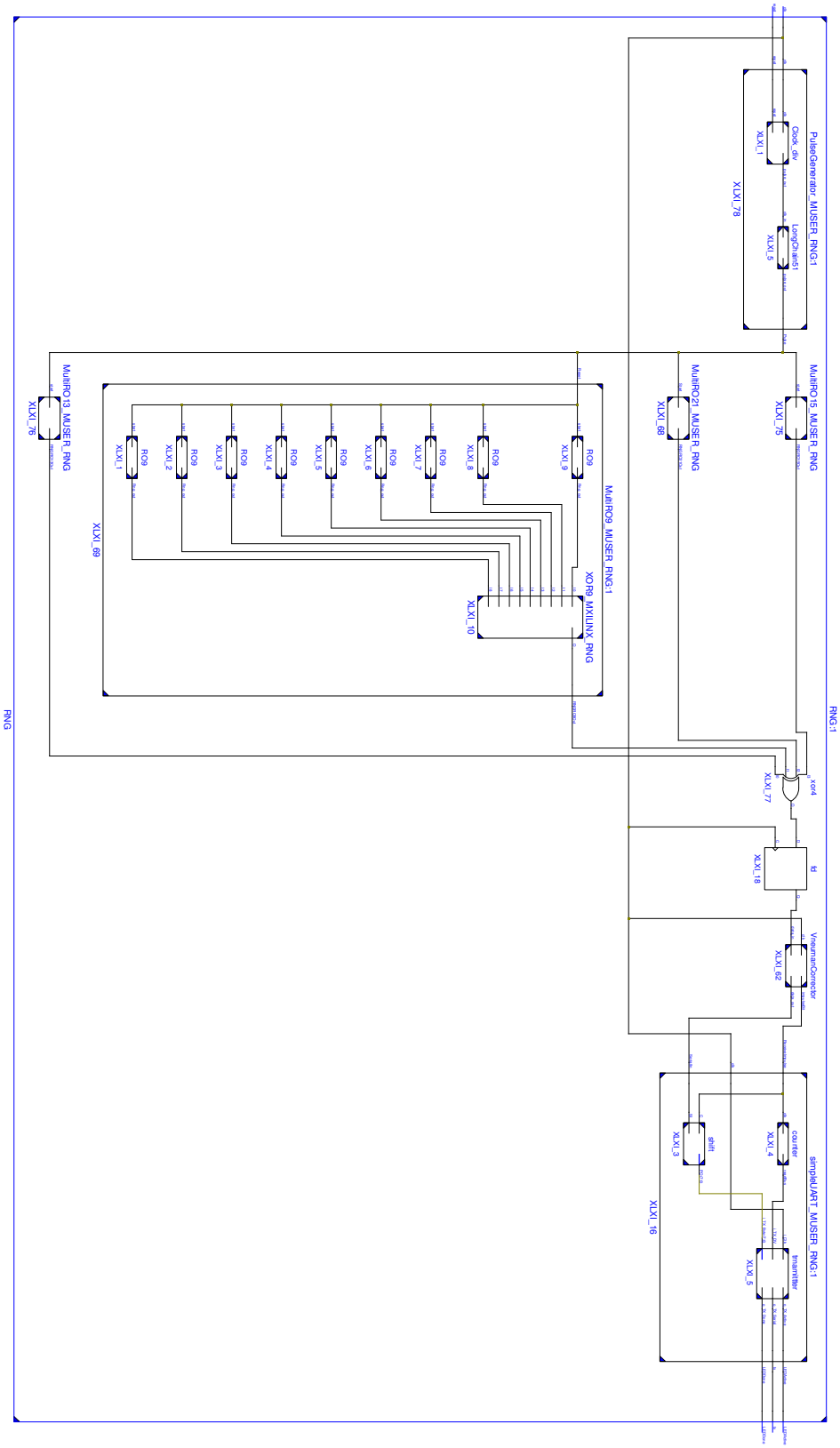



Figure I.2: Multi RO Length 9 SCH

ANNEX II

# ANNEX 2 - RTL DESIGN


RNG
RNG:1
PulseGenerator_MUSER_RNG:1
Clock_div
LongChain51
XLXI_78
XLXI_1
XLXI_5
MultiRO15_MUSER_RNG
XLXI_75
MultiRO21_MUSER_RNG
XLXI_68
MultiRO13_MUSER_RNG
XLXI_76
MultiRO9_MUSER_RNG:1
XLXI_69
RO9
XLXI_1
XLXI_2
XLXI_3
XLXI_4
XLXI_5
XLXI_6
XLXI_7
XLXI_8
XLXI_9
XOR9_MXILINX_RNG
XLXI_10
xor4
XLXI_77
fd
XLXI_18
VirtexInterconnector
XLXI_62
simpleUART_MUSER_RNG:1
XLXI_16
shift
XLXI_3
counter
XLXI_4
transmitter
XLXI_5