FPGA based entropy source for cryptographic use

Emma Hilmersson

Business Security
Lund

Advisor: Kennet Bckman at Business Security
and
Thomas Johansson at Electrical and Information Technology, LTH

August 12, 2013

Printed in Sweden
E-huset, Lund, 2013

Abstract

Random number generators, RNGs, are an important part of cryptographic modules and earlier there have been standards and evaluations schemes for all part of the module except the RNG. Evaluation schemes for entropy sources used as random number generations have now become mandatory in countries such as Germany and the U.S. The first part of this report summarises the requirements for these evaluation schemes with the purpose to pass in both countries.

The second part of the report focuses on implementations of random number generators in FPGA and different designs are discussed and compared to the summary of requirements. One of the designs with a possibility , according to the author, to pass the evaluations was implemented in three identical development boards with an Altera FPGA. The chosen implementation was also simulated using Modelsim Altera 10.1b to analyse the behaviour of the source, both with respect to the surroundings, such as temperature, and different placement and routing.

The final part is to compare the result of the implementation and simulation to the summary of the requirements. Some of the requirements were harder to satisfy and in the conclusions a discussion is made on what to do to be able to fulfil all the requirements and if it is possible at all.

Acknowledgements

I would like to thank my supervisor Kennet Bäckman at Business Security in Lund for all your guidance and enthusiasm throughout this thesis.

I will also like to thank Business Security for the opportunity to doing my thesis at your company and your kind hospitality.

Table of Contents

List of Figures

List of Tables

Aim and outline of this thesis

Random number generators (RNG) are an important part in all cryptographic systems where random numbers are used, such as in key generation processes, authentication protocols and to seed pseudo-random number generators. Until recently there have been standards for all parts of a cryptographic module, except for the RNG, and the source of the randomness in the RNG; the entropy source. Evaluation of a random number generator used in a cryptographic system is mandatory in Germany and the USA, according to AIS31 and SP800-90B respectively. The first part of this report summarises the requirements and a combined list is created. An RNG fulfilling the requirements of this combined list would then be able to pass the evaluation in both countries. The International Organization for Standardization, ISO have together with the International Electrotechnical Commission, IEC published another standard, 18031 which will be briefly discussed as well.

The aim of this project is then to study if it is possible for an entropy source from academic publications to pass the requirements given in the evaluation schemes. Some implementations, from academic publications, of random number generators are discussed, while trying to find one with a possibility to pass the evaluation. Only implementations aimed at FPGAs have been considered.

Section 2 in this thesis discusses the basics about random number generators and constructions. Section 3 summarises the requirements for entropy sources, both for AIS31, SP800-90B and for ISO/IEC 18031. In section 4 different implementations in FPGA are discussed with the aim to find one that can fulfil the requirements presented in section 3. Section 5 have details on the implementation for this project while section 6 shows the results for the first implementations. Timing simulations and the results from them are presented in section 7 and section 8 focuses on how well the requirements are fulfilled. Section 9 includes conclusions and future work.

Chapter 1

Introduction to random number generators

Random number generators, RNGs, are a major part in cryptographic applications and are used to create numbers that appear to be random.

A RNG can be divided into three categories: deterministic, true (non-deterministic, physical) and non-physical true(hybrid) random number generators. The difference between a deterministic (DRNG) and a true (TRNG) random number generator is that the DRNG uses a deterministic algorithm while the TRNG uses an uncontrollable physical process. A non-physical true random number generator (NTG) is a combination of a DRNG and a TRNG, where the TRNG repeatedly seeds a DRNG. This report focuses on finding a good entropy source, which can be used in a TRNG, discussed more in section 1.1. A basic TRNG consist of an entropy source, which is the part that gives entropy, and the output, as in Figure 1.1.

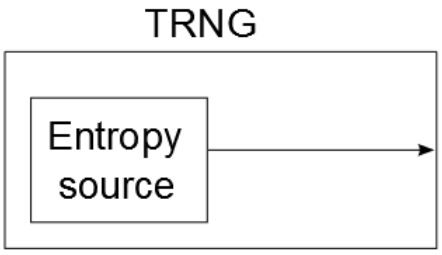

Figure 1.1: Basic design of a true random number generator

A more useful design of TRNGs is shown in Figure 1.2, where the raw random numbers are the output from the noise source and the external random numbers are the numbers out to the user. These sequences of numbers will be the same, if not processed by an algorithm.

1.1 Entropy Source

As mentioned earlier a TRNG uses an uncontrollable physical process as the entropy source and there are a few different types that can be used.

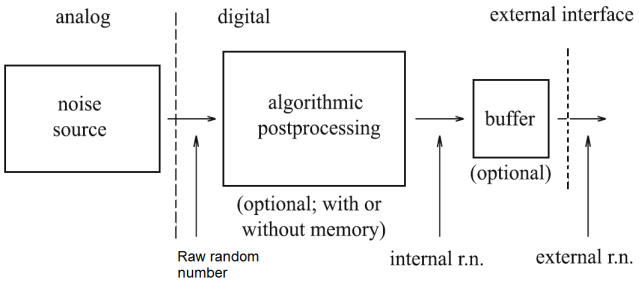

Figure 1.2: Implementation model

1.2 Post processing algorithm

As seen in Figure 1.2 a post-processing algorithm is an optional part of a TRNG implementation. The post-processing algorithm is used to increase the entropy and decrease statistical defects of the raw random numbers and this can be done in different ways. Some common post-processing algorithms are:

XOR corrector: A function that applies an exclusive-or operation on a block of size n bits to generate one output bit. The drawback of this is that the bit-rate is reduced n-times, but it reduces the bias and it is possible to have a constant bit-rate.

Von Neumann corrector: A function that tries to get a balanced distribution of '1' and '0' bits, in other words lowers the bias. It takes a pair of bits and uses the first bit if they are different otherwise it throws away both of them. The output bit-rate will depend on the data.

Chapter 2
Requirements

2.1 Introduction

Two evaluation schemes, AIS31 and SP800-90B, have become mandatory in Germany and the USA respectively. To be able to get a product passed the evaluation in both these countries it is important to study the similarities and differences in these schemes. The purpose of an evaluation scheme is to increase confidence in not just the random output but in the whole design. The reason for not only evaluating the output is that testing this data can only indicate if it is satisfactory from a statistical point of view. The tests can not tell the difference between a deterministic and truly random data generation. A post-processing algorithm for example, may transform data with poor or non existing random properties so that they pass all statistical tests. Statistical tests can still be important in order to distinguish how the output differs from an ideal true random sequence.

The evaluation must also consider tests of the random source as quality self-tests, which shall be based on the knowledge about the specific random number generator.

Some general requirements for a random number generator(RNG) are that; the output should have good statistical properties, the output should be unpredictable, the source should be robust(to resist attacks) and it should be possible to test the source.

2.2 AIS31

AIS31 is the German evaluation and certification scheme [1], for a True Random Number Generator(TRNG). AIS31 contains requirements from the Federal Office for Information Security(The Bundesamt für Sicherheit in der Informationstechnik(BSI)) and has been mandatory in Germany, and part of the scheme is mandatory in France, since 2001. In [1] are the updated versions of [2] and [3]. BSI is a German government agency responsible for communication security. In AIS31 there are properties and criteria that a TRNG should fulfil together with some tests to pass the evaluation. The main goal in this evaluation is to estimate the entropy per random bit.

AIS31 have different predefined classes for random number generators and for Physical TRNG:s, PTG.1,2 and 3.

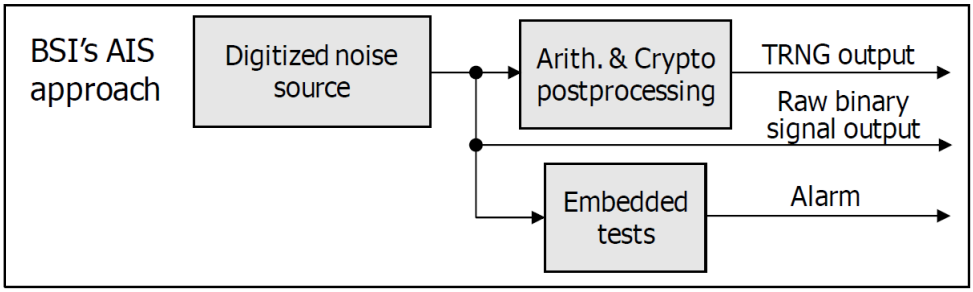

Figure 2.1: Implementation model according to AIS31

- PTG.1 is the lowest security class and can be used to generate random numbers but the output may be guessable.
- PTG.2 generates high-entropy random numbers and these numbers may be practically indistinguishable from independent and uniformly distributed random numbers(as the output from an ideal RNG).
- PTG.3 defines the highest security class in AIS31 and includes both the physical part and a cryptographic post-processing algorithm.

A PTG.2 can be used in a PTG.3 and since this project focuses on the quality of the entropy source, the requirements for a PTG.2 will be in focus.
As seen in Figure 2.1 the embedded tests are an important part of the entropy source and according to the AIS31 scheme the embedded tests are:

Total Failure Test The total failure test have the following requirements:

A1: Shall detect a total failure of the entropy source.

A2: If a total failure occurs while the RNG is being operated, the RNG shall either : i) numbers generated after the breakdown will be output ii) If a post-processing algorithm of class DRG.2 is used [1] the random sequence generated after the breakdown can be used, if the internal state entropy of the DRG.2 guarantees the claimed output entropy (i.e all generated entropy comes from the deterministic algorithm.)

Either requirement i) or ii) must be used, but since this project studies only entropy sources i) is used.

Online Tests The online tests are embedded tests that shall fulfil the following requirements :

A3: Detect non-tolerable statistical defects of the raw random sequence. This should be done both at start-up and while operating.

A4: The online test at start-up has to be successful before any random numbers can be generated.

[1] A Deterministic random number generator of class 2, more information in [2]

A5: The online test shall be able to detect non-tolerable weaknesses in an acceptable time.

A6: Shall test the quality of the raw random number sequence.

A7: The online test can be triggered externally, at regular intervals, continuously or at specific internal events.

A8: Shall detect non-tolerable statistical defects of the raw random numbers within an acceptable time.

A9: The online test should detect if the entropy decreases, by ageing, tolerance of components, environmental stress, even though it does not result in a total failure of the source.

The random sequence shall also be able to meet the following criteria:

A10: Test procedure A does not distinguish the internal random numbers from output sequences of an ideal RNG.

A11: The average Shannon entropy (see section 8.2) per internal random bit exceeds 0.997.

Requirement A11 can be checked by the statistical test procedure B, if the raw random numbers are binary-valued.

If a post-processing algorithm is used then a PTG.2 also includes some of the requirements from class PTG.1. If no post-processing algorithm is used, PTG.2.3 and PTG.2.5 will cover PTG.1.3 and PTG.1.4.

(PTG.1.3) The online test detects non-tolerable statistical defects of the internal random numbers. The online test is [selection: triggered externally, applied after regular intervals, applied continuously, applied upon specified internal events]. When a defect is detected, the output of further random numbers is prevented.

(PTG.1.4) Within one year of typical use, the probability that an online alarm occurs is in the order of 10^{-6} or larger if the RNG works properly.

Other requirements for a PTG.2 are:

A12: A PTRNG should include an internal entropy source, a digitization mechanism of the signal, a (if necessary) post-processing algorithm, the online test, a total failure test and a start-up test, all according to Figure 2.1.

A13: The internal random numbers can have an entropy defect, e.g. the numbers can have a bias, and a post-processing algorithm is not required.

A14: If the post-processing algorithm(if any) does not reduce the average entropy per bit, the average entropy per internal random bit equals at least the average entropy per raw random bit.

A15: A stochastic model substantiated by statistical tests should provide evidence that the entropy of the internal random numbers is large enough.

2.2.1 Evaluation methods

If the PTRNGs generate 1-bit raw random numbers there are two methods, A and B, that may be used for evaluation.

Method A

- A.1 (A16) The stochastic model should show that the raw random numbers are stationary distributed and no significant(long-step) dependencies occurs, which are not a part of test procedure B.
- A.2 (A17) The raw random numbers shall pass the statistical test procedure B under all relevant environmental conditions.
- A.3 (A18) There shall be included a proof that the post-processing algorithm does not reduce the entropy per bit. Or that the average entropy per internal random bit is large enough.
- A.4 (A19) The internal random numbers shall pass the statistical test procedure A (and other statistical standard test suites if applied) under all relevant environmental conditions.

Method B

- B.1 (A16)
- B.2 (A20) The developer verifies on the basis of the stochastic model that due to the post-processing algorithm the entropy per internal random number is sufficiently large. Under suitable conditions test procedure B might support this goal.
- B.3 (A17/A19) The internal random numbers pass the statistical test procedures A and B (and other statistical standard test) under all relevant environmental conditions

The evaluation method is chosen depending on your possibilities and if method B is applied, three cases are possible: (i) Test suite B is passed if it verifies the entropy per raw random numbers; (ii) Test suite B fails if the entropy per random numbers could not be verified; or (iii) It is not possible to access the raw random numbers or the numbers are not binary-valued, this leads to the result that test suite B cannot be applied.

These methods can only be used if the source generates a single raw random bit per time unit, if not another evaluation method needs to be performed.

2.2.2 Stochastic model

An important part in the evaluation by AIS31 is the stochastic model [1], [4] which shall fulfil the requirements in A15, A16 and A20. A stochastic model is part of the evaluation criteria because it is necessary to have an entropy per bit rate that is high enough and the main purpose with the stochastic model is to give

a probability distribution of the internal random numbers. It should also identify the characteristics that can affect the distribution of the random numbers. A model of the distribution of the raw random number shall be included.

2.2.3 Test suite for BSI

Some steps in the evaluation methods were to perform statistical tests and in AIS31 scheme the tests are divided into two procedures, A and B, more about the test procedures can be read in [2] section 2.4.4.
Test procedure A: Test if internal random numbers behave statistically inconspicuously by test T0-T5.
Test procedure B: Test to ensure that the entropy per raw bit is sufficiently large, by test T6-T8.
Both of these test procedures can be performed by the software downloaded from BSI [].

T0 : Disjointness
$w_1, ..., w_{2^{16}}$ if the members are pairwise different it will pass the disjointness test.

T1 : Monobit test
The bit sequence $b_1, ..., b_{20000}$ pass the monobit test if $9654 < X < 10346$ for the sequence

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

T2 : Poker test
For $j = 1, ..., 5000$ let $c_j = 8 \cdot b_{4j-3} + 4 \cdot b_{4j-2} + 2 \cdot b_{4j-1} + b_{4j}$ and $f[i] :=| j : c_j = i \mid$. The bit sequence $b_1, ..., b_{20000}$ pass the poker test if $1.03 < Y < 57.4$, where

$$Y = (16/5000) \cdot (\sum_{j=0}^{15} f[i]^2) - 5000.$$

T3 : Runs test
The maximum sub-sequence of consecutive zeroes or ones. The permitted intervals for the run lengths are shown in table.

Run length	Permitted interval
1	2267-2733
2	1079-1421
3	502-748
4	233-402
5	90-223
≥ 6	90-233

T4 : Long run test
The bit sequence $b_1, ..., b_{20000}$ will pass the long run test if no long runs

occur. A long run is a run of length ≥ 34.

T5 : Autocorrelation test
The bit sequence $b_1, ..., b_{20000}$ will pass the autocorrelation test if $2326 < Z_\tau < 2674$,where

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

T6 : Uniform distribution
The uniform distribution test with the sequence $w_1, ..., w_n \in 0, 1^k$ will pass, with parameters (k,n,a) if

$$\frac{1}{n} \cdot \mid j \leq n \mid w_j = x \mid \in [2^{-k} - a, 2^{-k} + a]$$

for all $x \in 0, 1^k$.

Parameter k is the length of the vectors to be tested, n is the length of the sequence to be tested and parameter a is a positive real number.

T7 : Comparative test for multinomial distribution
A sample $w_{il}, ..., W_{i,n}$ for $i \in \{1, ..., h\}$ are considered with assumed values from the set $\{0, 1, ..., s-1\}$.The null hypothesis says that for multinomial distribution the individual samples are identical. Let the relative frequency for the occurrence of t, from all the sample be: $p_t := (f_1[t] + ... + f_h[t])(hn)$ and for $t \in \{0, ..., s-1\}$ let $f_i[t] := |\{j : w_{ij} =\}|$ By null hypothesis

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

should be χ^2 -distributed with $(h-1)(s-1)$ degrees of freedom.

T8 : Entropy estimation
Used for estimating the entropy of the sequence. The bit sequence $b_1, ..., b_{(Q+K)L}$ is segmented into non-overlapping output words $w_1, ..., w_{Q+K}$ with the length L. The distance from w_n to its predecessor with the same value is called A_n and is determined as:

$$A_n = \begin{cases} n, & \text{if no } i < n \text{ exist in } w_n = w_{n-1}; \\ min\{i | i \geq 1, w_n = w_{n-i}\}, & \text{in all other cases;} \end{cases}$$

2.3 NIST SP800-90B

The recommendation SP800-90B (draft), Recommendation for the Entropy Sources Used for Random Bit Generation, [6] is published by NIST, (National Institute of Standards and Technology) in the US and is, together with SP800-90A [7] and C [8], the requirements for random bit generators. This report focuses only on

SP800-90B because that recommendation regards entropy sources, while A are requirements for deterministic random bit generators and C for constructions.

In NISTs publications a RBG (random bit generator) is divided into DRBG (Deterministic RBG) and NDRBG (Non-deterministic RBG), however no recommendations are given for a basic NDRBG, an entropy source without a deterministic algorithm.

This leads to some of the requirements that are given depending on which algorithm is used, this section will try to focus on just the entropy source.

The main difference between NIST and AIS31 is that SP800-90B focuses on entropy sources used as input to a DRBG, which can be compared to a PTG.3 in AIS31. However if an entropy source fulfils the requirements for AIS31 it can be used as input to a DRBG and still pass the evaluation.

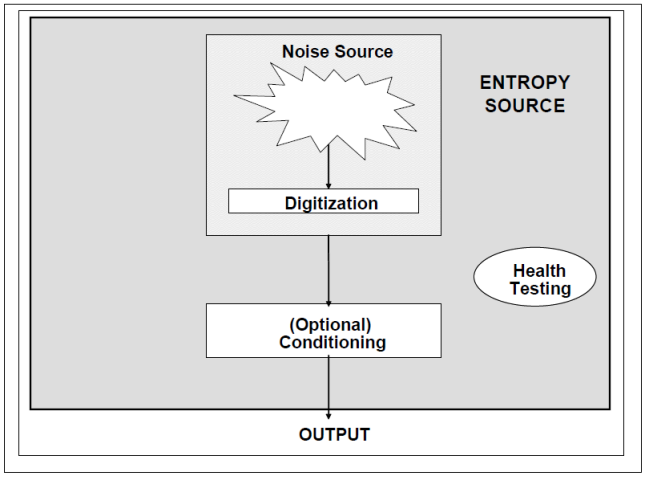

Figure 2.2: Entropy Source Model

2.3.1 Requirements

Figure 2.2 shows a model over the entropy source which shall fulfil the following requirements :

N1 The entropy source should contain the different parts; a noise source, digitization mechanism, health test and optional conditioning, according to Figure 2.2.

Health test contains three tests; start-up, continuous and on-demand, which shall be implemented. This component shall fulfil the following requirements:

N2 The health test shall make sure that the entropy source works as expected.

N3 If an error occurs at the health test no more random numbers can be given and the module must be aware of the error.

N4 Start-up tests shall be performed at start up and the test sequence shall pass before any random numbers can be at the output.

N5 The continuous test detects all known noise source failure modes and runs while the RBG is operating.

N6 No specified requirements are given on the on-demand test, but it must be possible to perform such a test.

The requirements define two tests that shall be implemented in both the start-up and continuous tests. Alternatively other tests may be implemented, if proof is given that they test the same thing.

N7 Repetition Count Test:
This test shall detect if the noise source is stuck on one value. It shall use the estimated min-entropy and calculate the probability for repeating values.

N8 Adaptive Proportion Test:
This test shall detect a large loss of entropy, which can occur after a physical failure or environmental change that affects the noise source.

The estimated min-entropy (see section 8.2) in requirement N9 depends on which DRBG mechanism, and its security strength is used at the output. This gives another requirement:

N9 The requested amount of entropy shall be met.

Other given requirements on the noise source are :

N10 The random numbers at the output do not need to be unbiased and independent, but they shall show probabilistic behaviour.

N11 It shall be possible to test data directly from the noise source.

2.3.2 How does SP800-90 corresponds to FIPS140

The Federal Information Processing Standards Publication(FIPS) PUB 140-3 (draft), [10] and previous versions are security requirements for the whole cryptographic modules, while SP800-9B are recommendations for the entropy source. In FIPS 140 four security levels are specified where the security increases until level 4, which is the highest.

Requirements in FIPS140 for a true random generator(called Non Deterministic Random Number Generator in the newest versions of FIPS140) are specified in [10] section 4.7.1 and includes the self-tests(specified in section 4.9) and the conditional Continuous Random Number Generator(specified in implementation Guidance(FIPS140-2IG) [11] in section 9.8). It also tells that the TRNG shall be identified in the security policy and a description, including all entropy sources, should be in the test report. For FIPS140-3 a documentation of the minimum entropy and the generated method of this entropy is required. If the security levels 3 and 4 are wanted the module has to have an error log which should provide information of the most recent errors.

FIPS 140-2 Section 4.9: The tests Power-up and Conditional self-tests shall be performed and if a self-test fails the module shall enter an error state. The documentation shall include proof of this together with what actions that have to be done to exit the error state.

FIPS140-2IG Section 9.8 The continuous random number test shall be passed and the conditional test shall be performed when specified conditions occurs for the tests, such as: pair-wise consistency test, software/firmware load test, manual key entry test, continuous random number generator test, and bypass test.

To summarise this more requirements for the entropy source are given:

- (N12) A documentation of the minimum entropy and the generated method must be included.
- (N13) Information of the most recent errors must be saved and possible to collect.

2.3.3 NIST test suite

The National Institute of Standards and Technology(NIST) have developed the test suite, SP800-22 [12], which includes 15 tests and even though they are not a part of the requirements for FIPS validation, it's used to verify the randomness of the output.

1. The frequency(Monobit) Test
   Tests the proportion of zeroes and ones in the sequence. The proportion should be about the same.
2. Frequency Test within a Block
   Tests the proportion of ones in a block of size M-bit. For a true random sequence the frequency of ones is approximately M/2.
3. The Runs Test
   Tests the total number of runs in the sequence. A run is an sequence of identical bits, which starts and ends with the opposite value.
4. Tests for the Longest-Run-of-Ones in a Block
   Tests the longest run of ones in a block of size M-bit.
5. The Binary Matrix Rank Test
   Determines the rank of a M x Q matrix, where M=Q=32 in the tests, and compares with statistical properties.
6. The Discrete Fourier Transform (Spectral) Test
   Test the peak heights in the Discrete Fourier Transform of the sequence. A periodic pattern in the test sequence is detected that can indicate a deviation from the assumption of randomness. The goal is to see whether the number of peaks exceeding the 95% threshold is significantly different than 5 %.

7. The Non-overlapping Template Matching Test
   This is a test to detect the number of occurrences of a given non-periodic pattern. To search for a specific m-bit pattern a m-bit window is used. The window slides one bit position if the pattern not found, otherwise the window is reset to the bit after the found pattern and continues the search.
8. The Overlapping Template Matching Test
   A m-bit window is used here as well for the non-overlapping template matching test and the idea with this test is to determine the number of runs of ones for a given length and see how it differentiates from the expected number.
9. Maurer's "Universal Statistical" Test
   The goal is to determine the number of bits between matching patterns.
10. The Linear Complexity Test
    Longer linear feedback shift register (LFSR) can represent random sequences and the idea with this test is to determine the length of a LFSR.
11. The Serial Test.
    This tests the existence of 2^m m-bit overlapping patterns and makes sure that it is approximately the same as for a random sequence.
12. The Approximate Entropy Test
    This tests the frequency of overlapping blocks of two close lengths(m and m+1) and compares it with the result for a random sequence.
13. The Cumulative Sums (Cusums) Test
    The digits in the random sequence are adjusted to $(-1, +1)$ and a random walk is performed, defined as the cumulative sum for the adjusted digits. This random walk should be near zero for a random sequence.
14. The Random Excursions Test
    In a cumulative sum random walk, the number of cycles having exactly K visits are compared to what's expected for a random sequence.
15. The Random Excursion Variant Test
    In a cumulative sum random walk the number of times a particular state occurs are compared to what's expected.

2.4 ISO/IEC 18031

Another standard is the international standard ISO(the International Organization for Standardization)/IEC(the International Electrotechnical Commission) 18031,[14], published 2011. This standard includes requirements for both non-deterministic and deterministic random number generators, as well as hybrid RNGs. A generator is non-deterministic if the entropy source is non-deterministic, which is a system where any amount of entropy can be extracted by sampling. Further on, a non-deterministic generator can be divided into physical and non-physical. A physical non-deterministic RNG (same as TRNG earlier) uses dedicated hardware to produce the random numbers. An example of a non-physical RNG can be time between keys pressed. As for the other requirements the focus will be on entropy sources used in physical non-deterministic RNGs.

2.4.1 Requirements

To be able to use the entropy source as a random number generator the following requirements have to be fulfilled.

I1 The principles behind the entropy source shall be well-established, which makes it possible to identify a statistical model.

I2 The bits shall show probabilistic behaviour.

I3 Bits shall be stationary.

I4 Bits do not need to be unbiased and independent.

I5 A total failure of the entropy source shall be detected.

I6 Other failures or severe degradation shall be detected.

I7 The health tests shall be done at initialisation, at periodic intervals during operating and on demand.

I8 If a failure occurs the source shall stop generating random numbers and it shall enter an error state.

I9 In the error state it shall be possible to collect information about the error.

I10 The health tests shall contain statistical tests that correspond to a stochastic model.

As requirements I10 tells the health tests shall be based on a stochastic model, but it can be hard to find such a test. If this is the problem it is possible that instead of finding tests that determine if the data fall into an acceptable range the test can determine if the data have any existence of values that are known to be associated with failures. The health test is used to make it possible to detect manipulations and influences by some unauthorised external body.

2.5 Comparison of requirements from AIS31 and SP800-90B

This section compares the requirements from AIS31 and SP800-90B, which are the standards that are mandatory in Germany and the US. Both methods use the raw binary signal(digitized signal) for testing, as well as the post-processed output. This is done because serious defects in the noise generator can be masked by the post processing and then pass the statistical tests.

A summary of the requirements is given in table 2.1 for AIS31 and SP800-90B, with comments and if the requirement direct or indirect is important for the entropy source. Most of the requirements are more or less the same and are put on the same line. Requirement A1/A9 and N8 are similar, with the difference that the requirements from AIS31 are much stricter. N8 can only detect a catastrophic failure of the noise source.

In AIS31 it is required to have a stochastic model for estimation of the entropy and statistical tests are made, both on the output of the TRNG and the raw binary signal (A15, A16, A20) This is not a part of the SP800-90B.

The requirements pose no contradictions.

SP800-90B	AIS31	Comments	Relevant for entropy source
N1	A12	Start up, continuous and on-demand for NIST. Online, total failure and start up for AIS.	
N2		Test the entropy source	X
N3	A2	If an error occurs, the output stops	
N4	A4	Must pass start up test.	X
N5	A3	In AIS31, detect non-tolerable statistical defects. Test during operation to detect all known failure modes.	X
N6	A7	In AIS31 online test can be triggered at specific events. On-demand test	
N7		Detect if noise source is stuck on one value	X
N8	A9/A1	Test to detect a large loss of entropy.	X
N9		Requested entropy shall be met	X
N10	A13	The output does not need to be unbiased and independent, e.g. a entropy defect.	X
N11		The raw data bits shall be possible to test.	X
N12	A11	Shannon entropy at least 0.997. Documentation of the min-entropy and generated method.	X
N13		Save information of most recent errors.	X
	A5	Non-tolerable weaknesses should be detect soon.	X
	A6	Test quality of the raw random sequence	X
	A8	Detect defects on raw random numbers quickly.	X
	A10	Internal numbers do not differ from the output from an ideal RNG.	
	A14/A18/A20	Proof of that post-processing does not reduce the average entropy per bit.	
	A15	Stochastic model should provide evidence that the entropy is large enough.	
	A16	A stochastic model should show that the raw random numbers are stationary distributed and no significant dependencies occurs.	
	A17/A19	Raw and internal random numbers shall pass statistical tests.	X

Table 2.1: Comparison of requirements

A summary of the requirements needed for a true random generator according to the requirements above:

1. Use an uncontrollable physical process as source.
2. It must be possible to use the raw binary signal
3. You need a stochastic model of the entropy source(or documentation). The stochastic model should estimate the distribution of the raw random numbers for focus on the entropy source and are used to estimate the entropy of the sequence and evaluate what can affect the quality of the raw random number sequence. (A5,A6,A8,A10)
4. If the entropy per bit are not large enough a post-processing algorithm is used.
5. Internal total failure test
6. Tests of non-tolerable statistical defects
7. Online test on raw binary signal
8. Statistical tests T0-T5 in test procedure A are passed under all relevant environmental conditions
9. Statistical tests T6-T8 on raw binary signal in test procedure B are passed under all relevant environmental conditions
10. Verify that the entropy per internal random number is sufficiently large.

Chapter 3

Implementation of true random number generators in FPGAs

In [15] the authors introduces a concept, the (absolute) inner testable generator, that is important when comparing different implementations. This means that it should be possible to evaluate the entropy of the digitized noise, which can be compared to the requirements where it should be possible to test the raw random numbers. If the generator is absolutely inner testable it is sure that the output does not contain any pseudo-random patterns. When implementing a random number generator in a FPGA (Field Programmable Gate Array), we are restricted to available resources, such as LUT (Look up tables) and PLLs (Phase-locked loop). Two different phenomena are used: the variation of the delay of logic gates(transition jitter), the analogue behaviour between two logic levels (metastability).

3.1 Different implementations of TRNGs

While searching for an entropy source that could pass the evaluation criteria mentioned in section 2.5, some different designs of random number generators were studied. This section gives a brief overview of the designs that were studied. The advantages and drawbacks of each are mentioned together with some short information about the source of randomness.

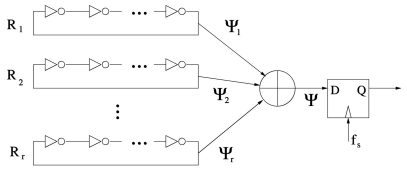

Figure 3.1: Sunar et al's Ring Oscillator, with the design given in [16]

- Ring Oscillator by Sunar et al's [16]
  Consists of n ring oscillators with 13 inverters, as in Figure 3.1 and the phenomena used is jitter. It is implemented with resources that are available in

all FPGAs. The biggest drawback of this generator is that the mathematical proof given in [16] is said to be unrealistic, which makes it hard to pass the evaluation criteria given by AIS31 [1]. Otherwise it is said to produce random numbers with a constant relatively high output bit-rate, according to [16].

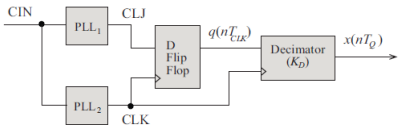

Figure 3.2: Fischer and Drutarovsky PLL RNG, given in [18]

- PLL generator by Fischer and Drutarovsky
  Presented in [17] 2002, the randomness is given by the jitter in the clock signal synthesized in an embedded analog PLL. The jitter is detected by the sampling of a reference (clock)signal using a correlated (clock) signal synthesized in the PLL. The generator doesn't use a post-processing algorithm, it already has a stochastic model, uses few resources and can fit all FPGAs, even though all does not have analogue PLLs. According to the publication the generator passes NIST test suite [12]. Design according to Figure 3.2. A possible drawback is that it requires two PLLs, which can be a large part of the resources in some FPGAs.

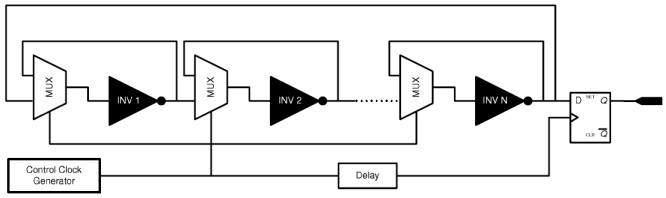

Figure 3.3: Architecture of Vasyltsoc et al's generator, given in [19]

- Vasyltsov et al's generator
  In [19] Vasyltsov et al's present a random number generator which consists of a ring oscillator which has two modes; metastability [1] and oscillation. The design is totally digital and it can be possible to achieve a high constant bit-rate. But in their paper, [19] there are details that show that the bit-rate can change with time and/or temperature. The paper gives no implementation but the authors claim that it passes FIPS 140 and AIS31 class P1(similar to PRNG.1) tests, but not AIS31 class P2(similar to PRNG.2).
- Danger et al's
  Proposed in [20]. It consists of a delay line which is composed of a chain of

[1]For more information about metastability see: http://en.wikipedia.org/wiki/Metastability_in_electronics (24 june 2013

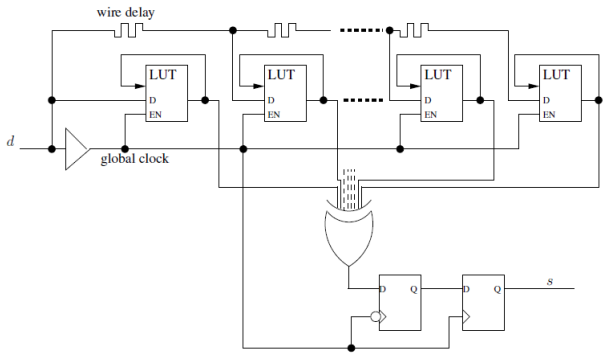

Figure 3.4: Danger et al's generator from [20]

n delay elements. Can be problematic if implemented in an FPGA, if the delay elements do not have sufficiently small delay. This generator requires a post-processing algorithm, but it is absolute inner testable. There is no stochastic model of the generator but it can be derived quite easily, according to the authors of [15].

Chapter 4

Transition Effect Ring Oscillator

The Transition Effect Ring Oscillator, TERO, is proposed by Varchola and Drutarovsky first in [21] and later in the book [22]. It was chosen to be implemented and tested with regards to the requirements in section 2.5. TERO was chosen mostly because of three reasons; it was claimed to be able to pass all tests in FIPS140-2, according to authors of [22], it is small (uses only 6 gates) and because it is a relatively new publication. A stochastic model is also given in [22] that describes the entropy source and can fulfil the requirements in AIS31.

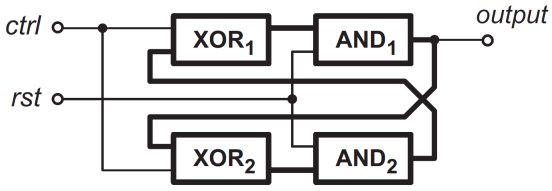

Figure 4.1: First published version of TERO

There are various versions of TERO, the first one proposed in [24] consists of two XORs and two ANDs, according to Figure 4.1. The loop has two control signals, for starting and reseting. The correct place and routing for this TERO is important to ensure the same length of the interconnections between XORs.

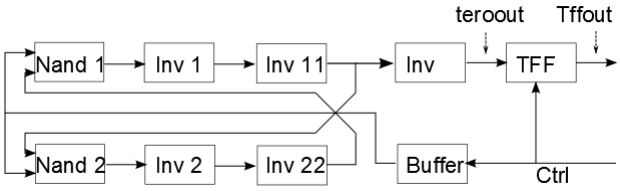

Figure 4.2: Block diagram over TERO loop with NAND gates

Because of the importance of routing in the first TERO, a simpler TERO loop was proposed in [23] where the XOR and AND are merged into NANDs as in Figure 4.2. Inverters are added in the feedback loop. This TERO only has one

control signal, ctrl, which decides the state of operation, either reset or oscillation phase. The reset phase, which occurs when ctrl is '0', resets the loop, to the same starting conditions before generating each random bit. The same starting conditions are important when generating random bits to make sure that the bits are not correlated to each other.

The oscillation phases start when ctrl changes from '0' to '1'. This transition will make the loop start to oscillate and keep oscillating for a random time.

Figure 4.3: Wave diagrams over TERO loop

Figure 4.3 shows the output from the TERO loop, teroout as in Figure 4.2. The pulse length of the oscillations will shorten because of timing asymmetry (different rise and fall time for each gate) according to Figure 4.3 and the oscillations will eventual stop, when the pulse length is smaller than the reject time in transistors. As seen in Figure 4.2 a TFF (T-Flip Flop), which is triggered at the falling edge of teroout is used and its output, tffout, is used as input to an asynchronous counter. The asynchronous counter counts the number of oscillations and the random bit will be the least significant bit, LSB, of the counter. A synchronous counter, which is the most common counter, cannot be used because it is too slow for this frequency of oscillations.

The randomness comes from jitter in the gates, which will affect the pulse length which results in a random number of oscillations.

Chapter 5

Implementation of a Transition Effect Ring Oscillator

5.1 Tools and Hardware

A TERO was implemented in an ORSoc OpenRISC Development board, ordb2a-ep4ce22 [27] with an Altera Cyclone IV FPGA. Altera Quartus II 12.1 [26] was used to develop the design and together with a VirtualBox image [25] the board was programmed. To simulate the implementation Modelsim Altera 10.1b was used.

Three individuals were used for implementation.
Some key features of ORSoc OpenRISC Development board:

- Altera Cyclone IV E, 22K LUT
- 4 general purpose PLLs
- 50.000 MHz clock from RMII
- 4 channels USB UART, FTDI FT4232
- Power supply over USB
- 594 kbits embedded memory
- 22320 Logic Elements

5.2 Details on the design

The whole block diagram of the implementation can be seen in Figure 5.1 and consists of four larger blocks. One for the entropy source, two for the UARTs and one for communication and control between the two others, called output control. The module output control includes one counter, two finite state machine and one module for statistics of the results from the asynchronous counter.

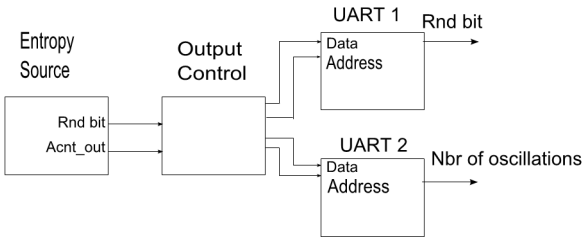

Figure 5.1: Block diagram for design

5.2.1 Entropy source

The entropy source module includes the noise source, the TERO loop and an asynchronous counter. The only two input signals are Reset and a 50 MHz clock. The clock is only used as input in a PLL for producing a control signal with a frequency of 1 MHz, which controls the rest of the logic. The outputs of the module are the random bit and the count value from the asynchronous counter.

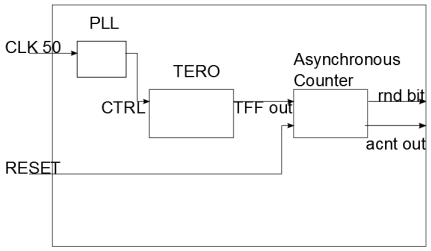

Figure 5.2: Block diagram of Entropy source module

5.2.2 Output control

The output control module is the interface between the entropy source and the UARTs. The input signals are reset, a 50 MHz clock, the random bit and count value, the last two from the entropy source. The four output signals are two data signals and those addressing both UARTs. According to Figure 5.3 the module includes a counter, a statistics module (TERO stat) and two finite state machines (FSM). The statistics module makes a histogram of the number of oscillations.

5.2.3 UART 16550

To be able to communicate with the board, two UART 16550 (Universal Asynchronous Receiver Transmitter) were implemented on the board, with a core from opencores [28]. UART is used for serial communication, sending in this block, one start bit, 8 data bits and one stop bit. The baud rate, same as the data rate for binary data, is 115200 bps and no parity bits are used. The baud rate has to be programmed with a baud rate divisor value using two registers, DLL and DLM

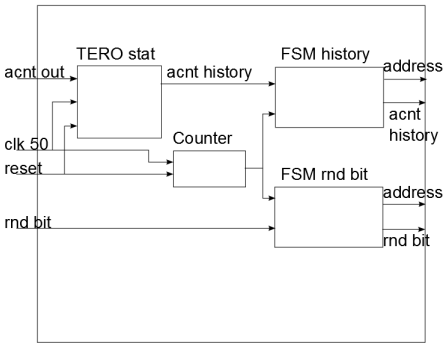

Figure 5.3: Blockdiagram of Output control

(divisor latch register), L respective M stands for LSB and MSB (most significant bit).

UART 16550 can be programmed with baud rates up to 115200 bps, which is the rate used in this design.

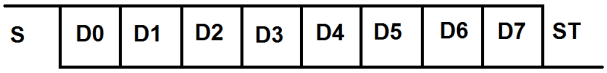

Figure 5.4: Data bits for transmitting UART

All settings and more information about UART 16550 can be found in [29].

In this implementation one UART was used to output the random numbers and the other for test information, such as number of oscillations(from asynchronous counter) and alarm signals. Both UARTs only have one transmitter part implemented, so it is only possible for the user to receive data from the board, not the other way around.

Chapter 6

Results from the first implementations of TERO

6.1 Placement and Routing

The first tries implementing the RNG was done with no constraints or assignments on placement and routing in Quartus II. While evaluating these results it was easy to see that the number of oscillations and the entropy differed a lot between placements. To be able to know that the placement and routing were the same in every implementation the TERO loop, according to Figure 4.2, was hand-placed as in Figure 6.1. The figure shows a LAB, configurable logic block, which consists of several logic units. Each unit looks like Figure 6.3, which in this case shows how a nand gate is implemented in one logic cell.

That the result could differ between implementations was a problem and the hand-placed loop was placed in a LogicLock region in Quartus II [1] and then as design partition[2] to keep the same placement and routing, when changing the rest of the design. The routing can be see in Figure 6.2, where both the local and global routing is shown together with how the cells are connected. An important lesson learned in finding good placements was that a more symmetric TERO loop gave more oscillations which means that more jitter is accumulated.

[1] Logic Lock Quartus, locks region for routing and placement

[2] Design partition in Quartus II allows the logic to be synthesised and placed separately

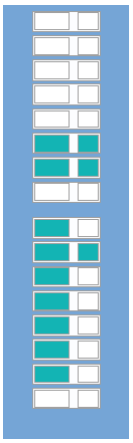

Figure 6.1: Placement of the TERO

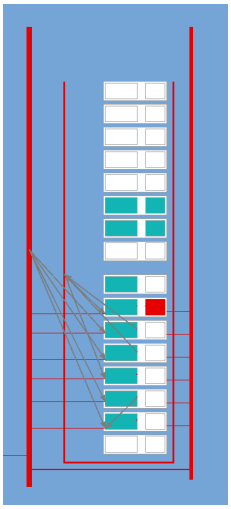

Figure 6.2: Routing of the TERO in one LAB

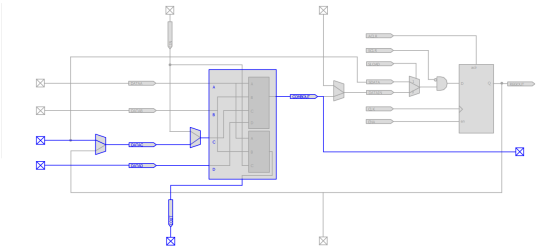

Figure 6.3: Nand gate implemented in LUT

Chapter 7

Timing Simulations

A couple of different simulations were performed in Modelsim ALTERA 10.1b [31], all with the goal of trying to understand the behaviour of the TERO loop. Quartus II was used to create timing files for simulation, by the function EDA Netlist Writer, which gave three [1] files with different parameters. The generated pairs are .vho [2] and .sdo [3] files. The three different files are: 6_0c_slow, 6_85c_slow and min_0c_fast. In the file name 6 respective min stands for the speed grade of the device, 0c and 85c are the junction temperature and slow respective fast is for slow or fast corners. These files correspond to the corner cases for a voltage of 1200 mV and a surrounding temperature of 25 ° C. To get the simulation more like reality in hardware a normal distribution jitter, VHDL code [30], was implemented to effect all rising and falling edges in the TERO loop. This means that the simulation will show the expected behaviour in the FPGA.

7.1 Impact of placement and routing

To get the best performance when implementing TERO in hardware the basic idea is to only let the jitter affect the randomness, which leads us to determine the importance of the routing and placement on gates. In this part the loop was divided into the upper and the lower part, as in Figure 7.1.

If $T_{p1} = T_{p2}$, the loop is symmetric and the TERO will not stop oscillating until a falling edge of the ctrl signal appears, which initiates reset phase. The result of this simulation is shown in Figure 7.2.

Keeping the same total delay through the loop, simulations are done for $T_{p1} \leq T_{p2}$ and for $T_{p2} \leq T_{p1}$.

7.2 Impact of jitter

To get an idea of what size of jitter is needed to fulfil the entropy requirements N12/A11, from the requirement in section 2.5, simulations were performed for

[1] uart 6 1200mV 0C slow, uart 6 1200mV 85C slow and uart min 1200mV 0C fast
[2] VHDL Output File is a standard netlist file, compiled from Quartus II
[3] Standard Delay Format Output file

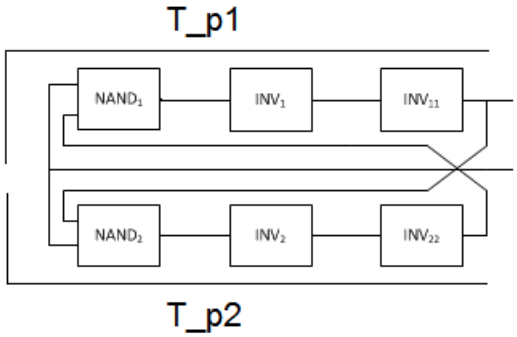

Figure 7.1: Loop divided into upper and lower part and showing the corresponding delay

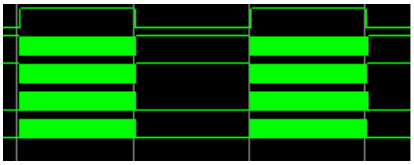

Figure 7.2: Wave diagrams of the symmetric TERO loop

different sizes of jitter. Starting with a jitter of size 0 ps (equals the standard deviation σ in $\mathcal{N}(0, \sigma^2)$, gave no oscillation at all and a Shannon entropy of 0.

sigma [ps]	0c slow	85c slow	0c fast
0	0	0	0
0.5	0.9999	0.9995	0.9387
1	0.9999	0.9999	0.9918
2	0.9999	0.9999	0.9999

Table 7.1: Entropy for different jitter sizes

sigma [ps]	0c slow	85c slow	0c fast
0	51	28	13
0.5	45-58	26-32	11-16
1	41-62	24-33	11-16
2	35-68	23-34	9-18

Table 7.2: Number of oscillation for different jitter sizes

While sweeping the size of jitter for the three different timing files it can be seen, according to table 7.1, that all files fulfil the entropy criteria of 0.997, as the requirement for AIS31 A11, at a jitter of size 2 ps. The entropy increases with a

higher jitter, as seen in table 7.1. In [22] it was proven that the TERO was effected by a jitter as small as 0.5 ps. But with regards to the entropy criteria in AIS31 this cannot be confirmed for this implementation of TERO.

The authors of [22] assumed the jitter in hardware to be around $7ps$. Even though this TERO is implemented in a different family of FPGAs the jitter is assumed to be in that region even for this project. The results of the timing simulations for sigma = 7 can be shown in Figure 7.3, 7.4 and 7.5

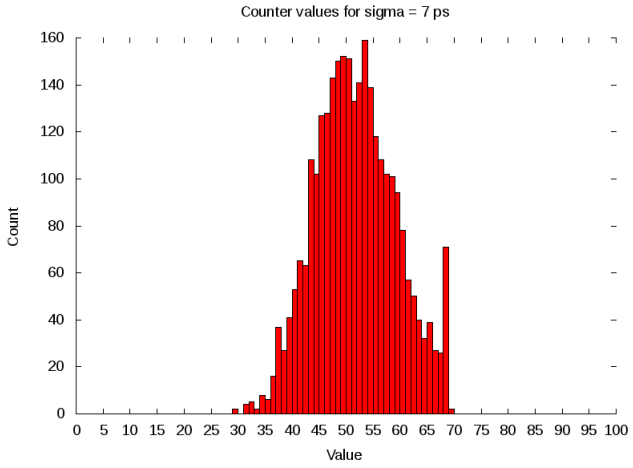

Figure 7.3: Number of oscillations for 0c slow with sigma 7 ps.

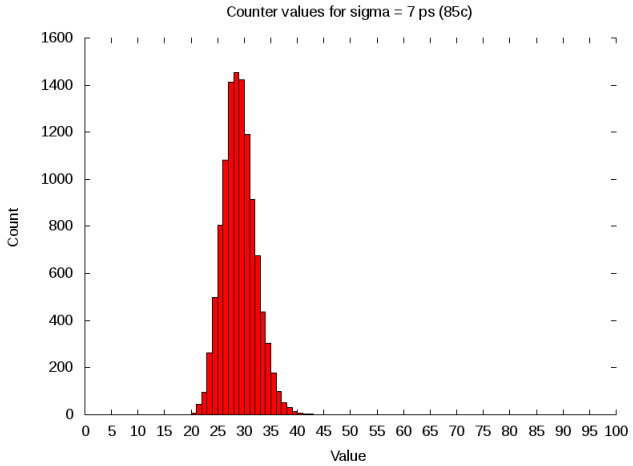

Figure 7.4: Number of oscillations for 85c slow with sigma 7 ps.

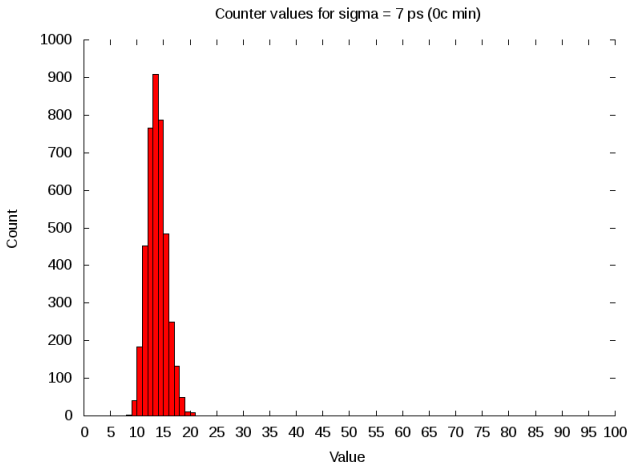

Figure 7.5: Number of oscillations for 0c fast with sigma 7 ps.

Chapter 8

How does TERO fulfil the requirements

To be able to see if TERO would be able to meet the requirements summarised in section 2.5, the requirements are divided into three major parts, statistical tests, entropy and embedded tests and stochastic model.

The TERO loop described earlier was implemented on three FPGAs of the same sort, named A, B and C , to be able to see the possible difference and if the TERO behaves differently.

8.1 Statistical test suites

One important part of the requirements from AIS31 and an extra part in NIST 900-80B was the statistical test suite [12]. The mandatory statistical tests are described in section 2.2.3, and the random bits from the TERO loop were tested. Both the random data from FPGA A and C were tested by the statistical test suite from AIS31. The bits from FPGA B was not tested because of very bad entropy. Test T0 to T5 was tested 257 times on the data and part of the results, from FPGA A, can be studied in Figure 8.1 and- 8.2. The results are in german but it can be seen that the data from FPGA A pass tests T0 to T7 and that T8 is not passed. For more information about the result, compare these results to [1] and [2].

8.2 Requirements for entropy

Shannon entropy is a requirement in AIS31 and this entropy are calculated from :

$$H(x) = -\sum_{\omega \in \Omega} P(X = \omega) log_2(P(X = \omega)).$$

Where $P(X = \omega)$ is the probability. In AIS31 the entropy requirements are passed if the Shannon entropy > 0.997.

For NIST the requirements for entropy depend on the security degree of the deterministic algorithm afterwards. NIST uses min entropy, which is defined as:

$$H(x) = -log_2(max(p_i)).$$

Where p_i is the probability. The min entropy is stricter than the Shannon entropy.

```
Test T0 bestanden.
Starte Test T1 (Monobittest);  Kriterium P1.i(ii)
Anzahl Einsen: 10136
Zulässiger Bereich: [9655; 10345]
Test T1 bestanden.
Starte Test T2 (Pokertest);  Kriterium P1.i(ii)
Testgroesse = 17.779199999999946
Test T2 bestanden.
Starte Test T3 (Runtest);  Kriterium P1.i(ii)
0-Runs[1] = 2569; zulässiges Intervall: [2267; 2733]
1-Runs[1] = 2474; zulässiges Intervall: [2267; 2733]
0-Runs[2] = 1168; zulässiges Intervall: [1079; 1421]
1-Runs[2] = 1198; zulässiges Intervall: [1079; 1421]
0-Runs[3] = 606; zulässiges Intervall: [502; 748]
1-Runs[3] = 625; zulässiges Intervall: [502; 748]
0-Runs[4] = 311; zulässiges Intervall: [233; 402]
1-Runs[4] = 330; zulässiges Intervall: [233; 402]
0-Runs[5] = 167; zulässiges Intervall: [90; 223]
1-Runs[5] = 168; zulässiges Intervall: [90; 223]
0-Runs[6] = 152; zulässiges Intervall: [90; 223]
1-Runs[6] = 178; zulässiges Intervall: [90; 223]
Test T3 bestanden.
Starte Test T4 (Long Runtest);  Kriterium P1.i(ii)
[Test T4 bestanden.
Starte Test T5 (Autokorrelationstest);  Kriterium P1.i(ii)
Maximale Z_tau-Abweichung von 2500: 125
Aufgetreten für Shifts:
Shift: 2722
 Nochmaliger Autokorrelationstest mit Shift: 2722 auf Bits 10.000 bis 14.999
Z_2722 = 2443
Test T5 bestanden.
```

Figure 8.1: Result from T0 to T5 from BSI testsuite for FPGA A

```
Testprozedur T6a zur Verifikation von P2.i)(vii.a) gestartet.
|P(1) - 0.5| = 0.0025299999999999767
Letztes Element: 100000
Testprozedur T6a bestanden.
Testprozedur T6b zur Verifikation von P2.i)(vii.b) gestartet.
p(01) = 0.49844
p(11) = 0.49529
|p_(01) - p_(11)| = 0.003149999999999986
Letztes Element: 502962
Testprozedur T6b bestanden.
Testprozedur T7a zur Verifikation von P2.i)(vii.c) gestartet.
Testgröße[0] = 4.23251586580575
Testgröße[1] = 4.087121078020387
Letztes Element: 1742247
Testprozedur T7a bestanden.
Testprozedur T7b zur Verifikation von P2.i)(vii.d) gestartet.
Testgröße[0] = 0.4322866901274989
Testgröße[1] = 13.520034719166786
Testgröße[2] = 2.9349575075935084
Testgröße[3] = 1.321330633714418
Letztes Element: 5144875
Testprozedur T7b bestanden.
Test T8 zur Verifikation von P2.i)(vii.e) gestartet.
Testgröße = 7.930555465410079
Letztes Element: 7213355
Test T8 zur Verifikation von P2.i)(vii.e) nicht bestanden.
Test wurden nicht (vollständig) durchgeführt ODER NICHT BESTANDEN.
```

Figure 8.2: Result from T6 to T8 from BSI testsuite for FPGA A

One easy way to get an approximate value of the Shannon entropy is to use Linux command function ent. Otherwise as told in section 2.2.3 this requirement is met if T8 is passed in test suite from BSI. The results from the Linux command ent can be studied in table **??**. It can also be said that because FPGA A passed test T8 in AIS31 the entropy will be large enough.

A	B	C
0.9930	0.0	0.6432

Table 8.1: Table over entropy for A, B and C in bits per bit

8.3 Embedded tests and stochastic model

The part regarding the stochastic model and self test were the major part of the requirements, especially for AIS31. A stochastic model was given in [22] that described the function of TERO as:

$$T_S - T_M = \sum_{i=1}^{Y_{T_j}} (T_D + \Delta T_{ij}) = T_D Y_{Tj} + \sum_{i=1}^{Y_{Tj}} \Delta_{Tij}.$$

T_S and T_M are the times for a logical one for the first respective the last oscillation, T_D is the time the pulse shortens each oscillation. Δ_{Tij} is the periodic jitter which follows $\mathcal{N}(0, \sigma^2)$. Y_{Tj} is the number of oscillations.

Figure 8.3: Part of wave diagrams of the TERO oscillation

Figure 8.3 shows the out signal from the TERO loop and it is possible to see the shortening of the logic '1'. This stochastic model is supposed to be used for online testing and to be able to understand how the test should be performed, all statistics behind the entropy source must be understood. Part of this is made in section 7 and the goal is to add an online test based on number of oscillations. This will be one easy way to test the entropy source in an acceptable time, as the requirements dictate. An acceptable bound for the number of oscillations is given by the three different timings in section 7 and the online test shall, according to the simulations in this report, fail if the numbers of oscillations are outside this boundary.

So according to these simulations, assuming a jitter of 7 ps the boundary shall be 8 to 69 number of oscillations. This will just be an easy online test giving an approximate result.

The results from the implementations on FPGAs were as said before inconclusive so the numbers of oscillations differed. For FPGA A, that passed the statistical tests the oscillation was 20 to more than 75. For FPGA B, that did not give any entropy the numbers of oscillations were 83 to 87 and finally FPGA C had all its oscillation in the interval 20 to 35.

A	B	C
20-75+	83-87	20-35

Table 8.2: Number of oscillation for FPGA A,B and C

8.4 Discussion of the results

In table 8.3 a summary of how the requirements given in section 2.5 are fulfilled. It can either be fulfilled by what the authors of [22] claim or by theoretical, simulated or experimental means. Regarding the stochastic model it is only given by the authors of [22]. A total failure test has not been discussed so much earlier but it shall detect a total failure of the noise source. This can be done easily because when a total failure occurs the number of oscillations will be zero. The requirement regarding the online test is fulfilled by the authors of [22] but cannot be confirmed by the results given in this report. The last requirement, the entropy, can be said to be fulfilled if the jitter in hardware is large enough.

		Claimed by authors	**Theoretical**	**Simulated**	**Reality**
Uncontrollable source	YES	X	X	N/A	N/A
Use raw signal	YES	X	X	X	X
Stochastic model	YES	X		N/A	N/A
Total failure test	YES		X		
Online test	YES?	X			
Pass statistical tests	NO	X	X		
Entropy large enough	YES?	X		X	

Table 8.3: Table over entropy for A, B and C in bits per bit

As mention earlier the simulation results differed from the result from the implementation. This discussion tries to explain what can be the problem.

If the stochastic model given by the authors of [22] is assumed to be true, the parameters that are unknown are the timing and the size of the jitter. The timing for this implementation is given by Quartus II and if this is assumed to be true the problem must be the jitter. The simulations described earlier showed the expected behaviour for different jitter sizes. This leads to the conclusion that if reality does not correspond to the simulations, the jitter must be too small. The jitter size cannot be affected in such a way that it can be used in generation of random bits, so then this design cannot be used.

But if the timing is assumed to be incorrect or that it can differ a lot from the timing files from Quartus II, the jitter can have a size close to what is given in this report. If this is the case it is possible to change the timing of the loop, by trying to find a better placement.

Chapter 9

Conclusions and future work

While comparing the mandatory requirements, AIS31 and SP800-90B it was interesting to see their different approaches to evaluating an entropy source. In AIS31 it was possible to use the entropy source alone for generation while in SP800-90B it had to be used together with an approved deterministic algorithm to generate random numbers. The largest difference between the methods is the stochastic model in AIS31, where it is used to prove statistics of the random numbers. Another difference was also that AIS31 had statistical tests and no such test was mandatory for SP800-90B, even though the organisation NIST has another publication for the statistical tests. Even though there is a lot of difference between the two methods, they both had the same basic idea with embedded tests and to access the raw random numbers. It can also be said that SP800-90B is just a draft and contains questions regarding basic NRBGs (without a DRBG) and how to assure a good output. This means that NIST can add information and requirements for this later on. Even though ISO/IEC 18031 was not a large part of the requirements in this report it is important to know that there exist other standards. While comparing this standard to the other two it seems like it is closer to AIS31 than SP800-90B, because of the importance of a stochastic model and embedded tests based on the model.

The simulations sweeping the jitter and calculating the entropy from it, showed that a jitter as small as 2 ps gave an entropy large enough to pass the requirements. And if the jitter in reality is assumed to be around 7, according to authors of [22], it is shown that this implementation shall have an entropy large enough.

So while summing up the result of the implementation of TERO shows that it is supposed to give an high entropy, according to the timing simulations. But it is hard in reality to get a proper result and it can change a lot depending on FPGA and probably external disturbances.

The result of these tests and simulations showed that it was not easy to take this particular design and make it pass the evaluation schemes. The hardest part was the stochastic model and the online test. The problem was that it took a lot of time trying to understand the performance, which probably would be easier if a more experienced, both with implementations and statistics, person did it. But according to the simulations and result in this report it was shown that TERO was dependent on a lot of parameters to make it work properly. Both that the results differed while changing the placement and also that the three implementations on FPGAs gave such a difference. Why the result from the implementations was

so different was hard to know, but it can be because of external disturbance, as difference in power supply or other small differences on the FPGAs.

9.1 Future work

There is still some parts left to be able to take an entropy source from academic literature and implement it so it passes the requirements from BSI and NIST. A more proper online test has to be implemented, with more studies of the boundary. Especially the lower bound has to be studied more, to get information on how many oscillations are needed to get an entropy large enough. More things that have to be studied in the future are:

- Understand why the implementation does not correspond to the simulated behaviour.
- Understand the complete behaviour of the entropy source, to be able to understand the importance of, for example, placement and routing.
- Implement TERO in such a way that it passes the statistical test suites in AIS31 and NIST 800-22.
- Invent and implement a proper online test.

References

[1] Killmann, W., Schindler, W.: *A proposal for: Functionality classes for random number generators, version 2.0.* Tech. rep., Bundesamt fur Sicherheit in der Informationstechnik (BSI), Bonn (September 2011), https://www.bsi.bund.de/EN/ Home/home_node.html

[2] Killmann, W., Schindler, W.: *A proposal for: Functionality classes and evaluation methodology for true (physical) random number generators, version 3.1* Tech. rep., Bundesamt fur Sicherheit in der Informationstechnik (BSI), Bonn (25 september 2001), https://www.bsi.bund.de/EN/ Home/home_node.html

[3] Schindler, W.: *Functionality Classes and Evaluation Methodology for Deterministic Random Number Generators* Tech. rep., Bundesamt fur Sicherheit in der Informationstechnik (BSI), Bonn Version 2.0 (2 December 2011), https://www.bsi.bund.de/EN/ Home/home_node.html

[4] Schindler, W. : *A Stochastic Model and Its Analysis for a Physical Random Number Generator Presented At CHES 2002* In Cryptography and Coding. Lecture Notes in Computer Science Volume 2898, 2003. Pages 276-289. Springer Berlin Heidelberg.

[5] Schindler, W. , Killmann W. *Evaluation Criteria for True (Physical) Random Number Generators Used in Cryptographic Applications* In Cryptographich Hardware and Embedded Systems-CHES 2002. Lecture Notes in Computer Science Volume 2523, 2003. Pages 431-449. Springer Berlin Heidelberg

[6] Barker, E. , Kelsey, J.: *Recommendation for the Entropy Sources Used for Random Bit Generation,* NIST DRAFT Special Publications 800-90B(August 2012) http://csrc.nist.gov/publications/PubsSPs.html

[7] Barker, E. , Kelsey, J.: *Recommendation for Random Number Generation Using Deterministic Random Bit Generator* NIST Special Publications 800-90A (January 2012) http://csrc.nist.gov/publications/PubsSPs.html

[8] Barker, E. , Kelsey, J.: *Recommendation for Random Bit Generator (RBG) Constructions* NIST DRAFT Special Publications 800-90C (August 2012) http://csrc.nist.gov/publications/PubsSPs.html

[9] NIST *Federal Information Processing Standards publication* FIPS PUB 140-2, (May 25 2001) http://csrc.nist.gov/publications/PubsFIPS.html

[10] NIST *Federal Information Processing Standards publication* FIPS PUB 140-3(Revised DRAFT 09/11/09) http://csrc.nist.gov/publications/PubsFIPS.html

[11] National Institute of Standards and Technology Communications Security Establishment Canada *Implementation Guideance for FIPS PUB 140-2 and the cryptographic Module Validation Program*, (December 21, 2012) http://csrc.nist.gov/groups/STM/cmvp/index.html

[12] A Rukhin et al. *Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications* NIST Special Publication 800-22 revision 1a,(April 2010). http://csrc.nist.gov/publications/PubsSPs.html

[13] W. Schindler. *Efficient Online Tests for True Random Generators* In C.K. Koc et al. (Eds.) Cryptographic Hardware and Embedded Systems 2001, LNCS 2162, pages 103117, Berlin, Germany, 2001. Springer Verlag.

[14] International Standard *ISO/IEC 18031:2011(E)* Information technology-Security technique- Random bit generation. Second edition 2011-11-15

[15] Fischer, V., Aubert, A., Bernard, F. , Valtchanov, B., Danger, J.L. and Bochard N. *True Random Number Generators in Configurable Logic Devices* Project ANR_ICTeR, version 1.02. February 12, 2009. Online, http://www.lirmm.fr/ w3mic/ANR/PDF/D2.pdf (2013-01-23)

[16] Sunar, B, Martin W.J, Stinson D.R : *A Provably Secure True Random Number Generator with Built-In Tolerance to Active Effect* Proc. of IEEE Transaction on Computers, 2007.

[17] Fischer, V., Drutarovsky, M.: *True Random Number Generator Embedded in Reconfigurable Hardware* In Cryptographich Hardware and Embedded Systems-CHES 2002. Lecture Notes in Computer Science Volume 2523, 2003. Pages 415-430. Springer Berlin Heidelberg

[18] Simka, M., Fischer, V., Drutarovsky, M. : *Testing of PLL-based True Random Number Generator in Changing Working Conditions* In RADIOENGINEERING, 20 (2011). Pages 94-101.

[19] Vasyltsov Ihor , Hambardzumyan Eduard, Kim Young-Sik, Karpinskyy Bohdan : *Fast Digital TRNG based on Metastable Ring Oscillator* In Cryptographich Hardware and Embedded Systems-CHES 2008. Lecture Notes in Computer Science Volume 5154, 2008. Pages 164-180. Springer Berlin Heidelberg

[20] Danger, Jean-Luc , Guilley, Sylvain and Hoogvorst, Philippe *Fast True Random Generator in FPGAs* In Circuit and Systems, 2007. NEWCAS 2007. IEEE Northeast Workshop on. Conference 5-8 Aug. 2007. Pages 506-509.

[21] Varchola, M,, Drutarovsky, M. : *New FPGA based TRNG Principle Using Transition Effect with Built-In Malfunction Detection* Processings of the 7th International Workshop on Cryptographic Architectures Embedded in Reconfigurable Devices-CrytArchi 2009, Prague, Czech Republic, June 24-27, 2009 pages 150-155.

[22] Varchola, M., Drutarovsky, M.: *Cryptographic True Random Number Generator with Malfunction Detector-Mathematical Model of the Noise Source, Synthesis and Testing in FPGAs, and Built-In Malfunction Detector Architecture* LAP LAMBERT Academic Publishing 2011

[23] Varchola, M., Drutarovsky, M. : *Analysis of Randomness Sources in Transition Effect Ring Oscillator Based TRNG* In Proceedings of 8th International Workshop on Cryptographic Architectures Embedded in Reconfigurable Devices (CrytArchi), Gif sur Yvette-Paris, France, June 27-30, 2010, pages 102-107.

[24] Varchola, M., Drutarovsky, M. : *New High Entropy Element for FPGA based True Random Number Generators* In Proceedings of 12th International Workshop on Cryptographic Hardware and Embedded Systems (CHES), Santa Barbara, CA, USA, August 17-20,2010, Springer,2010, pages 351-365.

[25] OpenCores *OpenRISC Ubuntu VirtualBox image details* Online. Available at: http://opencores.org/or1k/Ubuntu_VirtualBox-image_updates_and_information

[26] Altera *Altera Quarus II download* Online. Available at: https://www.altera.com/download/software/quartus-ii-we

[27] OpenCores *The NEW OpenRISC System-on-chip FPGA development board* Online. Available at: http://opencores.org/or1k/Ordb2a-ep4ce22

[28] OpenCores *UART 16550 core :: Overview* Online. Available at: http://opencores.org/project,uart1655

[29] National Semiconductor *PC16550D Universal Asynchronous Receiver/Transmitter with FIFOs* Online. Available at: http://www.ti.com/lit/ds/symlink/pc16550d.pdf

[30] Gnanasekaran Swaminathan *Random Number Generator, package RNG. VHDL* Online. Available at: http://read.pudn.com/downloads111/sourcecode/asm/459855/random1.vhd__.htm

[31] ModelSim Altera *Modelsim Altera Starter Edition Software 10.1b Download* Online. Available at : http://www.altera.com/products/software/quartus-ii/modelsim/qts-modelsim-index.html

Appendix A

VHDL Code

```
-------------------------------------------------------------------
---
--- VHDL code for TERO module
--- Emma Hilmersson, 2013
---
-------------------------------------------------------------------
library ieee;
library altera;
library wysiwyg;
use ieee.std_logic_1164.all;
use wysiwyg.cycloneive_components.all;
USE ALTERA.ALTERA_PRIMITIVES_COMPONENTS.ALL;


entity tero  is
port(ctrl : in std_logic;
tffout : out std_logic
);

end entity;

architecture struc of tero is
signal  teroout, nand1_out, nand2_out, inv11_out : std_logic;
signal inv111_out, inv22_out, inv222_out : std_logic;
signal teroout_i, teroout_o : std_logic;
signal ctrl_i : std_logic;

attribute keep: boolean;
attribute keep of ctrl_i: signal is true;
begin

ctrl_i <= ctrl;

nand1_gate : cycloneive_lcell_comb
```

```
generic map (
dont_touch => "on",
lut_mask => "0011001111111111",
sum_lutc_input => "datac")
port map(datab => inv22_out,
datad => ctrl_i,
combout => nand1_out);

nand2_gate : cycloneive_lcell_comb
generic map (
dont_touch => "on",
lut_mask => "0011001111111111",
sum_lutc_input => "datac")
port map(datab => inv11_out,
datad => ctrl_i,
combout => nand2_out);

inv111_gate : cycloneive_lcell_comb
generic map (
dont_touch => "on",
lut_mask => x"00FF",
sum_lutc_input => "datac")
port map (
datad => nand1_out,
combout => inv111_out);

inv11_gate : cycloneive_lcell_comb
generic map (
dont_touch => "on",
lut_mask => x"00FF",
sum_lutc_input => "datac")
port map (
datad => inv111_out,
combout => inv11_out);

inv222_gate : cycloneive_lcell_comb
generic map (
dont_touch => "on",
lut_mask => x"00FF",
sum_lutc_input => "datac")
port map (
datad => nand2_out,
combout => inv222_out);

inv22_gate : cycloneive_lcell_comb
generic map (
dont_touch => "on",
```

```
lut_mask => x"00FF",
sum_lutc_input => "datac")
port map (
datad => inv222_out,
combout => inv22_out);


inv_tero_gate : cycloneive_lcell_comb
generic map (
dont_touch => "on",
lut_mask => x"00FF",
sum_lutc_input => "datac")
port map (
datad => inv22_out,
combout => teroout);


t11 :entity work.tff1(struc) port map ('1',teroout, ctrl_i, tffout);
end architecture;
```

```
---
--- VHDL code for TFF flipflop
--- Used in TERO and Asynchronous counter
--- Emma Hilmersson, 2013
---
```

```
library ieee;
use ieee.std_logic_1164.all;

entity tff1 is
  port(T       : in  std_logic;
       teroout : in  std_logic;
       ctrl    : in  std_logic;
       Q       : out std_logic
       );
end entity;

architecture struc of tff1 is
  signal q_next, q_reg : std_logic := '0';
begin

  process(teroout, ctrl)
  begin
    if(ctrl = '0') then
      q_next <= '0';
    elsif(falling_edge(teroout)) then
```

```
      if T = '1' then
        q_next <= not q_next;
      end if;
    end if;
  end process;
  Q <= q_next;

end architecture;
```