UNIVERSITY OF CALIFORNIA

Los Angeles

Noise Source Agnostic Entropy Extractor

A thesis submitted in partial satisfaction

of the requirements for the degree Master of Science

in Electrical Engineering

By

Ashutosh Devendra Shiledar

2018

ABSTRACT OF THE THESIS

Noise Source Agnostic Entropy Extractor

By

Ashutosh Devendra Shiledar

Master of Science in Electrical Engineering

University of California, Los Angeles, 2018

Professor Chih-Kong Ken Yang, Chair

As the world becomes more connected and information flows rapidly between the billions of devices, snooping of nodes or security risks of information transfer are increasing. As a result, increasing levels of cryptography are required to keep private data safe and secure. Most cryptographic techniques rely on obtaining random sequences as keys or ciphers to encode and encrypt information. Dynamic environments lead to rapid establishment and teardown of sessions in networked applications, requiring high rates of random sequences, and placing a burden on random number generators (RNG) to deliver these sequences both quickly and securely. This thesis presents a new type of entropy extractor to address these needs. The proposed architecture introduces an architecture that uses a whitening filter in feedback to increase the potential entropy extracted from the noise source. Due to its placement in the loop, the whitening filter is implemented in the digital domain, allowing it to be reconfigured to work with a wide variety of noise sources. This thesis shows how the architecture was developed and analyzes the effects different component choices may have on the entropy extractor's performance.

The thesis of Ashutosh Devendra Shiledar is approved.

Sudhakar Pamarti

Puneet Gupta

Chih-Kong Ken Yang, Committee Chair

University of California, Los Angeles

2018

Table of Contents

Table of Figures

1. Introduction

This thesis presents an architecture for a high-rate entropy extractor for use in True Random Number Generators (TRNGs). The design maintains high entropy by accounting and compensating for any non-white spectral characteristics of the noise source. This chapter briefly describes the potential application and outlines the remainder of this thesis

1.1 Motivation

In recent years, the number of connected devices has increased exponentially. Everything from power meters to cars, and even refrigerators are now collecting its user's data and sending it over the internet. Keeping this data secure is a big concern and is handled using cryptography. Many cryptographic techniques rely on secure encryption keys, which are created using Random Number Generators (RNG). The security of the system depends on the quality of the keys, making the design requirements of the RNG quite stringent. Random number sequences need to meet a number of tests to verify the degree of entropy and sufficient forward/backward secrecy. RNGs are also often embedded as a part of an integrated system and the design must consider and minimize the area and implementation complexity. Furthermore, dynamic environments lead to rapid establishment and teardown of sessions in networked applications, requiring high rates of random sequences, and placing a burden on random number generators (RNG).

1.2 Summary

Chapter 2 gives a brief overview of RNGs, including the different types and the standards associated with each. Chapter 3 covers different architectures of entropy extractors, ending with the architecture proposed by this thesis. Chapter 4 goes in depth into the different aspects to

consider for an implementation, including delays and quantization. Finally, at the end of chapter 4 is a possible implementation for an RNG using the entropy extractor proposed in this thesis.

2. Background

This chapter outlines the basics of RNG evaluation and classification. Section 2.1 covers standards used to approve RNGs for different uses. Section 2.2 goes over the major categories of RNGs, along with their overall design and the associated benefits and drawbacks.

2.1 Evaluation

Random number generators are devices designed to produce a sequence of random outputs. The quality of RNGs are determined by 3 properties: statistical independence, uniform distribution, and unpredictability when under attack [1]. Samples from the RNG output sequence should be statistically independent of each other and show no correlation. All possible output values should be represented uniformly, avoiding bias towards or against any particular values. Finally, RNG outputs should have enough security measures to remain unpredictable to malicious users. These attackers should be unable to guess past or future values, even with knowledge of the design. Although these requirements are easy to understand qualitatively, they are difficult to quantify and measure. Over the years, numerous tests and standards have been introduced to determine the quality of RNGs.

2.1.1 Common Test Suites

Statistical tests have been one of the first methods used to judge the quality of RNGs. No single test is sufficient to determine the randomness of the RNG, so tests are typically grouped in suites for better statistical coverage. Common test suites include DIEHARD, and subsequently DIEHARDER [2], and the National Institute of Standards and Technology's (NIST) 2008 800-22 test suite [3]. To perform the tests, a large amount of raw data must first be collected. Then, the tests are run on the data set to determine various statistical properties. The end result is a pass

or fail for whether the collected sequence was considered random or not. This type of testing works well for checking if RNG outputs are statistically independent and uniformly distributed, but not for ensuring the design is secure from attackers. Statistical tests rely only on testing the output sequences of the RNG, which means a deterministic process can generate sequences that are deemed random. To solve this, newer standards also take the RNG design into account.

2.1.2 AIS31

The AIS31 standard, by Bundesamt für Sicherheit in der Informationstechnik (BSI), splits RNGs into different categories, including Deterministic Random Number Generators (DRG) and Physically True Random Number Generators (PTG) [4]. The categories are further divided into multiple classes, each requiring more cryptographic security than the last. For example, PTG.1 is the lowest class of PTG and only requires the device to include statistical health tests to detect failures. PTG.2 class RNGs must pass PTG.1 requirements and also provide a stochastic model of the noise source. Finally, PTG.3 class RNGs must follow the requirements of PTG.2, and add a cryptographic post-processor. DRG classes also follow a similar structure, mandating additional forward and backward secrecy with each step. DRG.4 provides an interesting requirement for periodic reseeding of the RNG. The seeds can either be provided by the user or a PTG, which consequently behaves as a hybrid that blends a PTG with a DRG as a post processor for generating longer sequences. Categorizing RNGs into classes allows AIS31 to recommend RNGs for particular use cases. PTG.1 and DRG.1 class RNGs can be used in many non-security-based applications, but cryptographic applications require additional safeguards, which can be provided by the higher level classes.

2.1.3 NIST 800-90

As a follow up to the NIST 800-22 statistical test suite, NIST released 800-90 A through C. These three publications provide recommendations for RNG designs. 800-90A focuses primarily on deterministic RNGs, including modelling and usage [5]. The document also contains designs for 4 cryptographically secure RNGs, including two based on hash functions and one based on block ciphers. NIST 800-90B is about entropy source selection and testing, including health tests [6]. The final publication, NIST 800-90C, applies the information from A and B to aid in construction and verification of approved RNGs [7]. In summary, NIST 800-90 was designed to fix the shortcomings of NIST 800-22, which solely relied on statistical testing methods.

2.2 Classification

Random number generators take on various implementations, ranging from simple computer programs to systems that can extract noise from the atmosphere [8]. Due to this large variety, as mentioned in the previous section 2.1 with regard to the various standards, RNGs are generally classified into three categories: Pseudo Random Number Generators (PRNG), True Random Number Generators (TRNG), and Hybrid Random Number Generators (HRNG).

2.2.1 Pseudo Random Number Generators

Common examples of DRG are PRNGs which are based on mathematical algorithms designed to generate statistically random sequences. These types of RNGs can be designed to produce sequences with good statistical properties. They have a stored internal state that gets periodically updated to generate additional values. At startup, the state is given an initial value based on a seed. The seed can be provided by the user, or generated from different system parameters, including time and process ID. PRNGs are deterministic since the outputs depend

only on the internal state and structure. If an attacker acquires either the seed or internal state, and has knowledge of the PRNG structure, the generated sequence becomes compromised as the attacker may be able to predict past or future values.

To mitigate the consequences of a compromised RNG, many standards, including AIS31, require different levels of forward and backward secrecy. Backward secrecy prevents attackers from guessing past values from the knowledge of future values or states. Forward secrecy attempts to stop attackers from guessing future values, given the past values or states. Having these safeguards in place makes cryptographic systems more secure.

2.2.2 True Random Number Generators

Unlike PRNGs, TRNGs incorporate physical entropy sources to generate randomness. These entropy sources include thermal noise, clock jitter, metastability, and many others. The design proposed in [9] utilizes the accumulated jitter from multiple ring oscillators. [10] generates random bits by driving a latch to metastability and allowing the output to resolve. Even spontaneous emissions, occurring in optical fibers have been used by [11] as a source of entropy.

TRNGs tend to be more secure than PRNGs due to the irreproducible nature of the entropy sources; however, this also makes them more difficult to implement and they tend to yield a much lower throughput. Figure 1 shows a generic architecture for a TRNG used in cryptographic applications.

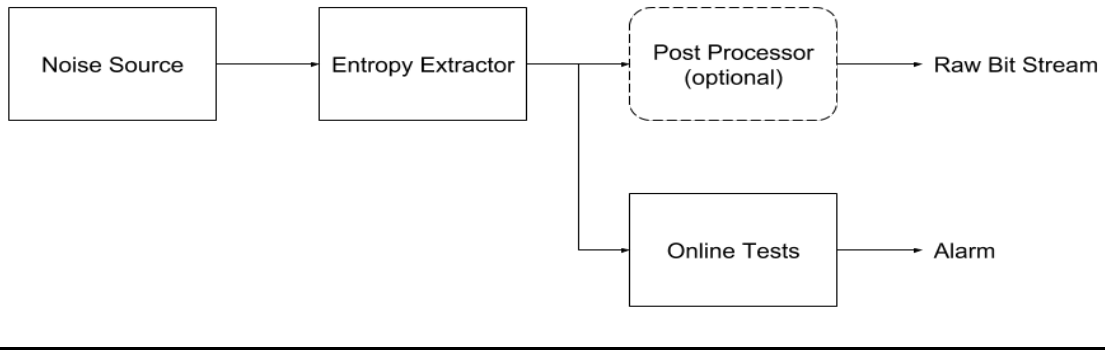


Figure 1: Generic TRNG architecture

The core of the TRNG is the entropy source and the entropy extractor. The entropy source outputs a noise waveform based on its underlying physical processes. The properties of this noise source, such as its spectral shape, varies with the type of source. That waveform is then processed and digitized by the entropy extractor to generate a random bit stream. To improve the statistical properties of the TRNG output, the generated bit stream can be sent to an optional post-processing block for further refinement. A suite of health tests is also included to check the quality of the output bit stream and alert the system in cases of failure. Statistical tests are incorporated as part of the health checks as many can be implemented efficiently in hardware [12].

2.2.3 Hybrid Random Number Generators

HRNGs combine elements of PRNGs and TRNGs, creating a system that is a compromise of the two. In general, TRNGs are more resilient to attacks due to their lack of internal state, but their outputs tend to have inferior statistical properties and slower generation rates. PRNGs, on the other hand, can produce sequences with great statistical properties at a much higher rate. One method of combining these two types of RNGs is by using the TRNG as a seed generator for the PRNG, as suggested by the AIS31 DRG.4 class specification [4]. In 2012, Intel released a new processor architecture incorporating a Digital RNG (DRNG). Their DRNG, shown in figure

2, uses a TRNG to generate a pool of seeds, which are then given to a Cryptographically Secure PRNG (CSPRNG) to generate the output sequences [1].

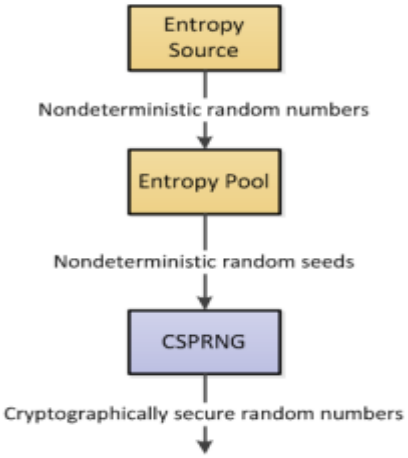


Figure 2: Intel Digital Random Number Generator Overview

The drawback of this, and other HRNG designs, is that the output sequences can still be compromised if either the seed or state of the PRNG is known. In this hybrid system, that risk, while still present, becomes mitigated, as the state is periodically reset to a random value, providing enhanced forward secrecy. As long as the TRNG and memory containing the seed pool are not compromised, the generated random number should be sufficiently safe.

2.3 Summary

This chapter presented an overview of the standards and classifications used to differentiate RNGs based on architecture and performance. This thesis focuses primarily on TRNGs. Specifically, it introduces a general entropy extractor, designed for use with multiple types of noise sources. The next chapter describes various architectures and compares them with simulations leading to the proposed design.

3. Architecture

This chapter introduces entropy extractor architectures for use with different noise sources. Section 3.1 starts with a simple design for use with a white noise source, while Section 3.2 provides enhancements to the design to handle more general shaped noise sources. The chapter ends with a feedback based entropy extractor, which is the proposed architecture with its implementation discussed in detail in chapter 4.

3.1 Additive White Gaussian Noise Source

Additive White Gaussian Noise (AWGN) is widely used in circuit and communication channel models. It is characterized by a flat frequency spectrum and a normal distribution in the time domain. AWGN also has zero autocorrelation, which is ideal for random number generation as neighboring values are statistically independent. Figure 3 shows a simple entropy extractor to generate a random bit stream from the AWGN entropy source.

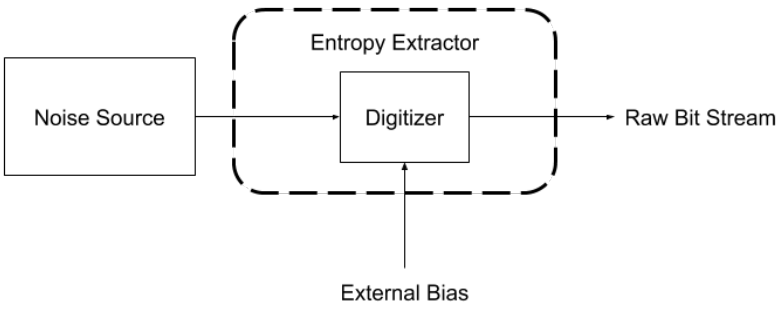


Figure 3: Simple Entropy Extractor with Adjustable Bias

Due to its simplicity, this entropy extractor is one of the most commonly used. The type of noise source determines how the entropy extractor is implemented. For example, with a thermal

noise source, a voltage comparator is used. This has been used by [13] to extract noise from high gain amplifiers. For oscillator phase noise, a phase detector is appropriate. [14] uses simple D flip-flops as binary phase detectors to generate the output bitstream. Assuming the bias for the decision circuit is set correctly, the output bitstream will be unbiased and uncorrelated. This behavior is confirmed through simulations of the entropy extractor in figure 3, where sequences of one million bits, generated from zero mean AWGN signals, passed the NIST 2008 test suite. The sequence size of one million samples was chosen based on the recommendation of the NIST test suite.

3.1.1 Bias Generation

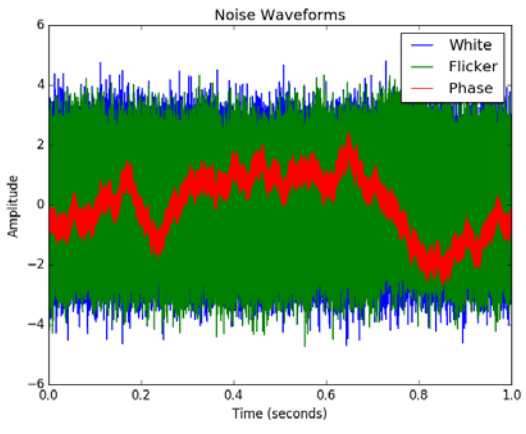
An AWGN signal may have a constant mean, but it will usually be nonzero. The entropy extractor must account for this mean, otherwise the outputs will be skewed, reducing the overall entropy of the RNG. The effects of this skew were examined using another simulation of the entropy extractor in figure 3. The bias was started at mean and swept until the generated bits failed the NIST test suite. The sequences started failing when the bias deviated by 0.7% of the standard deviation, illustrating the importance of correct biasing.

The correct bias can be found by measuring the skew at the output, and feeding it back to the digitizer. Measuring the skew is done by using a low pass filter to extract the DC bias of the output. [13] uses a simple RC low pass filter, while [15] has a counter to accumulate the bits. In digital implementations, the resolution of the device setting the bias must be high enough to produce the value accurately. Using the result from the bias sweeping simulation, the resolution must be less than 1.4% of the noise source's standard deviation.

3.2 Shaped Noise

The majority of noise modelling is done using AWGN, but noise in circuits is known to have different frequency spectra. Flicker noise, also known as $\frac{1}{f}$ and pink noise, is common in resistors and transistors. Compared to AWGN, flicker noise has greater contributions from the lower frequencies, but decrease with a slope of -10 dB per decade, or $\frac{1}{f}$, as the frequency increases. The point where the contribution from the noise floor becomes comparable to the flicker noise is called the noise corner.

Phase noise, often found in oscillators, has a similar shape to flicker noise, except the low frequency slope is -20 dB per decade due to the integration of phase by the oscillator. Sometimes, at lower frequencies, a $\frac{1}{f^3}$ corner becomes prominent due to the inherent $\frac{1}{f}$ noise of the devices within an oscillator, introducing a -30 db per decade slope. Figure 4 shows the frequency spectra of the different noise sources.



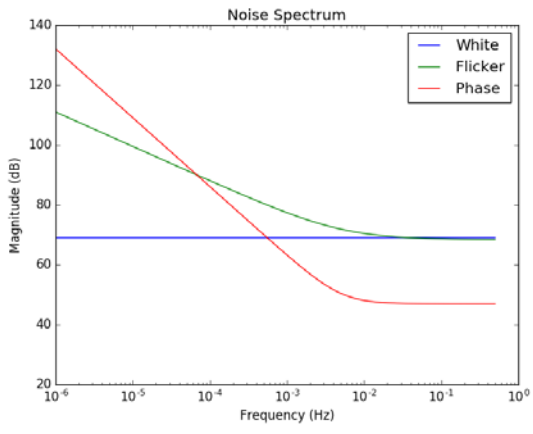


Figure 4: Noise Waveforms and Frequency Spectra

Further shaping of the noise spectral is possible depending on any filtering characteristics of the channel on which the noise is propagated or if the noise is further filtered by other elements such as an amplifier or a phase-locked loop in the phase domain.

3.2.1 Simple AWGN Extractor

Random bits can be extracted from these shaped noise sources with the same entropy extractor used for AWGN in section 3.1. These noise sources are modeled by generating samples of white noise, then imposing the desired shape on the frequency spectrum. In simulations, a sweep of the corner frequencies for flicker and phase noise shows that only very low noise corners, on the order of a thousandth of a percent of the sampling frequency, can be tolerated. With such low corner frequencies, the noise behaves much like AWGN. To deal with shaped noise, most RNG designs turn to post-processors to improve the statistical properties of the bitstream.

Post-processing algorithms compress the input bitstream to increase the entropy per bit at the cost of output bit rate. The design from [14] utilizes an Exclusive-Or post-processing function, which combines 16 cycles of data to generate a single bit output, leading to an efficiency of 6.25%. The post-processor from [15] uses a two-step algorithm, combining their own counter based filtering with a Von Neumann corrector. This design provides an efficiency of 12.5%. Post-processors introduce large penalties in efficiency, adversely affecting the bit rate of the RNG. The following section presents a new entropy extractor that employs feedback to reduce the need for post processing.

3.2.2 Whitening within the Entropy Extractor

A new entropy extractor architecture, proposed in this thesis, is shown in figure 5. It can be considered an extension of the entropy extractor described in section 3.1. The bias, which was

previously generated by an external component, has been brought inside the entropy extractor and is set by a digital-to-analog converter (DAC). A filter takes the output from the digitizer and generates a control signal for the DAC to track the entropy source waveform. The filter is implemented in the digital domain, allowing it to be easily configured and matched to the entropy source provided. In the case of an AWGN source, a simple counter may suffice to set the appropriate bias level for the digitizer, but for other sources, the whitening filter design can be more complex. While whitening filters have been used extensively for signal processing and noise cancellation applications, to the author's knowledge, this is the first time it is evaluated against the entropy test suites for entropy extraction.

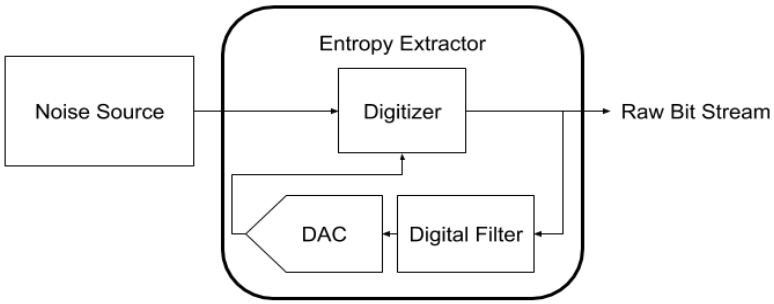


Figure 5: Entropy Extractor Architecture

As a simple example, consider phase noise. Its frequency spectrum can be described by the following equation: $1 + \frac{\omega_c}{\omega}$. The transfer function of the proposed filter, adapted from the AWGN bias circuit is as follows: $\frac{1}{1+F(s)}$, where F(s) is the transfer function of the matched whitening filter in feedback. The output of the comparator is set to $\pm\sigma$, which gives it a gain of 1. Setting the filter equal to the inverse of the phase noise equation gives $F(s) = \frac{\omega_c}{s}$, which means that for phase noise, the whitening filter is a matched integrator. In the digital domain, the integrator can be approximated using various methods such as forward Euler, backward Euler,

and the bilinear transformation [16]. Figure 6 shows a few of these approximations in comparison to the ideal phase noise whitening filter.

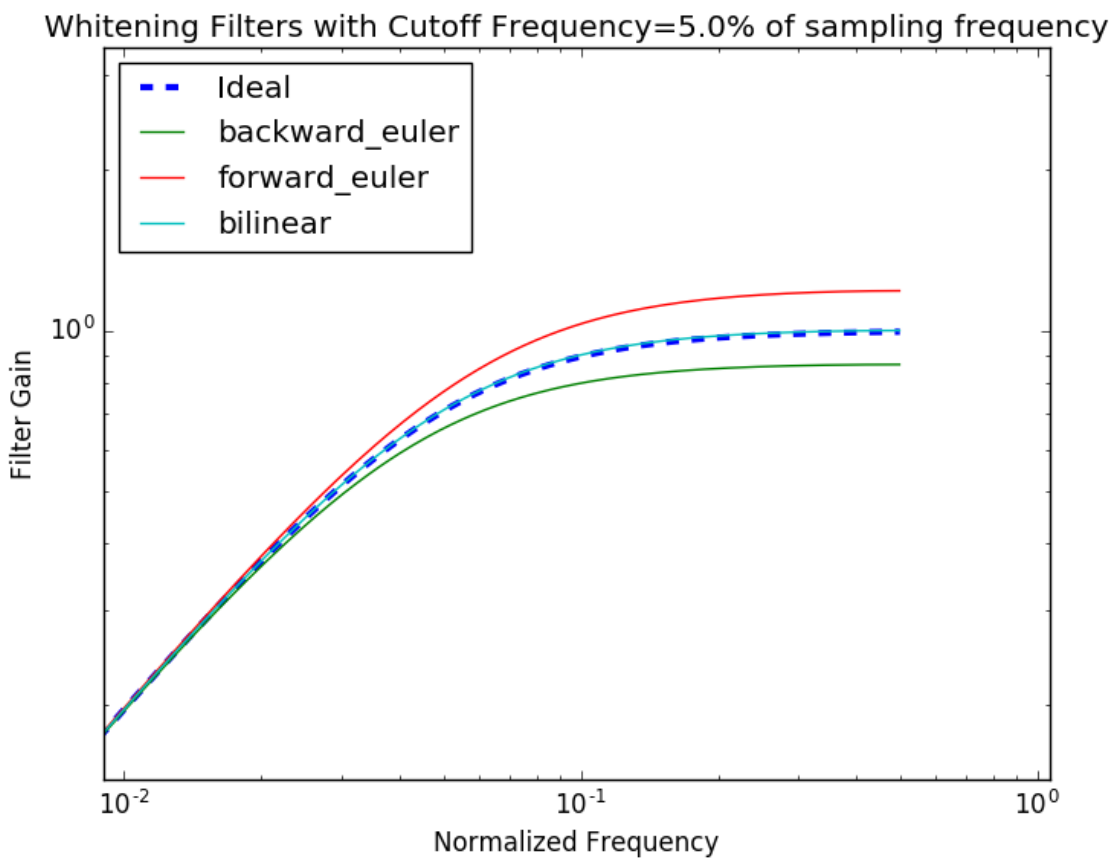


Figure 6: Comparison of Whitening Filters with Different Digital Integrators

For low frequencies, all three approximations match the ideal filter quite closely. Only at the high frequencies do the approximations start to differ. Forward Euler shows high frequency gain, while the backward Euler filter experiences attenuation in the same region. The bilinear transformation stays close to the ideal response throughout the frequency spectrum, making it the best choice of the three for implementation.

This method of filter design is only one of many. Whitening filter design is a vast field with a wide variety of techniques, possibly ones with better characteristics than the one describe above. Those techniques will need to be further studied and evaluated in future works.

3.3 Summary

This chapter presented multiple entropy extractors for use in TRNGs. The final design introduced employs a feedback loop, creating a whitening filter within the entropy extractor itself. Chapter 4 expands on this architecture, providing simulation results to demonstrate its performance.

4. Implementation

This chapter expands on the final entropy extractor architecture proposed in chapter 3 by considering the effects of different components in the design. Section 4.1 shows the effects of digitizer delay on the loop filter characteristics. Section 4.2 demonstrates the performance improvements of increasing digitizer resolution. Section 4.3 examines the effects of quantization in the DAC. Finally, section 4.4 proposes a possible implementation using this architecture.

4.1 Digitizer Delay

The analysis in section 3.2.2 assumes the digitizer has no delay; however, most implementations contain at least a one cycle delay. Delay in a feedback loops introduces overshoot in the response, affecting the filter characteristics. Figure 7 shows the filter characteristics taking the digitizer delay into account.

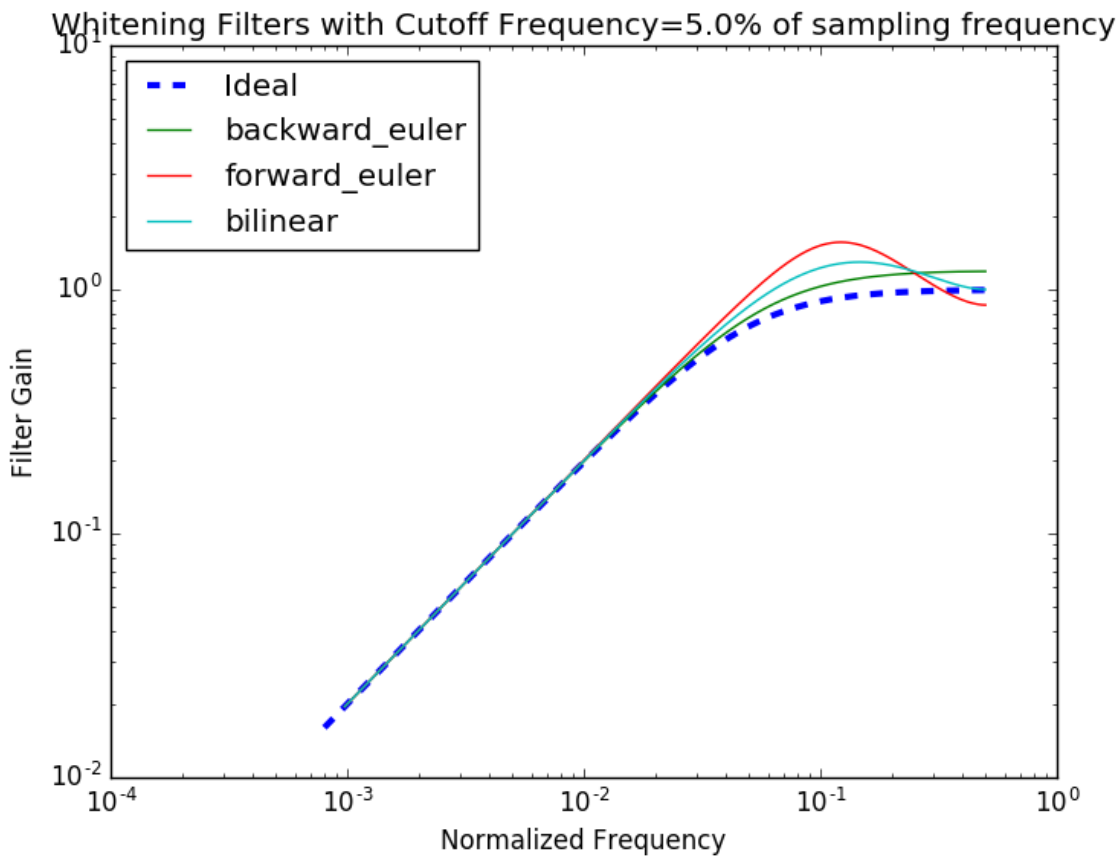


Figure 7: Comparison of Whitening Filters with Different Digital Integrators Accounting for Digitizer Delay

While previously the bilinear integrator has the best response in figure 6, after accounting for the digitizer delay, the backward Euler integrator has a better response. Although it introduces some high frequency gain, the backward Euler integrator has the least amount of peaking.

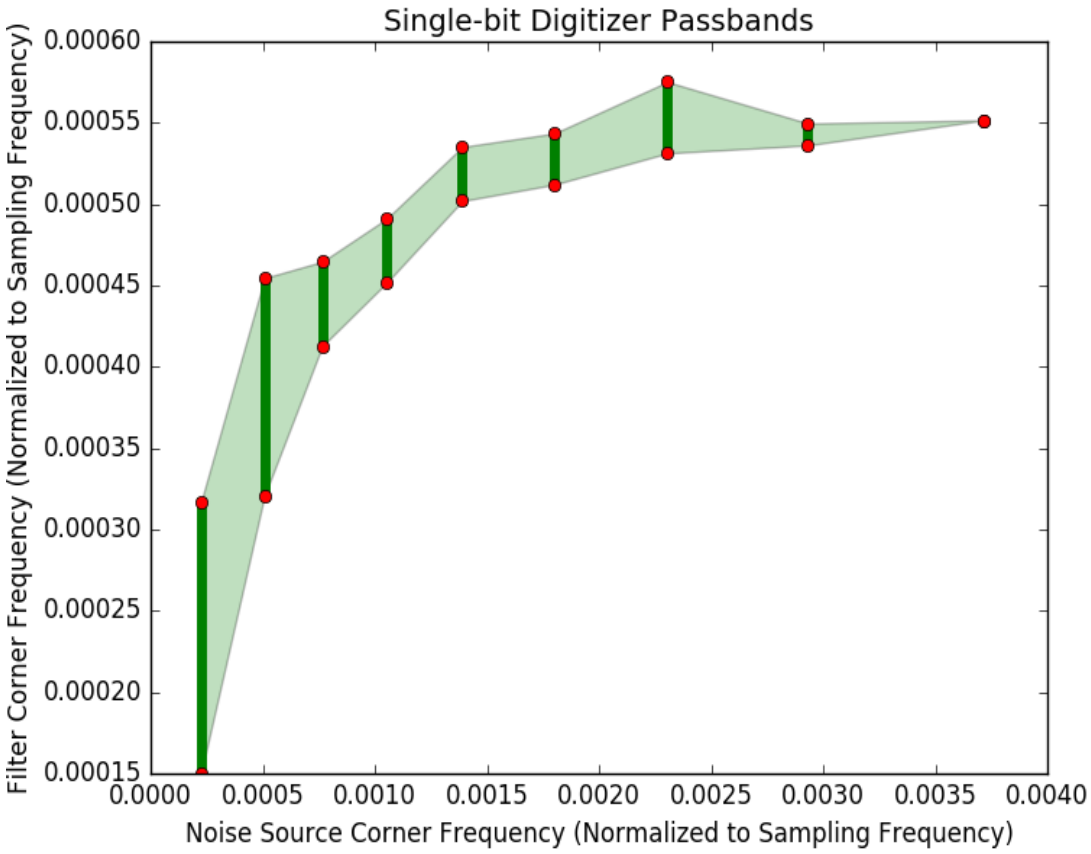


Figure 8: Range of Cutoff Frequencies with Sufficient Entropy using a Single-Bit Digitizer Filter

Figure 8 shows the passbands of an entropy extractor with a single bit digitizer and backward Euler integrator. Lower frequency noise corners have large and robust range of viable filter corners, but as the entropy source noise corners increase, these windows tend to decrease until the window disappears all together. Compared to the simple entropy extractor described in section 3.2.1, this whitening filter improves the acceptable entropy source corner frequency by two orders of magnitude.

4.2 Multi-bit Digitizer

It is evident that the digitizer quantization noise has adversely affected the frequency characteristics of the whitening filter. Due to the feedback loop, the quantization noise ends up back at the input of the digitizer, making the quantization noise correlated [17]. Increasing the resolution of the quantization elements may mitigate this issue. A multi-bit digitizer may seem similar to an analog-to-digital (ADC) converter, but there are a few differences. While both an Analog to Digital Converter (ADC) and multi-bit digitizer must have high resolutions, the digitizer does not require the same range. The tracking nature of the feedback loop reduces the range of inputs the digitizer needs to handle.

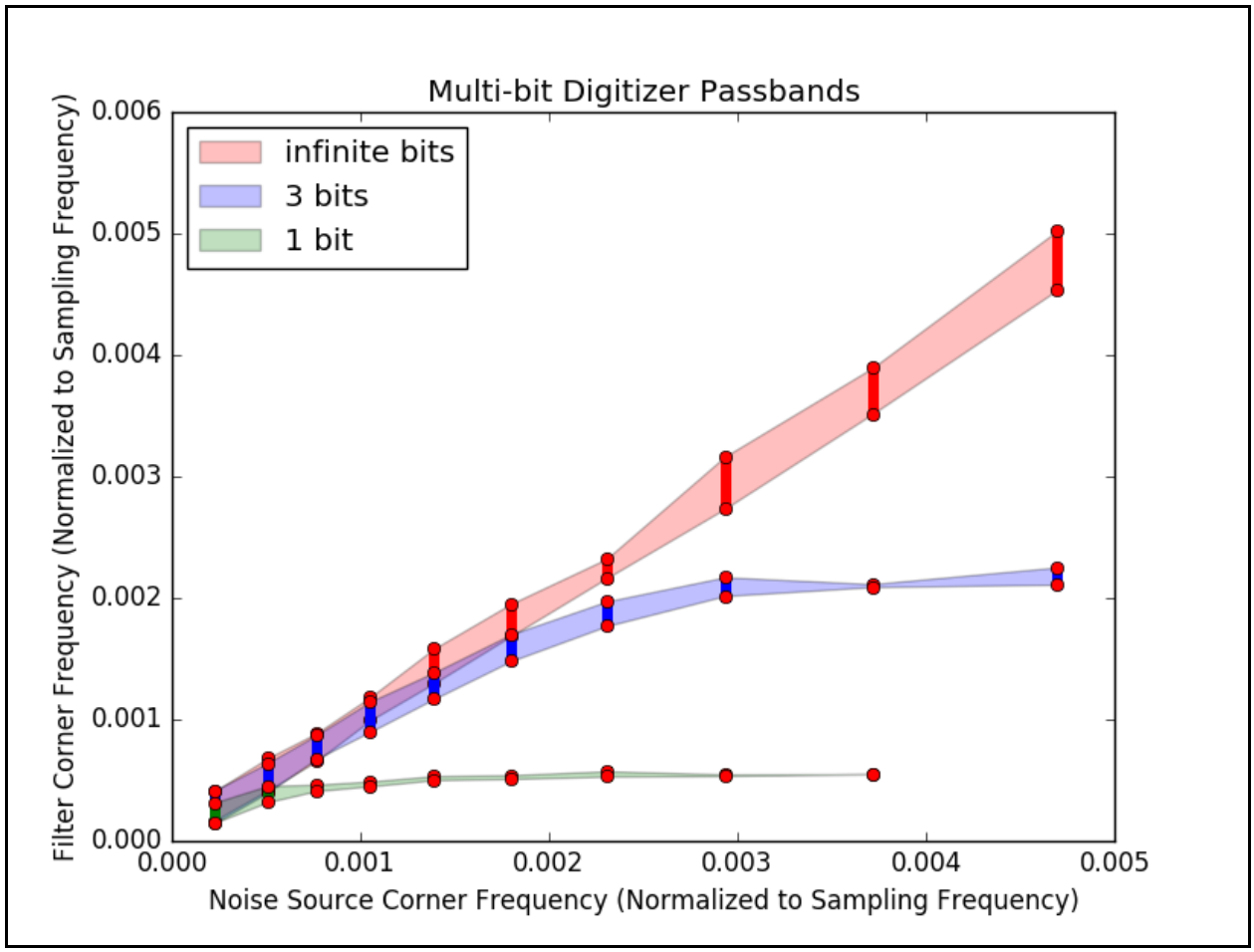


Figure 9: Range of Cutoff Frequencies with Sufficient Entropy using Multi-bit Digitizer Filters

Figure 9 shows the passbands of the entropy extractor for multiple digitizer resolutions. Increasing the resolution of the digitizer leads to a wider range of valid filter cutoff frequencies, which means the entropy extractor is more robust. Compared to the single bit digitizer, the 3 bit digitizer's curve is a similar shape, but starts leveling off at a frequency about 4 times higher. This result matches the expected behavior as the quantization noise is also 4 times lower for the 3 bit digitizer.

4.3 Digital to Analog Converter Quantization

Thus far, the quantization of the DAC has not been taken into account. Previous analysis of AWGN, in section 3.1.1, showed the minimum DAC resolution to be 1.4% of the standard deviation. Simulation results show almost no degradation in performance with a DAC of that resolution. However, creating a DAC with such fine resolution is challenging. Increasing the step size has shown to reduce performance for entropy sources with high corner frequencies.

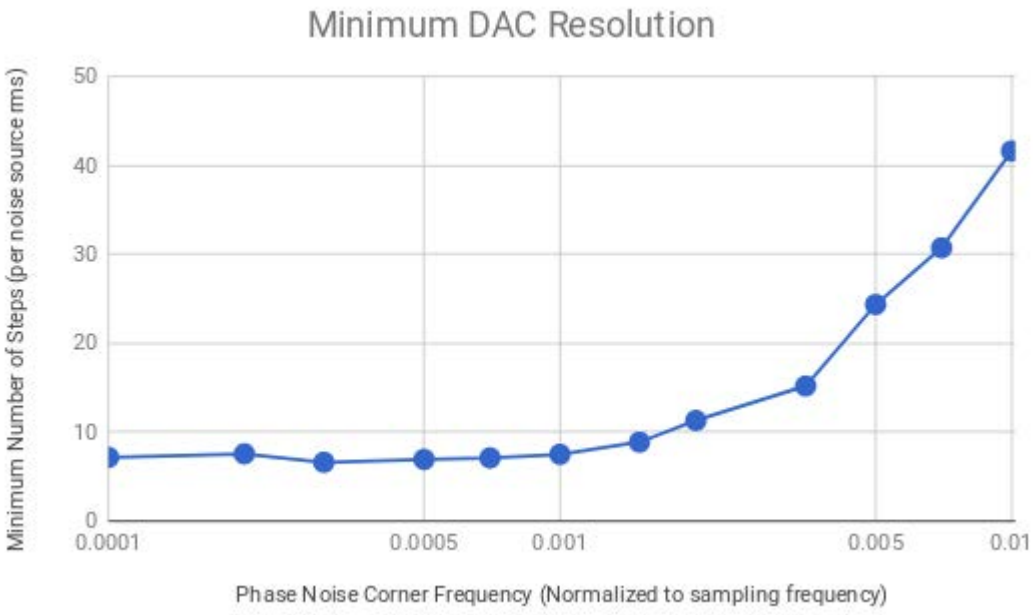


Figure 10: Minimum DAC Resolution for Various Phase Noise Corner Frequencies

Figure 10 shows a tradeoff between DAC resolution and performance at higher noise source corner frequencies. Entropy extractors with lower resolution digitizers inherently have lower performance at the higher noise corner frequencies, due to its own quantization noise. Because the quantization noise from the DAC and digitizer are uncorrelated [17], decreasing the DAC resolution to match the performance dictated by the digitizer resolution leads to the most cost efficient trade off. The data in figure 8 shows that an entropy extractor with a single bit digitizer works with phase noise sources that have corner frequencies below 0.1% of the sampling rate. For phase noise corners in that range, a DAC with 10 steps per standard deviation provides sufficient margin. To cover 6 standard deviations, which the simulations have shown to be the dynamic range, the DAC will have 60 steps, giving it a size of 6 bits.

4.4 Example Implementation

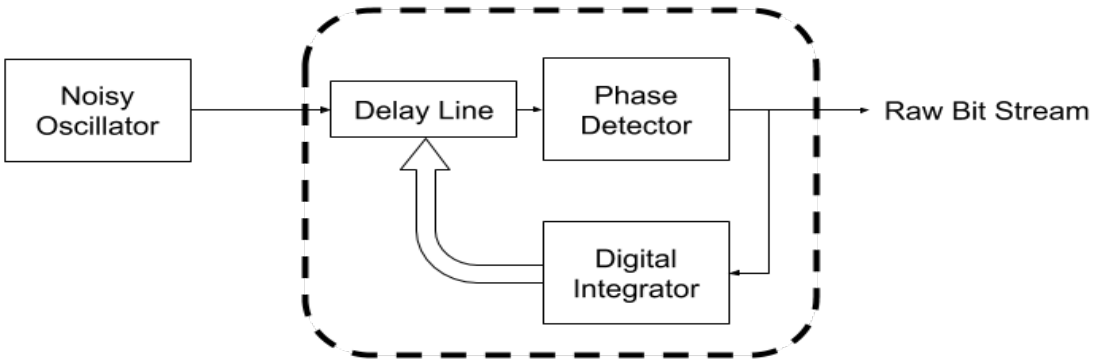


Figure 11: Implementation of Proposed Entropy Extractor

Figure 11 above shows a possible implementation of an RNG using the entropy extractor described in this thesis. This design uses an oscillator phase noise source, which is amenable to implementation on a field programmable gate array (FPGA). FPGAs have many clocking resources to generate various frequencies and phases of clocks. A global clock, generally sourced from a crystal oscillator, is used to generate the rest of the clocks using mixed-mode clock managers (MMCM) on Xilinx FPGAs. Here, an MMCM generated clock can be used as the noise source. The entropy extractor is implemented using 3 components. The phase detector, which uses the clean clock to sample the noise source, is implemented as a register. The random bit stream generated by the phase detector is fed to a digital integrator, which in turn sets the control of the delay line. This delay line will act as the DAC of the entropy extractor.

The Xilinx design tool suite, Vivado, gives a measurement of the peak-to-peak jitter of the generated MMCM clock. Figure 12 shows a snapshot of the clock generation tool and highlights the peak-to-peak jitter value, which is 149 ps. Here, the MMCM is synthesizing a 100 MHz clock from the global 100 MHz clock.

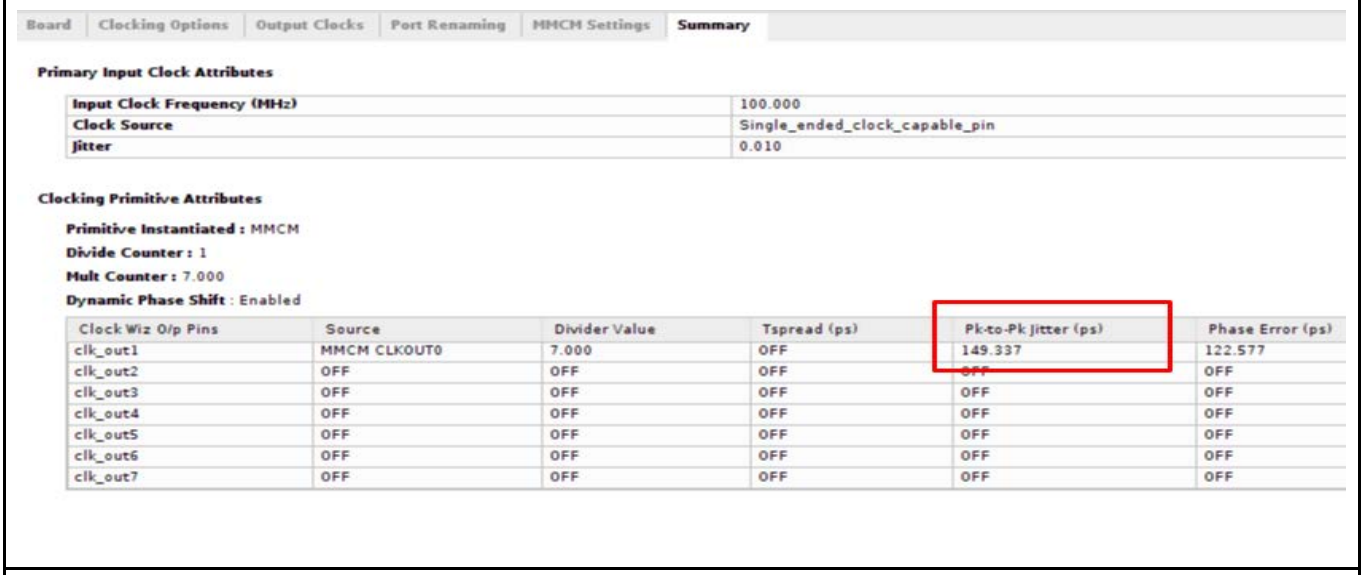


Figure 12: Xilinx MMCM Clock Jitter

According to Xilinx, their peak-to-peak jitter measurement is 14 times the standard deviation, which means a single standard deviation is around 10 ps. From the results in section 4.3, the delay line resolution should be 1/10 of the standard deviation, yielding a step size of 1 ps. Delay lines have been implemented in FPGAs before, as shown in [18] and [19]. Out of these three methods, the highest resolution achieved has been 10 ps. ASIC implementations, such as one presented by [20], can reach sub 1 ps resolutions, but would have to be placed off chip. In addition to the resolution, the range of the delay line must also be taken into account. The simulations run in this thesis have shown the dynamic range of the DAC to lie within 6 standard deviations, giving a full scale range of 60 ps. The delay lines mentioned above all have larger ranges, which can be useful when paired with the MMCM dynamic phase shifting. MMCMs have the ability to shift their output phase by 1/56 of the clock period. For a 100 MHz clock, the increments are 180 ps. Combining dynamic phase shifting and a delay line with a range greater than 180 ps, about 8 bits for a 1 ps resolution, can make a coarse/fine delay line. This technique was used by [15] and [20]. The benefit of using the dynamic phase shifting of the MMCM is that it wraps around, effectively eliminating the dynamic range requirement of the delay line.

The final component in this entropy extractor is the digital integrator. As discussed in section 4.1, the integrator was designed using the backward Euler transformation, to reduce the impact of digitizer delay. Its transfer function is as follows: $F(z) = \frac{\omega_c T}{1-z^{-1}}$, where ω_c is the phase noise source corner frequency in radians/sec and T is the sampling period, a term that comes from the backward euler transformation. Together, $\omega_c T$ forms the gain, which sets the filter cutoff frequency. An alternate form of the gain is $\omega_c T = {}^{2\pi f_c}/_{f_s}$, where f_c is the filter cutoff frequency, and f_s is the sampling frequency. The integrator can be realized using the direct form 1 implementation shown in Figure 13 below [21]. This form was chosen because the gain multiplication is at the front of the filter, operating on the single bit input, rather than the multi bit accumulated value.

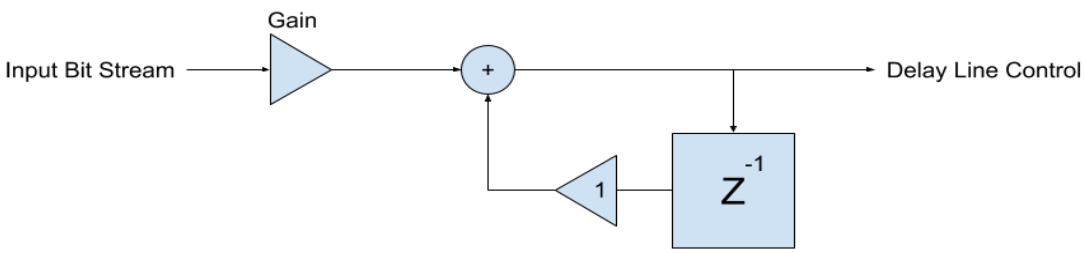


Figure 13: Direct Form 1 IIR representation

The integrator value is chosen to be represented in a fixed point representation. The integer portion of the value represents the delay line control signal, which is 8 bits wide, while the fractional portion represents the accumulated values from the input. The smallest value the integrator needs to represent accurately is the gain. Implementing an integrator that supports arbitrarily low gains is infeasible as it requires an infinite number of fractional bits. An appropriate minimum supported gain must be selected based on the noise source. The MMCM specifications

do not have the phase noise corner frequency, so for this example the integrator design will be done with a minimum supported corner frequency of 10 KHz, or 0.01% of the sampling frequency, the lowest noise corner frequency run in the prior simulations. For this minimum noise corner, the gain is $\omega_c T = {}^{2\pi f_c}/_{f_s} = 0.000628$. However, this number is with respect to the standard deviation. The delay line is designed with a step size of one tenth of the standard deviation, which reduces the required precision of the fractional component by a factor of 10. The data from figure 7 shows the entropy extractor filter cutoff frequency can have an error of 33% for lower noise source corner frequencies and still provide sufficient entropy to pass the NIST test suite. Representing the number 0.00628, which is 10 times the gain associated with the lowest supported cutoff frequency, to within 33% error can be done in 7.9 bits, rounded to 8 for implementation. Looking back at figure 7, the size of the passbands decrease as the noise source corner frequency increases. Eventually, the tolerable error drops to 3% for entropy extractor filter cutoff frequencies at 5 times the minimum gain. Representing 0.0314 with a 3% accuracy requires 9.1 bits. Taking this into account, the total size of the integrator is 18 bits, 8 for the delay line control and 10 for the fractional value. Figure 14 shows how this may be implemented on an FPGA.

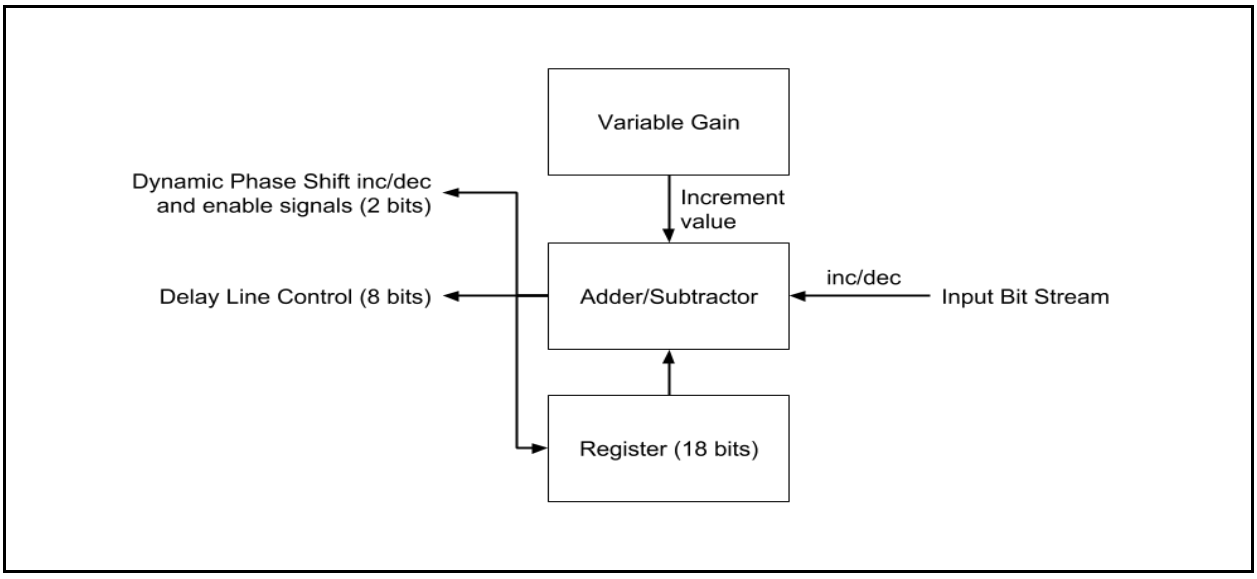


Figure 14: Integrator Implementation

As mentioned in section 3.2.2, the input bits are interpreted at either +1 or -1. They signal the adder/subtractor block to increment or decrement the value stored in the 18 bit register. The size of the increments is given by the variable gain block. The purpose of this implementation is to demonstrate the capabilities of the entropy extractor, so it is set manually. Later on, an adaptive filtering scheme can be applied to dynamically set the gain. Finally, the 8 most significant bits of the adder/subtractor result are set as the delay line control. When the integrator overflows or underflows, the MMCM dynamic phase shift is either incremented or decremented accordingly.

The implementation discussed in this section is not final in any way. It was presented to demonstrate the application of the results presented in the thesis. Further work and testing is required to determine the feasibility of this entropy extractor. These future steps are presented in the following chapter.

4.5 Summary

This chapter presented the details of an implementation of the entropy extractor outlined in section 3.2.2. It introduced non idealities due to digitizer delay and finite DAC resolution. Also discussed was a method of improving entropy extractor filter performance by increasing digitizer resolution. Finally, the chapter ended with a possible implementation using the entropy extractor presented in this thesis

.

5. Conclusion

This chapter will review the work presented in this thesis and provide suggestions for future work.

5.1 Summary

The main contribution of this thesis is a noise source agnostic entropy extractor architecture, first presented in section 3.2.2. The proposed architecture is general enough to work with a variety of noise sources with different spectral properties, including thermal and phase noise. This is accomplished by incorporating a digital whitening filter in feedback to bias the digitizer, removing autocorrelation in the generated bit stream. Different implementation considerations are covered in Chapter 4, such as the impact of digitizer delay and the finite quantization of the DAC. Finally, the end of chapter 4 shows a possible implementation using the entropy extractor presented in this thesis.

5.2 Future Work

The eventual goal is to implement the proposed entropy extractor in an actual TRNG to demonstrate its performance. To accomplish this, a few steps must be taken to solidify the analysis presented in this thesis. First is to expand on the filter used in the feedback loop. This thesis makes use of a simple digital integrator, but numerous types of whitening filters can be considered for implementation. Adaptive whitening filters are of particular interest due to their ability to whiten a wider variety of noise sources. Many implementation use algorithms such as least squares and least mean square to calibrate the filter coefficients. Another possibility is to use statistical tests to adapt the filter properties. Many TRNG standards already require built in

statistical tests to alert the system in case of poor performance. Making use of these already present metrics may potentially increase RNG performance with minimal increase in cost. Next is to characterize the benefit of using a ΣΔ DAC. Preliminary results suggest there is a benefit of using first and higher-order ΣΔ DAC, but a more detailed analysis is required to find the limits of its usage in this application. Finally, the only component not considered in this thesis is the post-processor. Cryptographic post-processors are widely used and often required by TRNG standards. Post-processors decrease the generation rate by compressing the bitstream to increase the entropy of each bit. This entropy extractor should be able to generate more entropy per bit, requiring less post-processing.

6. References

[1] Intel® Digital Random Number Generator (DRNG). (2014). 2nd ed. [ebook] Intel. Available at: https://software.intel.com/sites/default/files/managed/4d/91/DRNG_Software_Implementation_Guide_2.0.pdf [Accessed 7 Mar. 2018].

[2] Brown, R. Dieharder: A Random Number Test Suite. https://webhome.phy.duke.edu/~rgb/General/dieharder.php, 2018

[3] Rukhin et al., A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. Special-Pub:800-22 NIST, 2008

[4] Killmann, W., and Schindler, W. A proposal for: Functionality classes for random number generators, 2011.

[5] Barker et al., Recommendation for Random Number Generation Using Deterministic Random Bit Generators. Special-Pub:800-90A NIST, 2015

[6] Turan et al., Recommendation for the Entropy Sources Used for Random Bit Generation. Special-Pub:800-90B NIST, 2018

[7] Barker et al., Recommendation for Random Bit Generator (RBG) Constructions. Special-Pub:800-90C NIST, 2016

[8] Haahr, M. (2018). *RANDOM.ORG - True Random Number Service*. [online] Random.org. Available at: https://www.random.org/ [Accessed 6 Mar. 2018].

[9] D. Schellekens, B. Preneel, and I. Verbauwhede, FPGA vendor agnostic true random number generator, International Conference on Field Programmable Logic and Applications, FPL, pp. 1-6, 2006

[10] C. Tokunaga, D. Blaauw, T. Mudge, “True Random Number Generator with a Metastability-Based Quality Control,” ISSCC Dig. Tech. Papers, pp. 404-405, Feb. 2007.

[11] Caitlin R. S. Williams, Julia C. Salevan, Xiaowen Li, Rajarshi Roy, and Thomas E. Murphy, "Fast physical random number generator using amplified spontaneous emission," Opt. Express 18, 23584-23597 (2010)

[12] V. B. Suresh, D. Antonioli, and W. P. Burleson, “On-chip lightweight implementation of reduced NIST randomness test suite,” in Proc. IEEE Int. Symp. Hardw.-Oriented Secur. Trust, Jun. 2013, pp. 93–98.

[13] H. Zhun, H. Chen, "A truly random number generator based on thermal noise", *Proc. IEEE Int. Conf. ASIC*, pp. 862-864, 2001.

[14] N. Deák, T. Györfi, K. Márton, L. Vacariu, and O. Cret, “Highly efficient true random number generator in fpga devices using phase-locked loops,” in 2015 20th International Conference on Control Systems and Computer Science, May 2015, pp. 453–458

[15] Majzoobi M., Koushanfar F., Devadas S. (2011) FPGA-Based True Random Number Generation Using Circuit Metastability with Adaptive Feedback Control. In: Preneel B., Takagi T. (eds) Cryptographic Hardware and Embedded Systems – CHES 2011. CHES 2011. Lecture Notes in Computer Science, vol 6917. Springer, Berlin, Heidelberg

[16] Roubik Gregorian and Temes, G.C., “Analog MOS integrated circuits for signal processing,” Wiley, 1986, ISBN-13: 9780471097976

[17] Bernard Widrow and István Kollár, "Quantization Noise: Roundoff Error in Digital Computation, Signal Processing, Control, and Communications," Cambridge University Press, Cambridge, UK, 2008. 778 p. ISBN-13: 9780521886710, ISBN: 0521886716

[18] Chen YY, Huang JL, Kuo T (2013) Implementation of programmable delay lines on off-the-shelf FPGAs. In: Proceedings of AUTOTESTCON, pp 1–4

[19] Giordano R., Ameli F., Bifulco P., Bocci V., Cadeddu S., Izzo V., Lai A., Mastroianni S., Aloisio. A., High-Resolution Synthesizable Digitally-Controlled Delay Lines, IEEE Transactions on Nuclear Science, 62 (2015), n. 6, 3163-3171

[20] Zhao J, Kim Y B. A low-power digitally controlled oscillator for all digital phase-locked loops. VLSI Design Journal, Hindawi, 2010: 1

[21] S. K. Mitra, *Digital signal processing: a computer-based approach*. Boston: McGraw-Hill/Irwin, 2001, ISBN: 0072522615