2013

Design and Analysis of Digital True Random Number Generator

Avantika Yadav
Virginia Commonwealth University

Design and Analysis of Digital True Random Number Generator

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering at Virginia Commonwealth University

by

Avantika Yadav
Bachelor of Engineering
Motilal Nehru National Institute of Technology, Allahabad, India

Director: Dr Robert H. Klenke
Associate Professor
Department of Electrical and Computer Engineering

Virginia Commonwealth University
Richmond, Virginia
Fall 2013

ABSRACT

Random number generator is a key component for strengthening and securing the confidentiality of electronic communications. Random number generators can be divided as either pseudo random number generators or true random number generators. A pseudo random number generator produces a stream of numbers that appears to be random but actually follow predefined sequence. A true random number generator produces a stream of unpredictable numbers that have no defined pattern.

There has been growing interest to design true random number generator in past few years. Several Field Programmable Gate Array (FPGA) and Application Specific Integrated Circuit (ASIC) based approaches have been used to generate random data that requires analog circuit. RNGs having analog circuits demand for more power and area. These factors weaken hardware analog circuit-based RNG systems relative to hardware completely digital-based RNGs systems. This thesis is focused on the design of completely digital true random number generator ASIC.

TABLE OF CONTENTS

Acknowledgements

I offer my sincere gratitude to my advisor, Dr. Robert H. Klenke for giving me the opportunity to work under his knowledge and guidance. Throughout my thesis work, Dr. Klenke provided me constant feedback and valuable inputs to keep my thesis work on the right track. He also gave me ample time and flexibility to finish my thesis work.

I am greatly thankful for Samuel T. Mitchum, Jr. P.hd. for his valuable feedback in this work as this work was a continuation of his research work. He had been very helpful for providing me related information and necessary software throughout this work.

I would like to thank my advisory committee Dr. Alen Docef, Dr. Gary Atkinson and Dr. Carol Fung for their valuable feedback on my work. I would like to thank Advanced VLSI project team member David Jacob and Jeremy Cooper for helping with the layout of the RNG chip. I would like to thank the School of Engineering, Virginia Commonwealth University for providing the support and facility that I have needed to complete the thesis.

I would like to thank my husband Anurag Yadav for his support. I am greatly thankful to my daughter Ishi Yadav for her love and affection. I really appreciate her patience and understanding during my thesis work as I had not enough time to play with her. Finally, I would like to thank my family for their support.

CHAPTER 1 INTRODUCTION

Computer systems and telecommunications play an important role in modern world technology. The communication and data transfer through computers touches almost every aspect of life, i.e. transferring data, tracking personal data, trading over the internet, online banking and sending emails. As more vital information is transferred through wire or wireless means, the need to safeguard all this data from hackers is growing. All these security concerns emphasize the importance of developing methods and technology for the transformation of data to hide its information content, prevent its modification, and prevent unauthorized use.

Random number generation is a fundamental process for protecting the privacy of electronic communications. It is a key component of the encryption process that protects information from attackers by making it unreadable without the proper decryption process. Since the strength of an encryption mechanism is directly related to the randomness of the binary numbers used in it, there has been an enormous need to design and develop an efficient random number generator that can produce true random numbers to implement a safe and secure cryptographic system.

In addition to cyber security, random number generators (RNGs) are a vital ingredient in many other areas such as computer simulations, statistical sampling, and commercial applications like lottery games and slot machines.

1.1 Hardware vs. Software Random Number Generators

Most random number generators available today are software-based RNGs. As the algorithms used for software RNGs are fixed, the numbers generated by these appear random at any given time, but actually follow a predefined sequence. Thus, these numbers are not truly random and can easily be predicted by having knowledge of the generating algorithm. Random numbers generated by these algorithms can only mimic the true unpredictability of a truly random number generator. That is why software based RNGs are commonly referred to as "pseudo random number generators".

There are also hardware-based random number generators that are, in fact, pseudo random.

On the other hand, some hardware-based random number generators can generate truly random and unpredictable binary number sequences. Numbers generated by these types of hardware RNGs have no defined structure or order, the next number of the string is always a surprise, and it cannot be guessed by a human or any computational device by knowing the internal algorithm or structure of the circuit. These generators use some type of physical noise source as an initial value to generate truly random numbers. Some of the common physical noise sources that have been used so far as an inputs include: thermal noise, clock jitter, and nuclear decay. As the noise values these sources generate are unpredictable, it is impossible to predict the generated numbers.

There has been growing interest in the digital design area to develop simpler hardware-based true random number generators within last few years. Several Field Programmable

Gate Array (FPGA) and Application Specific Integrated Circuit (ASIC) based approaches have been used to implement these generators. Chip manufacturing companies are making silicon-based random number generators that can be integrated onto a processor chip to provide a high level of data security in digital systems. This thesis focuses the chip design for generating true random numbers using only standard digital components.

1.2Thesis Objective

The main goal of this thesis is:

a. To design a completely digital Application Specific Integrated Circuit (ASIC) based hardware True Random Number Generator (TRNG).
b. To prepare a test fixture to analyze the randomness of output random numbers and perform the Federal Information Processing Standard (FIPS) and the National Institute of Standards and Technology (NIST) randomness tests on the data.

To achieve thesis objectives, an ASIC was designed using the Mentor graphics Electronic Design Automation (EDA) tool and fabricated by the MOS Implementation Service (MOSIS) using AMI05 micron process technology.

A printed circuit board was designed to communicate between the fabricated TRNG and a microprocessor. A software program was written to control the ASIC operations and generate random numbers. The test board is used to assess the functionality of each block of the RNG chip and to generate and gather random numbers for long period of time. Once the chip functionality was tested, random numbers were generated by the RNG chip and uploaded to a personal computer (PC) for analysis. Finally, the FIPS and NIST statistical test suites are applied to the strings of random numbers.

1.3 Overview

The remaining chapters of this thesis are structured as follows: Chapter 2 covers the background of random numbers generators, such as descriptions of true and pseudo random number generators and their applications in digital world. Chapter 3 explains the design of the RNG of this thesis. Chapter 4 includes details that explain how testing has been performed on the RNG for its functionality as well as on random numbers generated from the chip. Chapter 5 covers results of the FIPS 140-1, FIPS 140-2 and NIST 800-22 tests on the RNG data. Finally, chapter 6 presents a summary of the work and possible future work on digital random number generation.

1.4 Abbreviations

Below are the listed abbreviations and acronyms used throughout this thesis.

Abbreviations	Stands For	Definition
ASIC	Application Specific Integrated Circuit	A chip designed for a specific application rather than general purpose use
LFSR	Linear Feedback Shift Register	A series of shift registers whose input is a logical combinations of previous states
RNG	Random Number Generator	A circuit that generates random numbers
PRNG	Pseudo Random Number Generator	A circuit that generates numbers which appear to be random but are actually predictable
TRNG	True Random Number Generator	A circuit that generates random numbers that are truly unpredictable
FIPS	Federal Information Processing Standard	Standards developed by government for use in computer systems
NIST	National Institute of Standards and Technology	Government organization to develop and apply technology standards

CHAPTER 2 BACKGROUND OF RANDOM NUMBER GENERATORS

2.1 Types of Hardware Random Number Generators (RNGs)

Hardware RNGs can be divided into two broad categories: pseudo random number generators (PRNGs) and true random number generators (TRNGs).

2.2 Pseudo Random Number generator (PRNG)

A hardware pseudo random number generator is a device capable of generating a sequence of binary numbers that imitates the properties of random numbers. The initial input value fed to the PRNG is called a ‘seed’. The output sequence generated appears to be random while it is not truly unpredictable.

A PRNG’s output sequence of binary numbers is a deterministic function of the seed value, meaning that sequence can be reproduced later if the seed is known. The term "pseudorandom" refers to the deterministic nature of the generator.

PRNGs are also periodic; as randomness is limited to seed generation, the output sequence of binary numbers will start repeating at regular intervals. The period of a PRNG is defined as maximum length of the non-repetitive pattern in the sequence. PRNG's containing *n* bits of internal state cannot have its period longer than 2^n or sometimes have a period that is much shorter depending on the given seed. The apparent randomness of a PRNG can be increased by including substantially more bits in the PRNG than are required by the

consuming application. However the additional bits require additional hardware to implement, which is an undesirable effect.

A good example of hardware pseudo random number generator is a Linear Feedback Shift Register (LFSR). An LFSR is basically set of shift registers connected in series with the outputs of some of the shift registers combined in exclusive-OR configuration to provide a feedback mechanism. When the inputs of the registers are fed with a seed value and the LFSR is clocked, it generates a pseudorandom pattern of 1s and 0s. Figure 1 shows the 4-bit LFSR using D flip flops and XOR gate. A LFSR is called a maximal length LFSR, if it can generate a stream of random numbers of maximum length of 2^n -1 before it starts to repeat, where n is number of register element in it. A 32-bit maximal length LFSR can generate about 4 billion random numbers before it begins to repeat the sequence of numbers again.

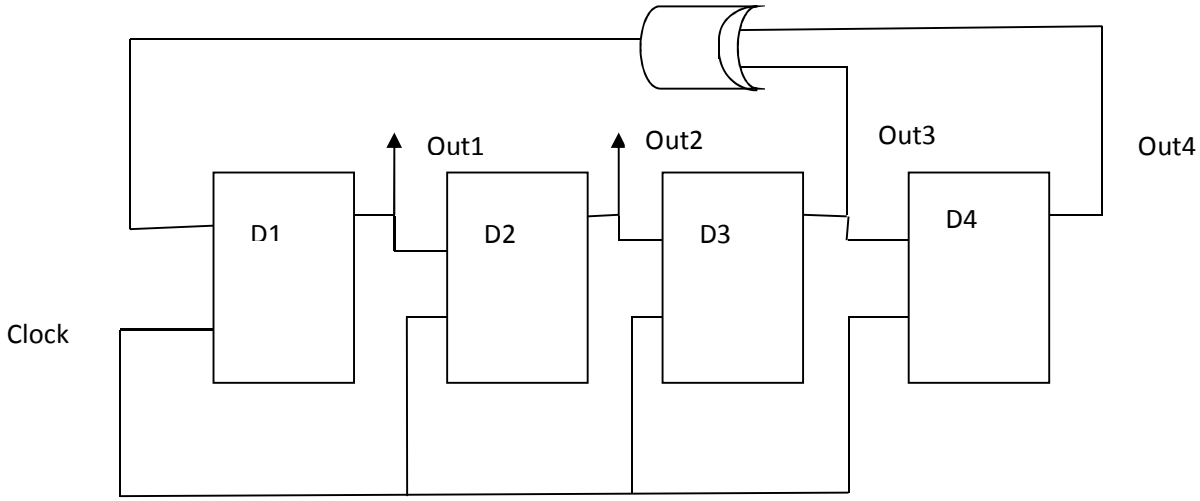
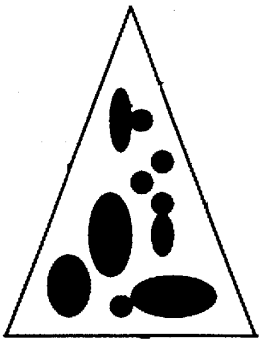


Figure 1: 4-bit LFSR

A very early and efficient approach of generating random numbers from a PRNG is described in the US patent 5732138 [1], which uses a chaotic system as a source of randomness to generate random numbers.

In this invention, a state of a chaotic system is digitized (changed into sequence of bits) to generate a binary sequence. This binary string is then cryptographically hashed to generate a second binary sequence. The second binary string is used as an initial state or seed of a pseudo-random number generator to generate random numbers. A random output data may be used for the security of confidential information.

A chaotic system is one that changes states unpredictably over time. Here, a chaotic system is used as a source of randomness to transform the state of the system into a binary sequence. In this invention, lava lamps are used as a chaotic system. Lava lamps are a system in which two different types of fluid which has different colors and different chemical properties combine in closed container. The color pattern of the fluids changes in unpredictable manner over time. Figure 2 in [1] shows how the pattern of the liquid changes after t seconds. It is reproduced below in Figure 2. In this invention, four lava lamps are used and their changing images or color patterns are converted to digital state using a digital camera. A digital camera records the picture as a digital image and is placed in front of the lava lamps. The images recorded can be described as a rectangular array of values such that each value corresponds to the color pattern at that point. The rectangular array consists of pixel values is a binary representation of the associated color and intensity.



Lava lamp after t seconds:

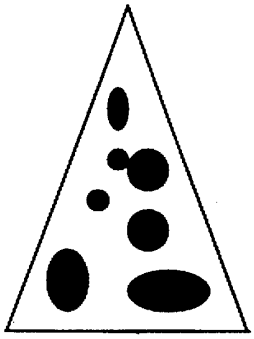


Figure 2: Lava Lamps [1]

The binary string generated from the images is transformed into a secondary binary string using a cryptographic hash function. This invention uses the dual mode Secure Hash Standard cryptographic hash function by NIST. A cryptographic hash function takes binary strings produced from images and converts into another binary string. For example, a cryptographic hash function takes a string x = 00001101100 and transforms it into another binary string hash(x) = 10110010011. The cryptographic hash function has following properties.

a. x and hash(x) may not have same length. A user will observe hash(x) only, and it will be hard for a user to use the output to determine the state of the chaotic system x.
b. The values of hash(x) and hash(x+1) are totally different. This property ensures that predicting a future output will be extremely difficult from the current output.
c. For two different values, x and y, there is possibility to get the value of hash(x) =hash(y). The third property explains that it will be extremely difficult to find an alternative value of x that has same hash value.

These properties make it difficult for a user to utilize the output of the generator to determine the state of the chaotic system.

The PRNG receives a seed, which is generated by applying a cryptographic hash function to the output string obtained from a chaotic source. Then the PRNG takes the appropriate number of bytes as a seed or an initial state and applies a deterministic transformation to obtain a new state and a random number. This step is repeated until the pattern generated from the PRNG eventually starts repeating. To refresh the system, a new seed is used which can be obtained as required. Figure 3 is the flow chart from [1] that describes the steps for generating random numbers.

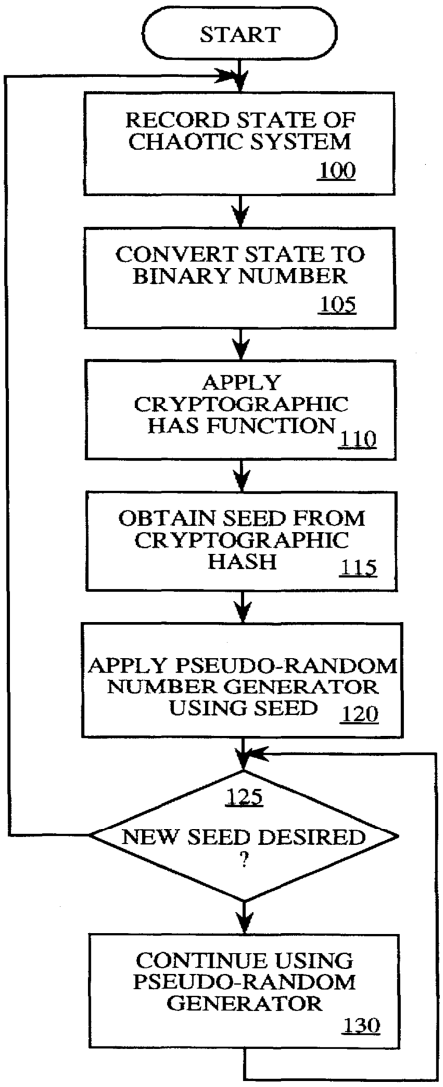


Figure 3: Flow Chart from [1]

2.2.1 Uses of Pseudo Random Numbers Generators

PRNGs are not suitable for applications where unpredictability of binary numbers is the most desirable feature, such as gaming and generating encryption/decryption keys. The characteristics of PRNGs make these generators suitable for applications such as simulation, modeling, fault grading and digital communication.

Texas Instrument uses PRNGs for the purpose of fault grading as described in [2]. The paper describes how the PRNG generates patterns which have been proven to provide high fault coverage. Once a design engineer has completed design and functional verification for an ASIC, the LFSR is designed and outputs of the LFSR are connected with the inputs of the ASIC. Figure 4 from [2] shows that the LFSR outputs are multiplexed with the ASIC input in such a way that the ASIC logic can be simulated by both the LFSR's output and the normal data input.

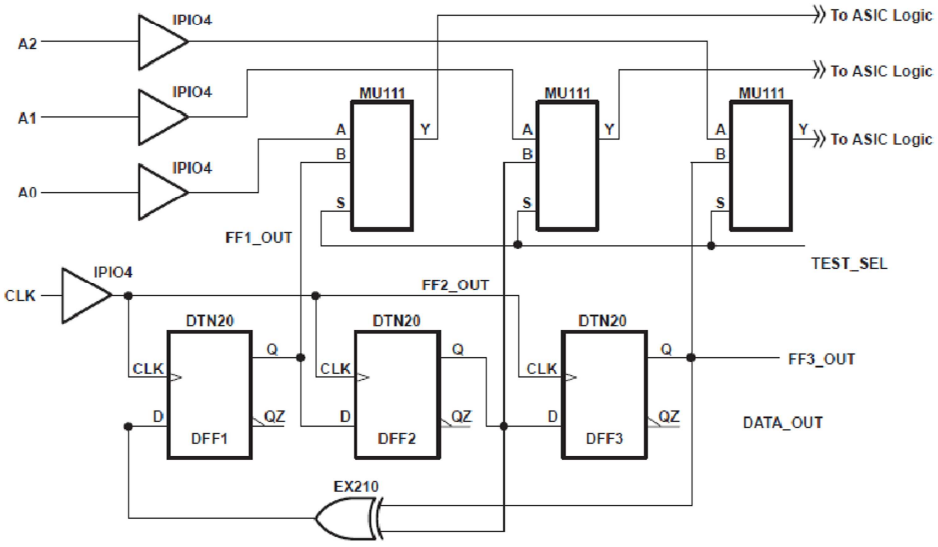


Figure 4: LFSRs Outputs Multiplexed with ASIC Inputs [2]

First, a designer would take the LFSR sub circuit and simulate it by itself so that he can observe the output pattern generated by the LFSR. The designer could then easily observe the effect of different seed values on the generated output patterns. It is important if all the 2^n -1 pattern that the LFSR can generate are not being used for circuit testing. A designer would take a print out of the output pattern generated by the LFSR and use it to verify that the LFSR is producing the correct signal, when the complete circuit (the ASIC and the

LFSR) is simulated. Next, the device is set in the test mode and the LFSR generates the inputs to the ASIC. The outputs of the complete circuit are sampled every clock cycle to generate the expected outputs from the ASIC with the inputs provided by the LFSR.

Once it is verified that the LFSR is producing correct inputs for the circuit, the resulting simulation vector can be used to fault grade the design. Fault grading ensures that the test vector set applied for the design will detect manufacturing defects, such as shorted transistors and open metal lines.

2.3 True Random Number generator (TRNG)

A hardware true random number generator is an electronic device that generates truly random and unpredictable binary numbers. The output pattern of a TRNG is arbitrary and non-deterministic in nature, meaning that the output binary numbers cannot be reproduced even if internal design and seed of the generator is known. As complete unpredictability is the key aspect of the true random number generator, a seed given to the TRNG must be random. Fortunately, it is not so difficult to collect true unpredictable randomness by tapping a chaotic world. Some of the examples of physical random sources are, thermal noise, shot noise, atmospheric noise, radioactive decay and clock jitter. A TRNG can be fed a seed from such a physical random process to get true random outputs.

Another characteristic of TRNGs is that they are non-periodic in nature (as opposed to PRNGs), meaning that output binary pattern of a TRNGs is never repeated even if the same seed is applied to the generator.

2.3.1 The Intel RNGs

Chip manufacturing companies such as Intel are including random number generators in their chips [3]. This paper illustrates a hardware based Intel Random Number Generator for use in cryptographic applications. Figure 5 shows the block diagram of the Intel RNG from [3]. The Intel hardware random number generators are based on unpredictable analog property such as junction or thermal noise. In this design, Intel RNG samples the thermal noise of undriven resistors by amplifying the voltage across it. One significant problem associated with this noise amplification technique is that random components are associated with local pseudorandom noise sources such as temperature and power supply fluctuations. The effects of these sources are minimized by subtracting the signals sampled from two adjacent resistors.

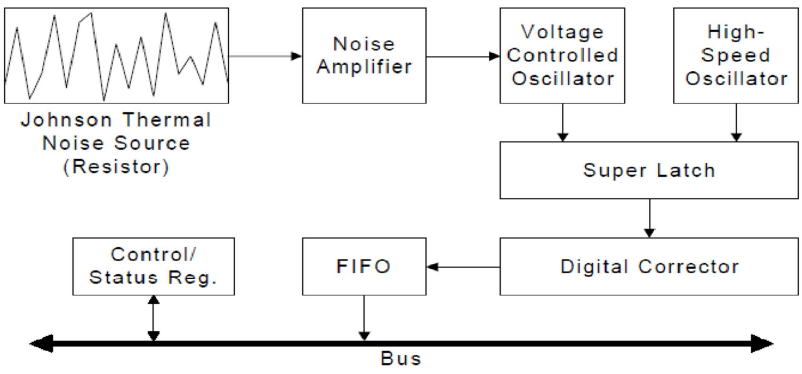


Figure 5: Block Diagram of Intel RNG from [3]

The Intel RNG uses two free-running oscillators, one fast and other much slower. The thermal noise source is used to modulate the frequency of the slower clock. The variable, noise-modulated slower clock is used to start the fast clock. The drift between the noise modulated slower clock and the fast clock provides the source of random binary numbers. Both oscillator signals are latched and then fed to the digital corrector. A Von Neumann

corrector converts pairs of bits into output bits by changing the bit pair (0, 1) into an output bit 1, changing (1, 0) into an output bit 0, and generating no output bit for pair (0, 0) or (1, 1).

Intel's early attempt to generate random numbers is described above in section 2.3.1. This approach includes ring oscillator based analog design of a RNG. An analog circuit consumes lots of power and area. Every year, chip makers are coming up with fabrication processes at finer scales, and packing more transistors in small area, which is very difficult if the circuit has analog circuitry. Intel came up with the design of a random number generator that contained only digital hardware.

This digital random number generator includes a pair of inverters, where the output of each inverter is connected to the input of each inverter. If the output of first inverter is 0, then the input of the second inverter is also 0, and then its output is 1. If the output of the first inverter is 1, the second inverter's output will be 0.

But these inverters are placed with two transistors in such a way that switching those transistors forces the inputs and the outputs of both inverters to the logic state 1. Figure 6 from reference [4] shows that the two transistors are connected to a clock that regularly turns both of inverters either ON or OFF.

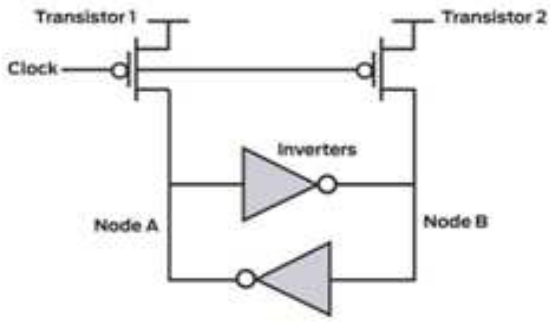


Figure 6: Intel RNG [4]

The two inverters in the circuit are not stable when all the inputs and outputs are in the same state. The inverter circuit is stable when the input and output are of opposite states. When these transistors are turned off, the two inverters, which were forced to have all their inputs and outputs in same state, race forward to acquire a stable state. Even though the inverters are the same in design, there is always a subtle difference in the speed or strength of their response. Thermal noise present within the circuit of the inverters determines the outcome of the inverter. One random output bit is generated at each clock cycle.

To make these output numbers even more random, Intel designs a three stage circuit which contains: the RNG, a conditioner, and a pseudorandom-number generator. Figure 7 shows the block diagram for three stage process [4]

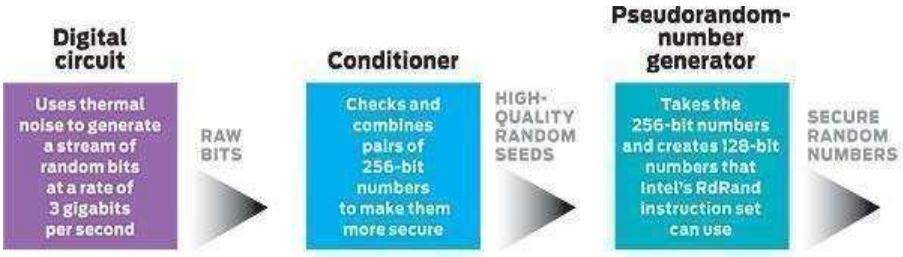


Figure 7: Block Diagram for three stage process [4]

This RNG generates random output numbers at a rate of around 3 gigabits per second. The RNG starts by collecting 512-bits at a time, which are the mostly random outputs of the two inverters. Furthermore, the circuit divides each package of 512-bits into a pair of 256-bits numbers. To make numbers more random they are mathematically combined or conditioned in such a way that produces 256-bits numbers that are close to perfect. The circuit takes two of these 256-bit values, multiply them together, and then collects the upper 256-bits of the resultant 512-bit number. Next, these 256-bit random numbers are used to seed a cryptographically secure pseudorandom-number generator that generates 128-bit output numbers. From one 256-bit seed, the pseudorandom-number generator can throw out many secure pseudorandom numbers.

2.3.2 Uses of True Random Numbers Generators

True random number generators are suitable for many specific applications, such as government-sponsored lotteries, gambling, simulation, and cryptographic-based security systems. True random number generators are fundamental keys to all aspects of security requirements to protect sensitive information in a computer and telecommunication world. Thus, it is important that hardware TRNGs used for generating binary numbers are not at risk to fail by knowing circuit algorithm or seed disclosure.

In [5], engineers from Intel presented a design of a ‘Digital Coin Tosser’ for future processors. Intel’s engineers announced that they could build an all-digital true random-number generator using the complementary metal-oxide-semiconductor (CMOS) process for chips with feature size as small as 32 nanometers or 22 nanometers. They informed that

they had already made an all-digital random-number generator using the 45-nm CMOS process that has been used to build Intel processors since 2007.

The Intel engineer describes that this device can generate billions of random numbers per second at very low voltage. Every bit in each string of output binary numbers is the result of "metastability." Generally, a digital device's output is sampled when it has settled on a definite value, either a 'one' or a 'zero'. Metastability occurs when the voltage is sampled during a bit transition, and the bit is caught between 'one' and 'zero'. The bit will settle down to one state, but it is hard to predict which side it will stay. The Intel researchers take advantage of this process and sample the output binary bits during transition. They improve the randomness even more by tuning the metastability so that the bit falls to 'one' or 'zero' with reasonably equal probability in random pattern, which is crucial for a coin flip.

2.4 Summary of Characteristics of Random Number Generators:

Below is the summary of the properties of a pseudo random and a true number generator.

Characteristic	**Pseudo Random Number Generator**	**True Random Number Generator**
Non-deterministic	No	Yes
Unpredictable	No	Yes
Reproducible	Yes	No
Periodic	Yes	No

Table 1

The summary of suitable application of random number generators is given in Table 2

Uses	**Suitable Generator**
Lotteries	TRNG
Gambling	TRNG
Security (Data and Voice encryption)	TRNG for seed
Simulation and Modeling	PRNG and TRNG
Fault Grading (Ex. Texas Instruments)	PRNG

Table 2

2.5 Clock Jitter based TRNGs

The period of oscillation for an ideal oscillator is constant, such that the time between the consecutive rise and fall of edges would be same. The period of oscillation for a ring oscillator composed of real elements is not constant, however, because the time between similar edges is not constant. The time period of such a clock is unpredictable. This variation of the time period in a ring oscillator is known as clock jitter (or phase noise). The simplest ring oscillator is made by connecting odd numbers of inverter gates in the form of a ring as shown in figure 8, the output of one gate becomes the input of the other gate. The last inverter output is fed back into the first inverter such that the last output of a chain is the logical NOT of the first input. The feedback provided by the last output to the input of the first inverter causes oscillation. A ring composed of an even number of inverters cannot be used as a ring oscillator; in that case the last output would be the same as the input of the first inverter which would create a stable state, suppressing oscillation.

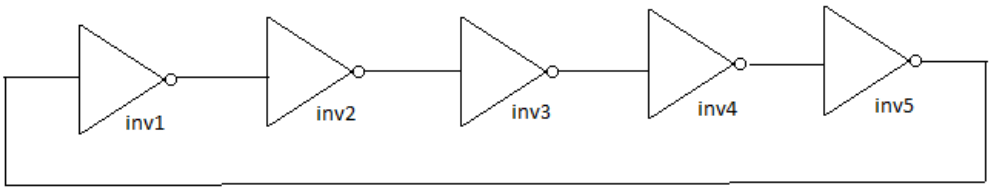


Figure 8: Ring Oscillator using five inverter gates

The time period of a ring oscillator changes randomly. Figure 9 shows how the ideal clocks are different from the clocks having phase noise or clock jitter.

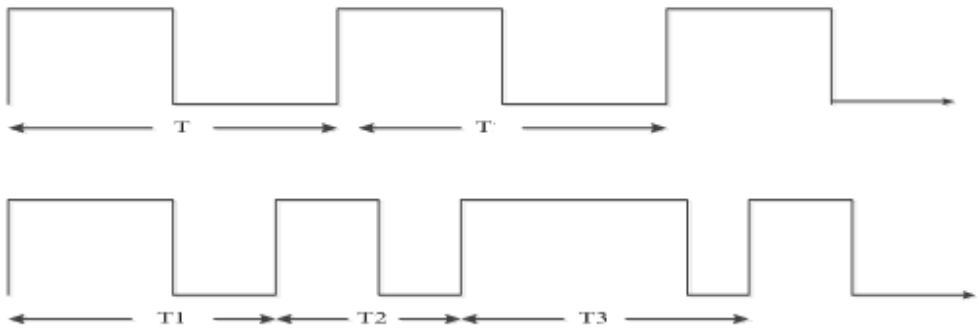


Figure 9: Clock Jitter

A clock jitter property of free running ring oscillator has been used for long time to generate true random number generator as a source of randomness. A very good discussion of building a two ring oscillator based RNG is illustrated in [6]. In this TRNG, a clock jitter in a ring oscillator is used as a source of noise. The TRNG is built with two ring oscillators and three LFSRs 13-bit, 19-bit and 32-bit. One ring oscillator clocks 19-bit LFSR and the other clocks the 13-bit and 32-bit LFSR. Figure 10 from [6] shows the block diagram of the TRNG.

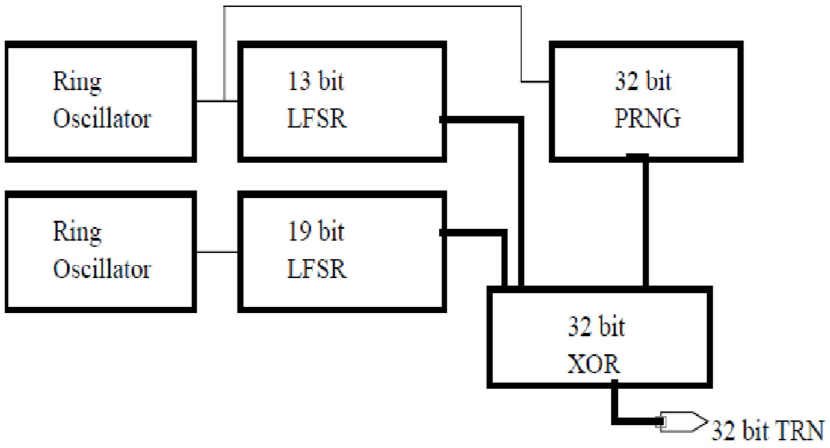


Figure 10: Block Diagram of TRNG [6]

Adding an independent action of split rotation on two LFSRs gives a more uncertain output sequence then the normally operating LFSRs. An extra circuit is added to provide split rotation for the 19-bit LFSR and the 13-bit LFSR. The upper half bits of these LFSRs are rotated left and the lower half bits are rotated right. These two LFSRs are then concatenated to generate the 32-bit long random number sequence, and then XORed with the 32-bit LFSR output to generate the final whitened 32-bit random number. A similar concept is used to design the TRNG of this thesis, however, the differences exist in the design of the ring oscillators and the control registers for LFSRs, such as addition a different independent action in 19-bit and 13-bit LFSR. In this RNG, a 'preloader' is added to reshuffle the 13-bit and 19-bit LFSR bits.

Another paper, [7], describes how the randomness related to the unpredictability of the frequency of ring oscillators is used as a source of implementing the true random number generator. This approach uses counters clocked by free-running ring oscillators and sampled at a regular interval. As there is some jitter associated with the clock, there is a

small amount of uncertainty about any particular value of the counter, and the output of the counter is random numbers at sample instants. However, the value generated by this generator can be approximated, as path of the generator through the range of possible values is fixed; that is, the generator always counts by incrementing its output every clock. To overcome this weakness, the path traversed by the generator as it counts, is varied by changing the generator operation only at sampling instant. Normally, the function performed by a counter is either incrementing or decrementing the number. If the function is changed to an independent operation like left shifting (doubling) only at the sample instant, then the output value becomes more random. Adding a tertiary independent operation such as transposition at the sampling instant generates even more random output. The generator, which previously had only one path through the range of possible values, now has many more paths throughout the range of possible values. Thus, it can be seen that adding more independent operations to the counter at sample instant increases the randomness of the output sequence.

As shown in figure 11 from [7], the RNG is designed with two 16-bit counters to generate a 32-bit random sequence: One counter counts up while the other counts down. Dividing the 32-bit counter into two 16-bit counters decreases the overall time required for generating all possible bit signals. Each counter has a unique ring oscillator. A secondary operation is provided by a 32-bit logarithmic shifter where the 5-bit counter is clocked by its unique ring oscillator that provides shift count for the log shifter. A tertiary operation is provided by adding a transposer unit that is 4:1 multiplexer. It selectively transposes bits in one of four possible patterns. The output sequence coming out from the generator is truly random.

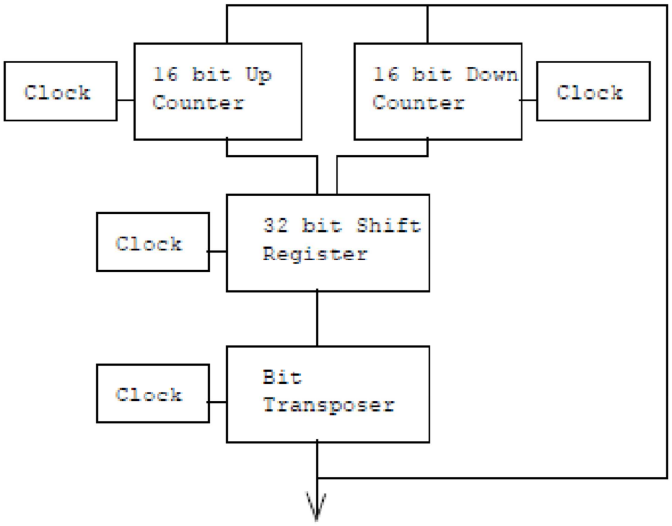


Figure 11: Random Number Generator [7]

2.6 RNG U.S. Patents of True Random Number Generators

The US patent 6,581,078 was issued in 2003 to STMicroelectronics SA.A for a true random number generator [8]. In this invention, random numbers are generated by combining a physical noise source's signals with signals produced by a pseudo random number generator. The combined signal is fed as an input to the PRNG. The final random output numbers are unpredictable thus suitable for cryptography.

The circuit of this RNG includes: a pseudo random number generator, a physical noise source, a logic circuit, a memory unit and an output interface. The PRNG used in the generator is based on a linear congruence algorithm. The PRNG is characterized by the equation x (n+1) = a*x (n) + b (mod c), where its last output signal is related to its previous outputs. Here, x (n+1) is the last output signal and x (n) is the previous output signal. The coefficients a, b and c are dependent on the statistical characteristic of the PRNG. The

physical noise source is formed by shift registers sampled at a frequency other than the frequency of the central processing unit which is controlling the whole circuit. The shift registers produce a digital signal having a size appropriate for the PRNG to receive the digital signals as an input. The logic circuit is a two-input exclusive-OR (EX-OR) gate. The memory unit receives the output signal from the pseudo-random number generator and supplies this signal to the logic circuit. The content of the memory unit is erased as soon as it receives a new digital output signal coming from the PRNG. The output interface receives the generated true random numbers. Figure 12 shows the block diagram of RNG from [8].

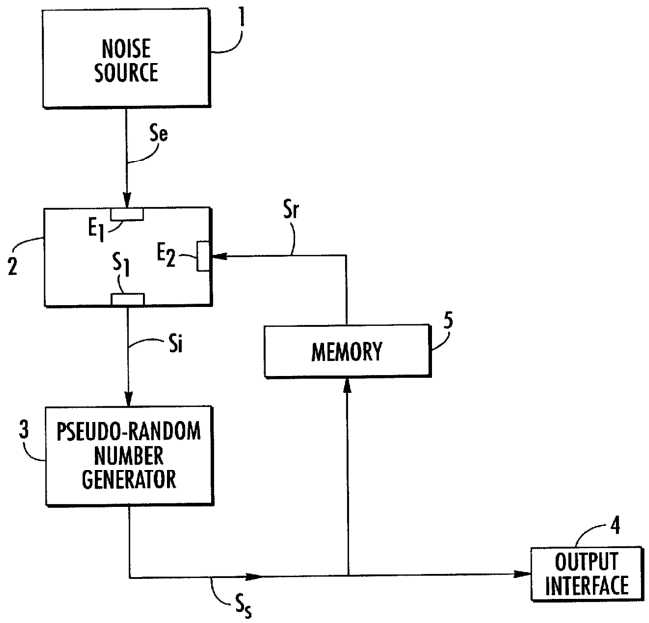


Figure 12: Block Diagram of RNG patent 6,581,078 from [8]

The operating principle of the circuit is as follows: The physical noise source is sampled either at a fixed frequency different from the operating frequency of the central processing unit, or at a variable frequency produces digital signals. These signals are sent at a first

input of the logic circuit. At the same time, the memory unit which stores the outputted digital signals from the PRNG sends the digital signal to the second input of the logic circuit. The logic circuit (EX-OR) combines these two inputs (received from the physical noise source and the memory unit) and feed as the input of the pseudo random number generator. The resulting output signals from the PRNG are true random digital numbers, which are stored in the memory unit, and also received by the output interface.

In the beginning, the memory element is empty; in that case, the logic circuit sends an intermediate digital signal similar to the input noise signal to the PRNG as the input signal. However, if the memory unit is not empty when a first random number is generated, it sends a return signal to the logic circuit while emptying the contents of its memory. In other words, the memory unit sends its information to the logic circuit in the form of digital signals immediately upon receiving a new digital signal from the PRNG.

In summary, digital signals generated from a physical noise source are combined with signals generated by a pseudo-random number generator. The combined signals are fed to the pseudo random number generator; the resulting output signals are true random digital bits. The output random numbers are suitable for cryptography.

The US Patent 7124157 was issued in 2006 to HMI Co., Ltd. for a random number generator [9]. In this invention, a random number generator has an amplifier to amplify noise signals generated from a noise source and sent to a digitizer to digitize the amplified noise signals. The digitizer includes a serial register, which outputs serial digital random numbers. The generator consists of a serial-parallel converter, which converts serial digital random output from the digitizer to parallel signals. In order to adjust the probability of the

output random numbers, a generator includes a bit masking section, which can mask and output some of the random bits. The bit masking section may mask bits controlled by an external device. This generator also includes memory to store the generated output random numbers.

The operation of the generator is as follows: The thermal noise of a semiconductor is used as a source of noise input for the generator. This signal is amplified by the amplifier and the amplified signal is sent to a Schmidt trigger gate, which converts an analog noise signal to rectangular waves with a pulse width similar to the magnitude of an analog noise signal.

The output signal from the Schmidt trigger is inputted into the serial register, which is mainly three D-flip flops connected in series utilizing a sampling clock. The digitizer digitizes a noise signal at the sample time and the output is random binary bits. The output from the digitizer is fed to the serial-parallel converter to convert the 8-bit parallel output data.

The bit masking section masks some bits of the 8-bit random number output. It has an OR circuit which performs logical OR operation between the output data bit and the bit of the masking register and masks certain bits of the random data. The generated random numbers can be used for an encryption system.

CHAPTER 3 TRNG DESIGN

3.1 Overview

The TRNG in this thesis uses the clock jitter property of the ring oscillator as the source of noise to generate true random numbers. A clock jitter is described in section 2.5. The principles behind the TRNG ASIC are to operate Linear-Feedback Shift Registers (LFSR) clocked by such ring oscillators to take advantage of the unstable output frequency of the oscillator. LFSRs are an easy way of generating pseudo-random numbers. The basic design of a LFSR is simply to have a shift register of appropriate length that has feedback taps at certain points. These taps are XORed together to form the input of the shift register. If the right set of taps is used, then the LFSR can generate a pseudo-random sequence of the maximum length without repeating. A design of the LFSR is described earlier in section 2.2 of chapter 2. A paper published by Xilinx lists the appropriate taps for maximum-length LFSR counters of up to 168 bits [10].

There are 3 different LFSRs in this design: the 13-bit, 19-bit and 32-bit LFSR, which have taps to generate sequences of maximum length random numbers. Figure 13 shows the block diagram of the TRNG design.

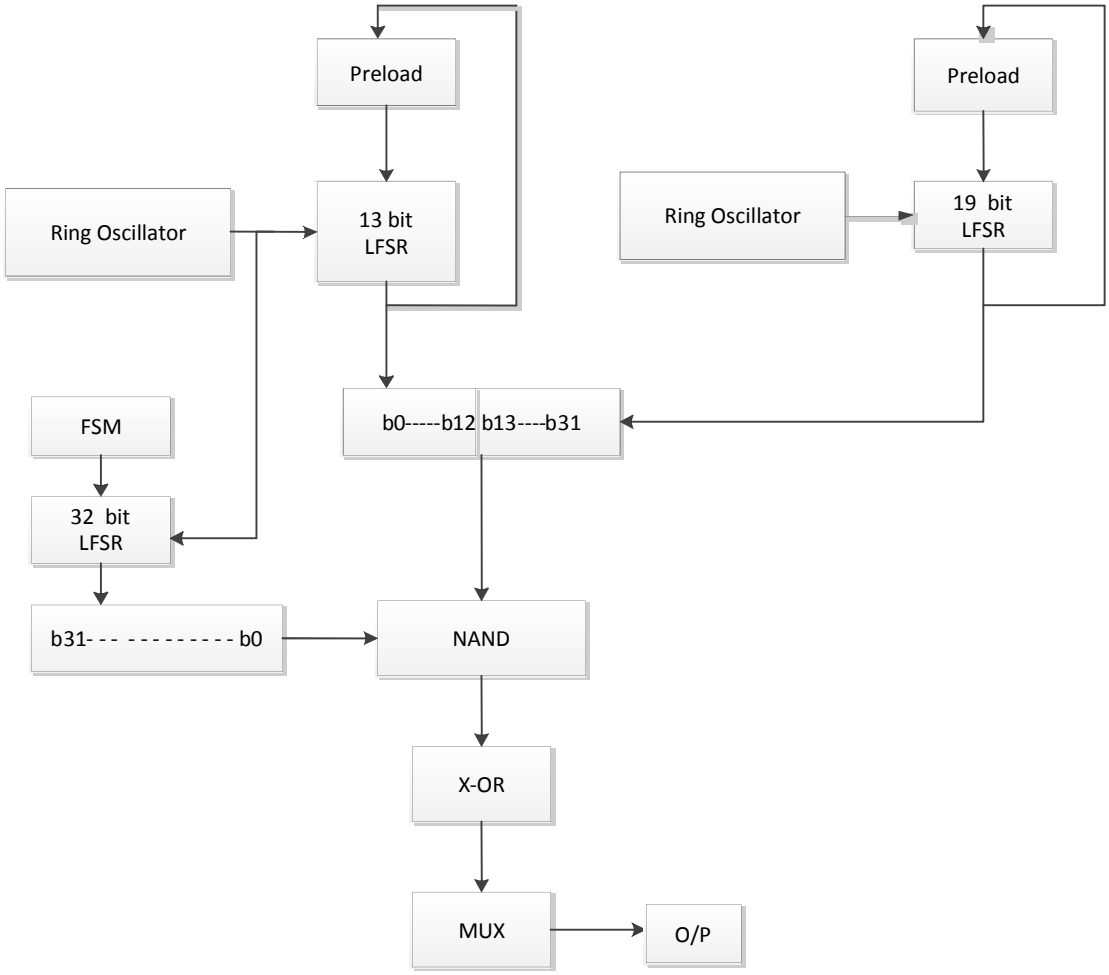


Figure 13: Block Diagram of TRNG Design

The 13-bit and the 19-bit LFSRs are each clocked by a noisy ring oscillator; each LFSR is preloaded with its own output bits and the bits are shuffled before preloading. The output binary numbers from the two LFSRs are concatenated to generate a 32-bit random number. Another 32-bit LFSR which is clocked by the other ring oscillator has its own finite state machine to control the LFSR operations, which generates a 32-bit pseudo random numbers. The outputs from the 13-bit and the 19-bit LFSRs (that are combined to make 32-bit random numbers) and 32-bit PRNG are selectively flipped (XOR). The output will be

random number, which is then multiplexed to get the 16-bit output data bus that will be either high or low word depending on a ‘word select input’. Below is the description of each functional block.

3.2 Functional Blocks Description of the TRNG

3.2.1 13-bit and 19-bit LFSR

The 13-bit LFSR is made with shift registers with XOR taps at 12, 3, 2 and 0 to provide linear feedback logic. Another 13-bit register is added to shuffle the output more, and a 13-bit register that holds the output until the next read request. There are MUXs on the input of the LFSR that input from either the shuffle register or back from the output of the LFSR. The 19-bit LFSR is similar in design to the 13-bit LFSR. It has 19-bit shift registers with XOR taps at 18, 5, 1 and 0 to provide feedback to the LFSR. Similar to the 13-bit LFSR, it has another 19-bit register to shuffle the output more, a 19-bit register that holds the output until the next read request, and MUXs to input the output. Figure 14 shows the block diagram of the 13/19 bit LFSR.

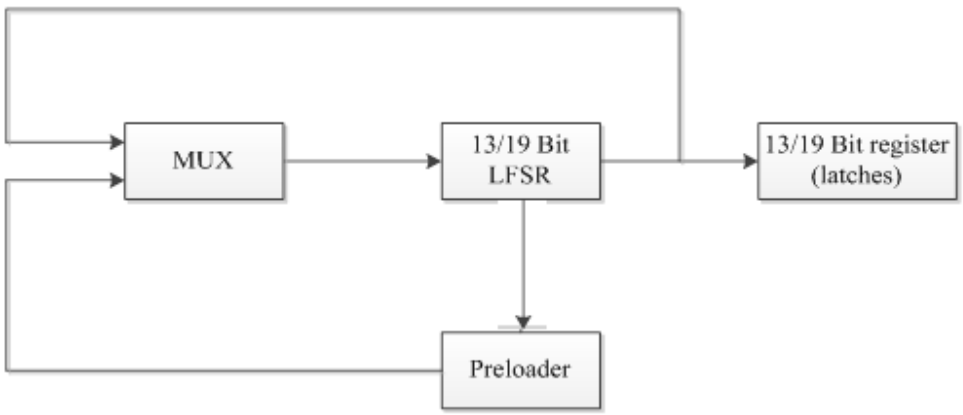


Figure 14: Block Diagram of 13/19 Bit LFSR

When these LFSRs get a read request from the system, they internally generate a pulse called “did”. At the beginning of this pulse, the current value of the LFSR is latched out, while a special “preloader” shuffling of the bits is stored in the LFSR. Here, these values are being shifted based on a variable frequency clock, so the amount of time it takes to shift through these values is random. After this, the LFSR continually shifts with taps at 18, 5, 1 and 0 for the 19-bit LFSR and 12, 3, 2, and 0 for the 13-bit LFSR. This continues until the next read request comes into the system. Also, each LFSR provides an asynchronous reset which sets all values back to 0.

3.2.2 32-bit LFSR

The 32-bit LFSR is divided up into two different sections. First, there is the actual LFSR itself. It has taps from the following bits: 31, 21, 1, and 0. It also has an input signal that tells it when to shift, and another reset signal which resets the LFSR to all zeros. Second, there is a finite state machine (FSM) for this LFSR that controls the LFSR. This state machine makes sure that the LFSR shifts exactly 32 times and then stops until it needs to generate a new number. This is to make sure that the old number is entirely shifted out, so that the new random number will contain all new bits.

The FSM and the LFSR have been partitioned into two different parts to ease the implementation. To do this it has two 2-bit state variables, and a 5-bit counter. There are also the 32-bit shift register, a 32-bit register to hold the output, and four control registers. The LFSR and the counter will have to be reset at times, so these 37 will have to have ‘reset’ functionality, while the other 40 can just be standard D-FFs. Then, there is the linear

feedback logic with three XORs and inverter. There are various MUXs that control what happens depending on the state of the FSM.

The FSM controls when the PRNG generates random numbers. It takes a reset, clock, and read signal, and outputs a ready signal and a state that is used in the 32-bit LFSR.

The LFSR shifts the value while the state input is “01”, which is the input given by the FSM to control the shifting. Figure 15 shows the block diagram for 32-bit PRNG.

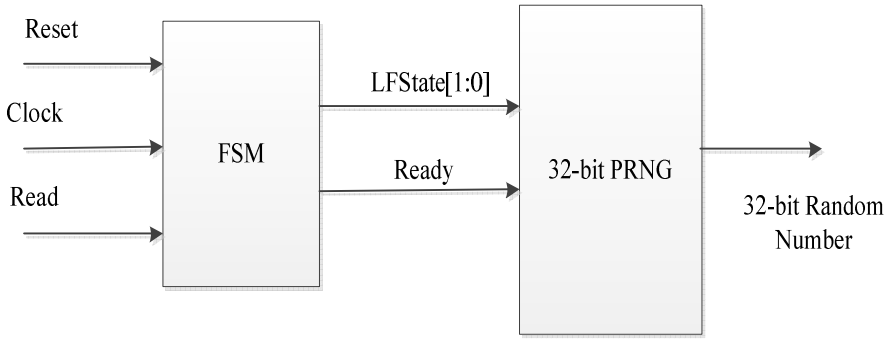


Figure 15: Block Diagram of 32-Bit PRNG

3.2.3 Ring Oscillator

The design of the ring oscillator is based on the design that has been described in paper [11]. Two ring oscillators are used to provide clock signals in this TRNG design. One ring oscillator clocks the 19-bit LFSR and the other ring oscillator provides clocks to the 13-bit LFSR and the 32-bit PRNG. The ring oscillator that is used in this design is not that large, but is fairly complicated in construction. First, the input frequency select signals are latched, back to back, into the system to reduce the metastability of the clock. Then these latched select lines are decoded into four one-hot signals that select what frequency is being used at the moment.

Second, it has the actual oscillation ring. The first part of this is fairly standard; it's simply a chain of inverters. The output of this section is selected by sel_x_n. Then there are three more sections of this chain which are simply an inverter followed by a NOR gate. The output of each of these sections is selected by sel_x_{n-1} down to sel_x_0 in that order. Each of these NOR gates not only have the inverter before them as an input, but also the select line of the signal before it. This disables the last part of the chain which is unused and forces those outputs to be low. This prevents glitches when the frequency select lines switch to a slower clock frequency.

Each of the output clock signal are then routed into an AND-OR block using NANDs. Using this, each of the signals is selected only if the associated sel_x line is high. Then, the output of this block runs into another inverter which then runs into a NOR gate along with the 'reset' signal and the 'notRun' signal. This simply prevents the clock from oscillating when the clock is disabled. The output of this NOR gate is the clock signal that is outputted. This signal is also routed back around to the beginning of the inverter chain to make the signal loop, so that oscillations can continue. Figure 16 shows the block diagram of the ring oscillator.

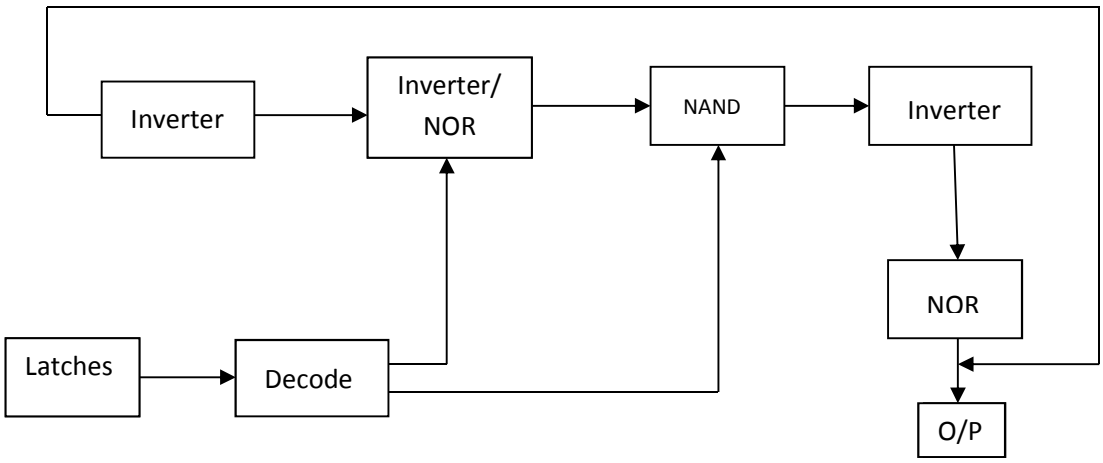


Figure 16: Block Diagram of Ring Oscillator

3.2.4 NAND/XOR Block

This block inputs the 32-bit data buses from the 19-bit and 13-bit LFSRs and the 32-bit data bus from the PRNG and inhibit signal for each. The output is a 32-bit bus which implements the function:

(Data1 *NAND* T_ Inhibit) XOR (Data2 *NAND* P_ Inhibit).

The 'true and pseudo random number inhibit' signals are helpful for debugging purposes. Setting any of those signals disables the values of either the TRNG or the PRNG from being output, so that they can be examined one at a time. The block diagram is shown in figure 17.

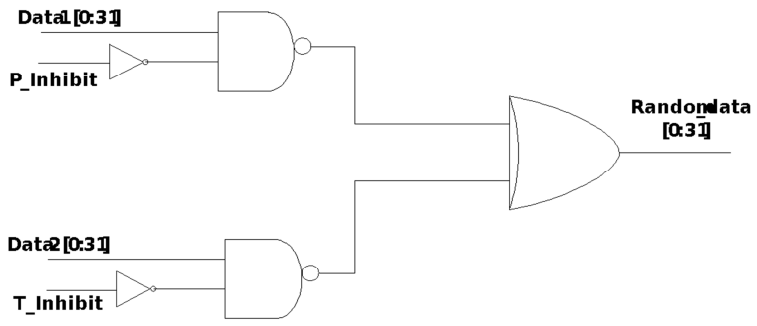


Figure 17: NAND/XOR Circuit

3.2.5 Output Block

To facilitate the pin limitations of the tiny chip for fabrication, multiplexing of the output pins was required to get the 32-bit output to the outside world. It accepts 32-bit random numbers and generates a 16-bit output bus that will be either the high or low word

depending on a ‘word select input’ at the output of the TRNG. This block contains latches and MUXs to store and output a 16-bit data bus respectively. When the ‘most significant word’ line is high, the latched high word is correctly output, and when it is low the current ‘least significant word’ is passed straight through to the 16-bit output bus. The block diagram is shown in figure 18.

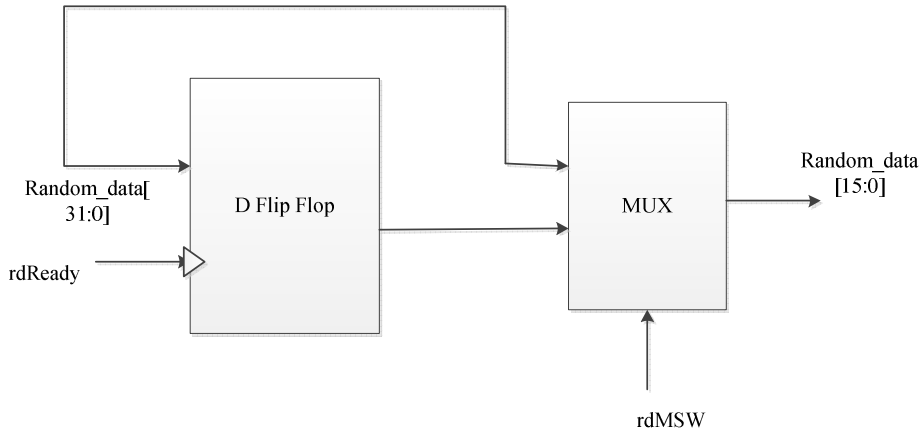


Figure 18: Output Circuitry

3.3 Design Implementation

Following are the steps to describe the design methodology of each block.

1. The VHDL code was written for each functional block to partition the design into relatively small pieces which could be implemented individually.
2. The Mentor Graphics Modelsim and the Leonardo Spectrum were used to simulate the behavioral VHDL description and synthesize into a gate level (structural) Verilog description.

3. The Design Architect (DAIC) was used to create and verify the schematic and symbol from the gate level description. The ELDO simulation was run for each schematic to make sure that it functions correctly.
4. The IC station's Schematic Driven Layout (SDL) feature was used to lay out each functional block.
5. The Layout Versus Schematic (LVS) check was run frequently to be sure that each block was wired the same as the schematic.
6. Finally, the MachTA was used for the post layout simulation for each functional block and then to verify a top level design layout.

Figure 19 shows the IC layout of the TRNG. The chip layout area was 3000λ*6000λ. There were 20 pads aligned horizontally at each side, 10 pads vertically at each side and four corner pads to connect altogether. Six types of pads were used: input(PadInC), output(PadOut), unconnected pins(PadSpace), corner pad(PadFC), VDD(PadVdd), and GND(PadGnd). When they were aligned properly it allocated an exact area of 3000λ*6000λ to fit a chip design layout inside the frame.

Looking from the top, the first layout placed was the 32-bit LFSR along with the PRNG FSM., two ring oscillators were placed below the PRNG layout and the 19-bit and the 13-bit LFSR were located just underneath them. At the bottom, the AND-XOR layout and output stage layout were placed.

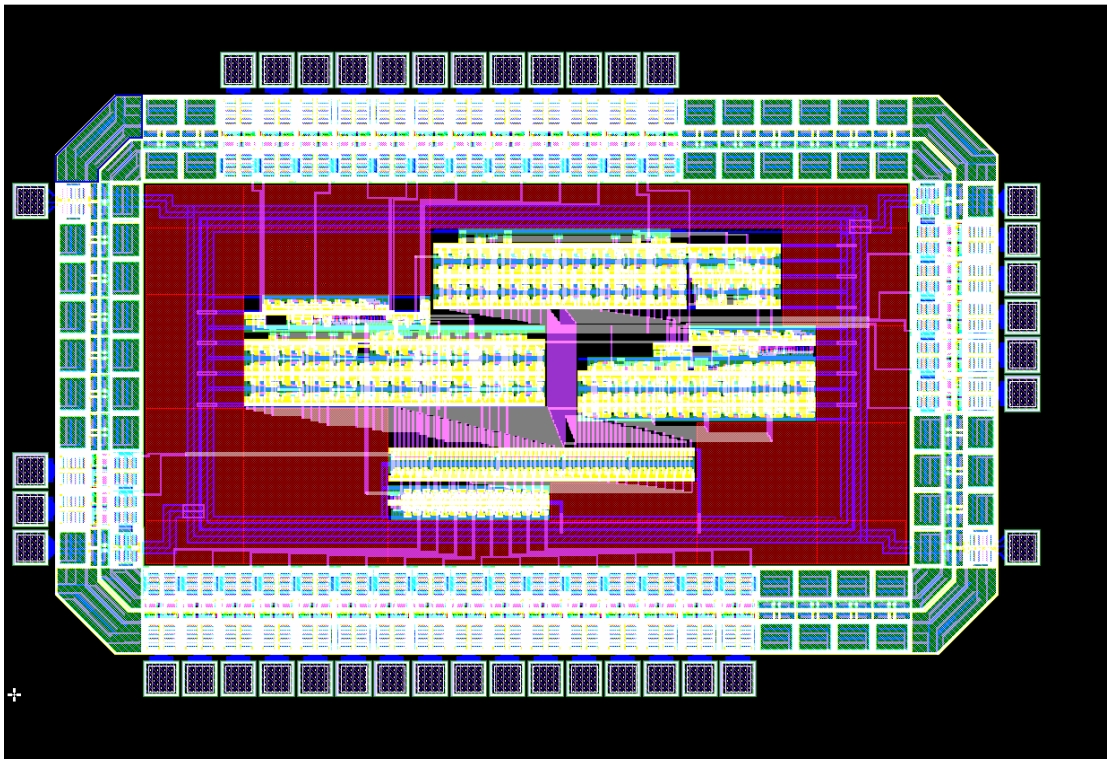


Figure 19: TRNG Chip Layout

Once the design was completed and successfully simulated, it was sent to MOSIS for fabrication in AMI05 technology.

3.4 Chip-Level Functional Description

This TRNG chip has a total of 20 output signals and 15 input signals, power signal VDD and GND signal. Table 3 lists a description of the operation of each pin.

Pin Name	Description
VDD	5V power input
GND	Ground signal
busRst	Reset signal
busCS	Chip Select
busOBO	On Board Oscillator select line
busIRun	Run signal
busMSW	High/Low word select line
T_Inhibit	inhibit true random number control line
P_Inhibit	inhibit pseudo random number control line
busC19Fix	LRO19 fixed frequency select line
busC19S[0,1]	LRO19 external frequency select lines
busC13Fix	LRO13 fixed frequency select line
busC13S[0,1]	LRO13 external frequency select lines
busO[1,2]I	external oscillator inputs
busO[1,2]O	external oscillator outputs
busData[0,..,15]	random number output lines
busAck[1,2]	TRN and PRN acknowledgement lines

Table 3: Input/output pin descriptions of ASIC

The overall chip was controlled by relatively few signals. The reset, run, chip select, and word select inputs were the most important for overall operations. Here is the description of each signal.

1. The ‘reset signal’ resets all LFSR flip flops, disables the ring oscillators, and restarts the pseudo random number generator.

2. The ‘run signal’ enables a generation of the clock signals and the LFSR computations.
3. The ‘chip select’ signal affects the read request inputs to the different LFSR’s, and affects when the outputs are stable.
4. The ‘word select input’ selects between the ‘least’ and the ‘most’ significant word to output to the 16-bit data bus. It also signals when to generate a new random number.
5. The ‘true and pseudo random numbers inhibit signals’ are used for debugging purpose.
6. There are two ‘ACK’ outputs that inform the user when the PRN or the TRN is ready for reading.
7. The 16-bit data output bus is the output for the generator. Finally, there are clock related signals that allow the most user-control of the behavior of the chip.

There are two clock inputs that can be used to replace the ring oscillators. An ‘on board oscillator’ select signal is used to select between the external clocks and the internal ring oscillators. There are also two clock outputs. If the ‘on board oscillator’ select line is enabled, and the ring oscillator clock signals are output, otherwise the external oscillators are routed through those output lines. If the internal ring oscillators are being used, there are six other signals used for changing their frequency by selecting which stage the final clock signal comes from. There is one fixed frequency select signal for each ring oscillator. In addition, each ring oscillator takes two frequency select lines to choose one of four possible frequencies (output stages). The fixed frequency select signals determine whether or not the frequency select lines are chosen externally or internally.

The initial setup for general operation of the RNG chip is performed by disabling the ‘reset’ signal, enabling the ‘run’ and ‘chip select’ signals, and disabling the ‘word select line’. Either the ‘true’ or ‘pseudo inhibit signals’ can be enabled to disable one of the RNG sources on the output. Also, the system clocks can be set up to use a variety of sources. For instance, if the ‘on board oscillator’ is disabled, the frequency select signals don’t need to be set, but two external clocks must be input. On the contrary, if the ‘on board oscillator’ is selected, the external clock inputs are unnecessary but the frequency select signals must be set to the right mode. Usually, the user will want to set the fixed frequency select signals low, so that the frequencies will vary randomly over time, based on certain TRN bits. The fixed frequency select signals are most likely to be used for debugging purposes.

Once the control signals are set up, the actual processing occurs after the ‘run’ signal goes high. If the ‘chip select’ is high, the true random number should be ready in just a few clock cycles, and the TRNG’s ‘ACK’ signal should go high. The PRNG will take at least 32 clock cycles to complete generating its number. When it is complete, the PRNG’s ‘ACK’ signal should go high. The value should remain high until the ‘word select line’ is pulsed high to read the ‘most significant word’. When the ‘word select line’ goes low again, the next pseudo random number should begin computing. The next true random number is also latched few clock cycles after the ‘word select line’ goes low. In other words, the user should only have to wait for ‘ACK’ to be high, read the LSW from the data bus, pulse the ‘word select line’ high for a time while reading the MSW of the RN from the data bus, and set it low again to begin a new RN generation.

3.5 TRNG Chip Simulation

3.5.1 Simulation with External Clocking

Figure 20 shows a random number computation with the chip running off of external clocks.

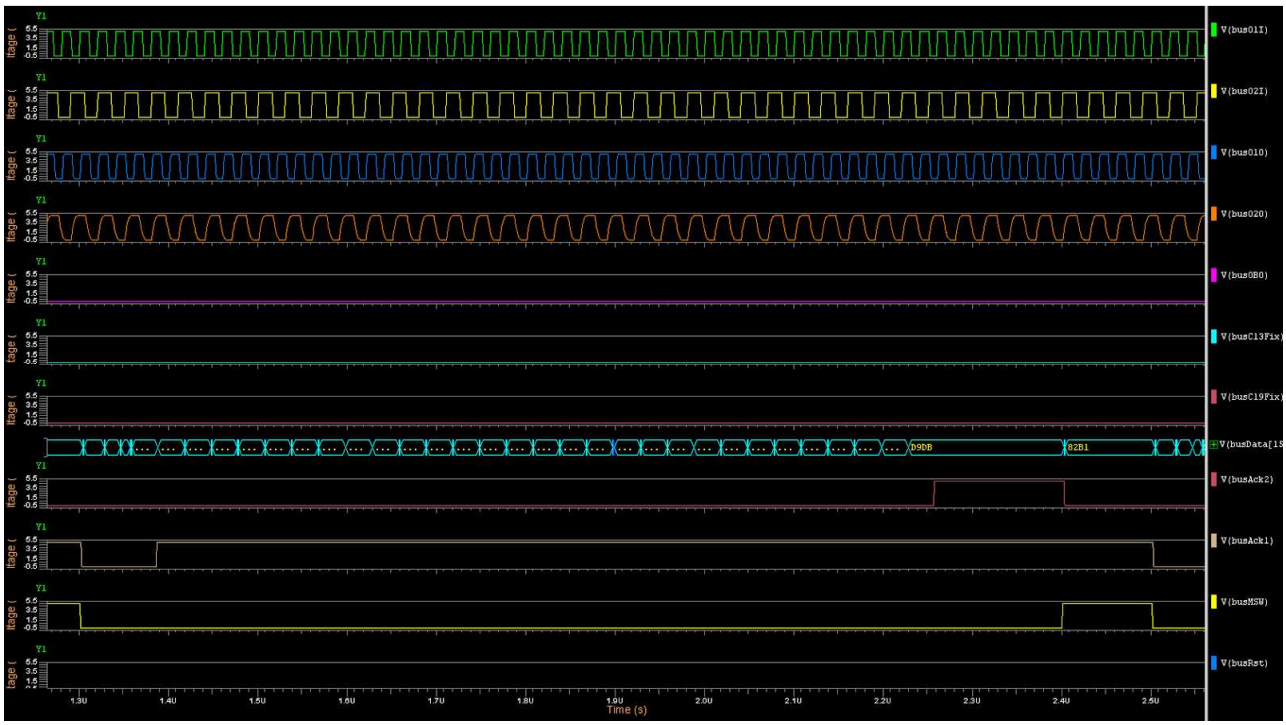


Figure 20: Chip Simulation with External Clocking

The ‘busO1O’ and ‘busO2O’ are the clock signals that are being used as the two clocks in the design. The ‘busO2O’ clock signal has significantly higher rise and fall times. This is because this clock runs the 13-bit LFSR, the 32-bit LFSR and the 32-bit FSM, while the other clock only runs the 19-bit LFSR, so it has a much higher capacitance to drive. As long as ‘busOBO’ signal goes low, the internal oscillator does not affect the computation, so the value of ‘BusC13Fix’ and ‘BusC19Fix’ signals have no effect on operation. The output data bus undergoes 32 transitions before both of the ‘Ack’ signals go high. This is

the 32-bit LFSR completely shifting out the old value before signaling that it is ready. The output value changes when the ‘busMSW’ value goes high because it is then displaying the high order word of the 32-bit value computed. Then, when the ‘busMSW’ signal goes low, the logic begins computing the next value. Table 4 gives a summary of status of control signals to run the chip with external clocks.

Control Signal	**Status**
busRst	Low
busCS	High
busOBO	Low
busIRun	High
busMSW	Low/High
busAck2	High
busAck1	High

Table 4

3.5.2 Simulation with Internal Clock

The normal mode of operation of the TRNG chip is working with the internal clock running at a random frequency. The ‘busOBO’ signal goes high which indicates that internal ring oscillators are selected to generate clock signals for the LFSRs. It can be noticed that the‘busO1O’ and ‘busO2O’ clock signals are not following the external clock sources now. The ‘busC13Fix’ and ‘busC19Fix’ signals go low, so that frequency provided by the oscillator varies randomly depending on the TRN bits. The 32-bit LFSR is being

inhibited by setting P_inhibit signal high. The ‘ACK1’ signal goes high when true random numbers are ready. The ‘busMSW’ signal goes low to read the ‘least significant word’ and goes high to read the high order word of the 32-bit output data. Table 5 summarizes the status of control signal in normal mode operation and figure 21 shows the MachTA simulation of the TRNG chip generating the true random numbers.

Control Signal	**Status**
busRst	Low
busCS	High
busOBO	High
busIRun	High
busMSW	Low/High
busAck2	Low
busAck1	High

Table 5

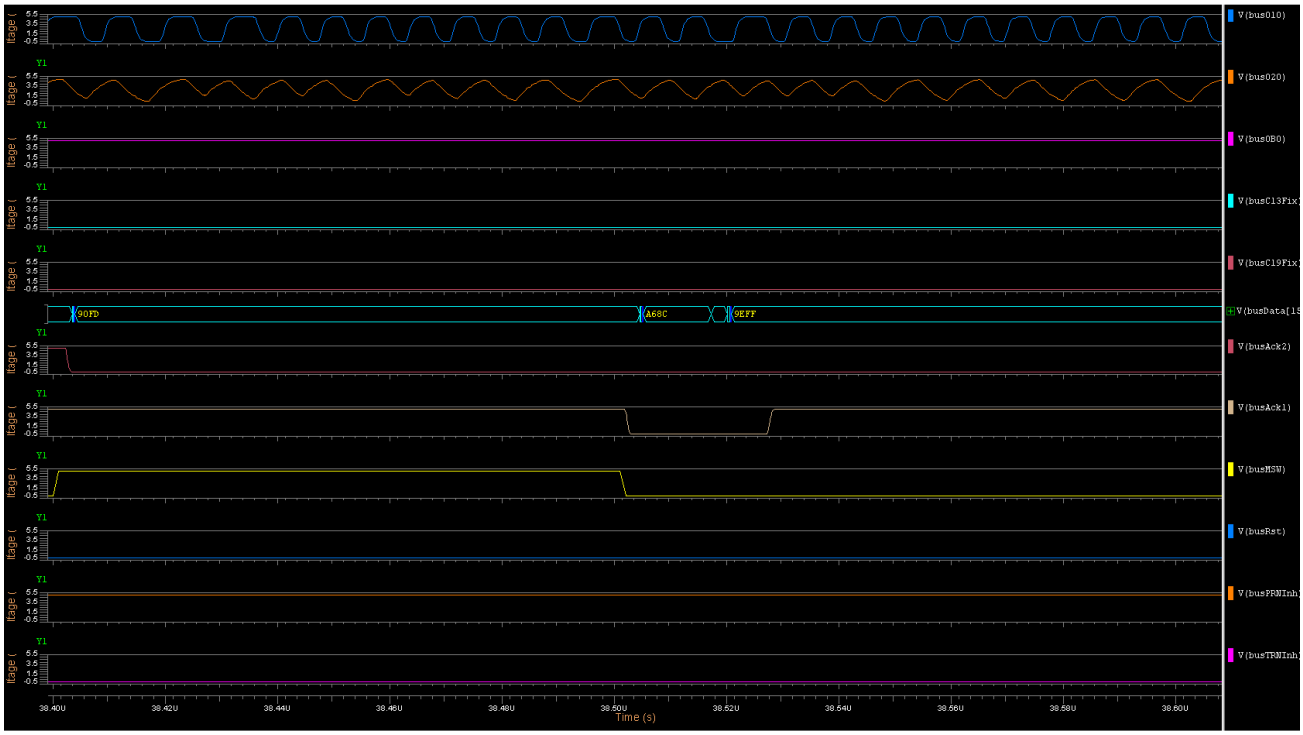


Figure 21: TRNG Chip Simulation with ring oscillators providing clock signals

CHAPTER 4 TRNG TESTING

The TRNG testing was divided into two stages. First, chip functionality was tested to verify its operations of all signals, such as to check the function of each block and then as a whole chip. Once chip testing was done successfully, it was run for a long time to generate and gather the random numbers.

Next, testing was performed on the random numbers generated from the chip. The numbers outputted from the RNG should be random enough to meet the cryptographic standards such as those set by the FIPS and the NIST.

4.1 Test Board for RNG ASIC

To perform the first level of testing, a small printed circuit board (PCB) was designed so that the RNG could be fixed to the test board to run and generate random numbers.

Figure 22 is the block diagram of the test board. This test board containing the RNG is equipped with a voltage regulator which converts 5V input to 3.3V output voltage to provide power supply to the RNG ASIC. It has headers provided to connect four ‘on board oscillator’ outputs of the RNG, so that an oscilloscope can be attached to each oscillator output to verify its oscillations. Another set of headers are used to attach external oscillators to the RNG in the case when ‘on board oscillators’ are not working. There are also headers to connect the RNG output data bus, such that an oscilloscope can be connected to read the output data bits. Finally, there is a circuit which has transceivers and decoders that controls the signals which are necessary to run the RNG ASIC properly. This test board can be connected to any microprocessor to generate and collect random numbers for a longer period.

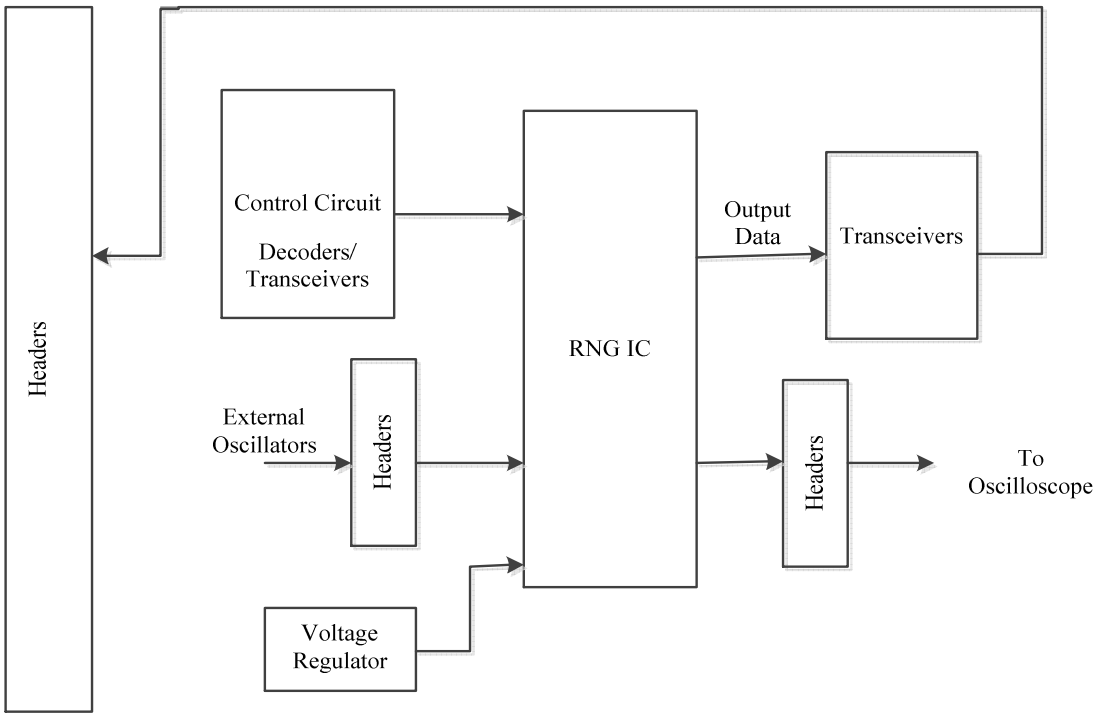


Figure 22: block diagram of the test board

The Mentor Graphics PADS tool was used to design the schematic and layout of the printed circuit board. The PCB was designed as a four layered board: top, bottom, power and ground. To reduce the effect of crosstalk and interference, the top and bottom planes were designed as signal planes, and the power and ground planes were designed on the inner layers. Once the PCB layout was completed, the appropriate Gerber file was generated and sent to 'Advanced Circuits' for the fabrication. After the test board was fabricated, it was then populated. The RNG IC was installed in a socket and could be removed easily when needed.

4.1.1 Hardware Equipment Used for RNG Testing

Gumstix was used to interface with the RNG test board. Gumstix is a company established by Gordon Kruberg in 2003 which manufactures single-board computers. As the name

refers, these computers have a motherboard as small as a stick of gum. These computers are pre-installed with Linux Kernel 2.6.2 operating system. Gumstix also provides a variety of feature-rich expansion components such as general purpose input/output (GPIO), additional serial ports, compact flash cards, USB connectivity, ethernet and different power sources which can be easily connected to a motherboard. Gumstix motherboards come in two different configurations: Overo Earth and Verdex-Pro[1].

In this thesis, the Verdex-pro motherboard was used. It was connected with two other expansion boards: Breakout-vx and Netpro-vx and the whole setup was used to network with the RNG test board. The entire assembly (Verdex-pro motherboard + Breakout-vx + Netpro-vx) is referred to as “gumstix" in this thesis. Figure 23 shows the block diagram of the gumstix setup networked with test board and a personal computer.

---

[1]More information can be found on www.gumstix.com , www.gumstix.org

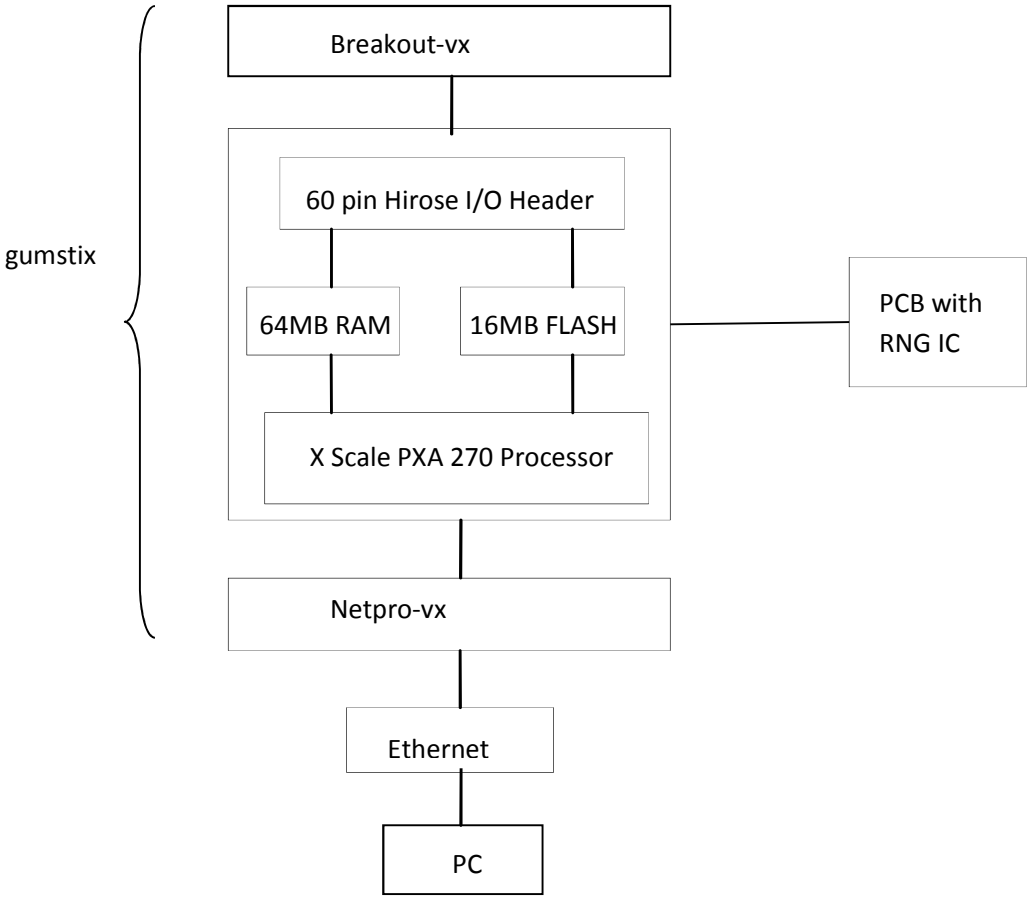


Figure 23: block diagram of the hardware setup to test RNGs

The Verdex-pro motherboard contains a PXA270 microprocessor manufactured by Marvell with Xscale technology running at 400MZ. It also has 64 MB of RAM, 16 MB of Flash and a 60-pin Hirose I/O connector. The Breakout-vx expansion board was connected with the motherboard for easy access to the GPIO lines from the test board. The 'Netpro-vx' was attached to the gumstix motherboard so that the whole gumstix setup can be connected to a PC with Ethernet.

Once the gumstix was connected to the RNG test board and the whole setup was attached to a PC through Ethernet, software was developed to control the RNG and generate and upload the random numbers to a PC.

A total of 24 GPIO pins were used to interface with the RNG test board: 16 GPIO pins were attached to the 16-bit output data bus of the test board, 6 GPIO pins were attached to control signals, and 2 GPIO pins were used to provide power and ground signal to the gumstix.

C code was written to control the gumstix and the test board set up. In order to generate random numbers from the RNG, values were assigned to control signals. Below are the steps that describe how the control values were assigned through programming.

1. To assign the control signals, 6 GPIO pins were set up in the 'OUT' mode and values were allotted to those pins. For example, GPIO 11 is assigned to chip select (CS) signal, and it should be 'Set' to run the RNG chip. Here are the lines of code for this operation:

```
gpio_function (11, GPIO);
gpio_direction (11, OUT);
gpio_set (11);
```

Similarly all the control signals were assigned either 'Set' or 'Reset' using the C code.

2. For reading the generated bit from the RNG, 16 pins of GPIO were put in the 'IN' mode so that the value accepted could be read by knowing the status of the GPIO pins. If GPIO pin is 'Set' (high), the output RNG bit is 1, and when GPIO pin status

is ‘Reset’ (low), the output RNG bit is 0. Here is the example of code to read the generated bit. The GPIO 16 is assigned to read ‘lowest significant bit’:

```
gpio_function (16, GPIO);
gpio_direction (16, IN);
b0 = gpio_status (16);
```

After reading the status of the GPIO 16 that can be either ‘Set’ or ‘Reset’, the value is stored in b0. Similarly, all output data bits can be read.

Once the GPIO pins were allotted in the correct mode and assigned the proper value. As soon as power was switched ON, the RNG started generating random numbers.

The Logic Analyzer was helpful to observe the RNG ASIC functionality and to verify random data generation. The Logic Analyzer was connected with the test board. All of 16 channel of the oscilloscope were attached to the 16-bit output data bus. Figure 24 shows the screenshot of random number generation from the Tektronix MSO 4034 Mixed Signal Oscilloscope.

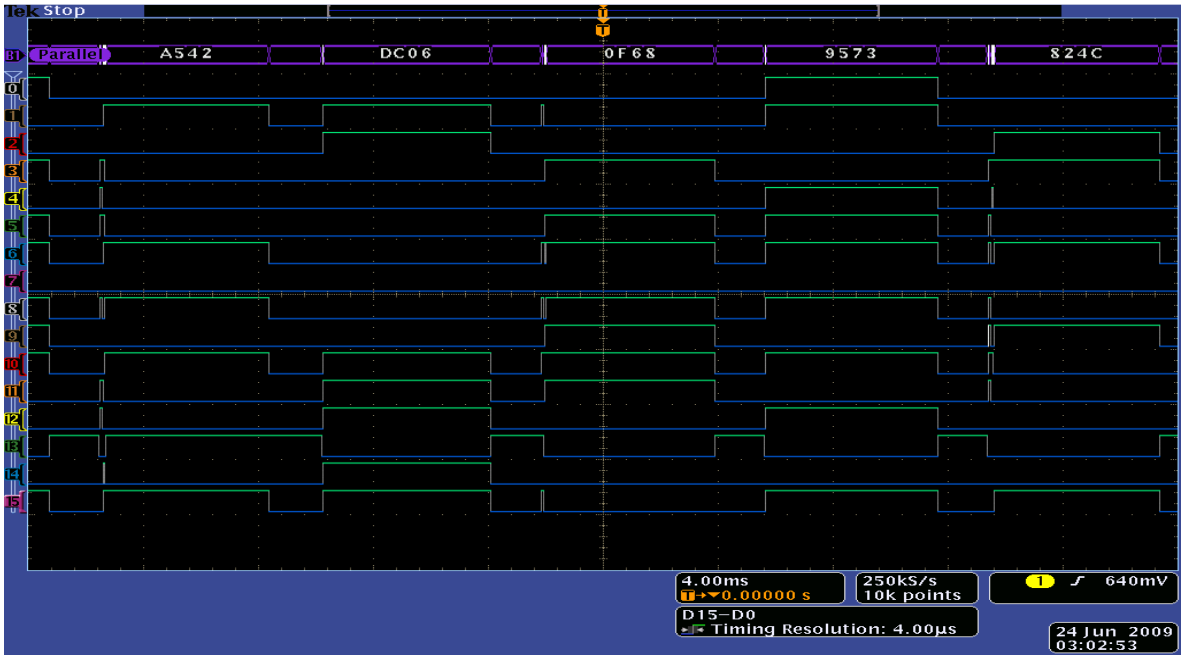


Figure 24: Random Number Generation

The screenshot illustrates that all 16 bits are changing randomly providing first LSB then MSB.

4.2 Randomness Testing of Random Numbers

Various statistical tests are designed to test the extent of the randomness of the random numbers. These tests help to detect any kind of weakness a random number generator might have. For example, the very basic test for random numbers is that the probability of occurrence of 0's and 1's in the sequence should be roughly equal. Too many 0's or too many 1's in the sequence would disqualify the numbers from being random.

A range of tests are developed to test the random numbers which verify for presence or absence of certain 'pattern' and if not detected, considered being nonrandom sequence. The NIST 800-22 test suite has 15 statistical tests to test the randomness of long binary sequence generated by any hardware or software based pseudorandom number generators. If random numbers do not pass the statistical tests, it can be considered as pseudo random

numbers and can be discarded as being true random numbers and if the random numbers do pass the entire statistical test, it can be accepted as truly random numbers. However, these statistical tests provides only probabilistic idea that is, the properties of a random binary sequence can be characterized and illustrated in terms of probability as there is no mathematical proof that binary sequence is truly random [12].

The NIST 800-22 statistical tests are explained below.

1. Frequency (Monobit) Test: This test gives a very fundamental proof of the absence of randomness in a binary sequence; if this test fails probability of failing other statistical tests is high, so it is recommended to run the frequency test first. The test focuses on the amount of ones and zeroes in the entire long sequence. The test fails if total number of 1's and 0's are not approximately equal in the sequence.
2. Frequency Test within a Block: This test focus on the quantity of 1's within blocks. The purpose of this test is to find out whether the occurrence of 1's in a M-bit block is approximately the half of the block size as expected as for truly random sequence. For M=1, this test is equivalent to the Frequency test.
3. Runs Test: This test measures the total number of runs in the sequence, where a run is defined as continuous stream of identical bits (either all ones or all zeros) present in the sequence. A run of length *n* has *n* identical bits that are enclosed with an opposite bit at the start and end of the run. The runs test determines whether the number of runs of ones and zeros of various lengths present in the sequence as expected as for random sequence.
4. Test for the Longest Run of Ones in a Block: This test measures the longest run of 1's within the specified bit block size. The test fails if the length of the longest run

of 1’s within the tested block does not match as expected as in a random sequence. If there is an irregularity in the expected length of the longest run of 1’s, it is most likely to have an irregularity in the expected length of the longest run of 0’s too.

5. Binary Matrix Rank Test: The binary sequence is arranged in rows and columns of matrices. It then calculates the rank of matrix to check for linear dependence among fixed length of bit strings of the original sequence. The test fails if it detects a deviation of the rank distribution from that corresponding to a random sequence.
6. Discrete Fourier Transform (Spectral) Test: The test observes the peak heights in the Discrete Fourier Transform of the binary sequence. The purpose of this test is to find out periodic patterns that are close to each other in the sequence. The test fails if the number of peaks exceeding the threshold is significantly high.
7. Non-overlapping Template Matching Test: The focus of this test is to detect the number of occurrences of pre-specified non-periodic pattern in a specified bit block size. If the pattern is not found, the search begins from the next bit position of the block. If the pattern is found, the block is reset to the bit after the found pattern and the search starts again. The test fails if too many occurrences of the pattern found in the sequence.
8. Overlapping Template Matching Test: The test is similar to the Non-overlapping Template Matching test. The difference from the above test is that if the pattern is found, the block slides only one bit and the search is continued again. The test fails if too many occurrences of the pattern are found in the sequence.
9. Maurer’s “Universal Statistical” Test: The test detects for the number of bits between matching patterns in a binary sequence. The purpose of the test is to detect

the sequence that can be compressed without loss of information. The test fails if sequence is compressible.

10. Linear Complexity Test: This test computes the length of a linear feedback shift register (LFSR) that would be needed to generate the bit pattern. A random sequence should be complex enough to be characterized by longer LFSRs. The test fails if the size of the required LFSR is too small.
11. Serial Test: This test searches for the occurrence of all possible overlapping patterns of specified bits in the entire binary sequence. The number of occurrences of each overlapping patterns should approximately the same. The test fails if the frequency of overlapping pattern is not uniform. For the case of 1-bit patterns, the Serial test is equivalent to the Frequency test.
12. Approximate Entropy Test: This test compares the occurrence of all possible overlapping *m*-bit patterns with *(m+1)*-bit patterns in the entire binary sequence. The test fails if the frequency of overlapping blocks of two consecutive lengths (*m* and *m+1*) is not as expected as for random sequence.
13. Cumulative Sums Test: The Cumulative sum is calculated by transferring the [0, 1] stream to the appropriate [-1, +1] sequence by using $x_i = 2a_i - 1$; where a_i is the original bit pattern. The cumulative sum random walk is derived from partial sums within the new sequence. The test fails if the excursions of the random walk are too large or too small relative to the expected behavior of the random sequences.
14. Random Excursions Test: In this test cumulative sum is calculated by taking a random walk that begins at one point considered to be origin and return to that point. The cumulative sum is calculated in similar way as described in the above

test, the sum is derived after the [0, 1] sequence is transferred to the appropriate [-1, +1] sequence. This test actually examines series of eight tests, such that how many times each of the states: -4, -3, -2, -1 and +1, +2, +3, +4 run into the random walk. The test fails if there are deviations from the expected number of visits to various states in the random walk.

15. Random Excursions Variant Test: This test determines the total number of visits for particular state in the cumulative sum random walk. This test measures deviations for eighteen states [-9, -8, …, -1, +1, +2, …, +9], where the random excursions test walks for only eight states. The test fails if it detects deviations from the expected number of visits to various states in the random walk.

4.3 FIPS Certification

Federal organizations and the community rely on cryptography to protect information and data transfer used in electronic communications. Cryptographic modules are implemented in systems to provide cryptographic service such as confidentiality, reliability, authentication and security requirements. Federal agencies and the community can benefit from the use of tested and validated products as without adequate testing of the cryptographic module can result in insecure consequences.

In 1995, the National Institute of Standards and Technology (NIST) established the Cryptographic Module Validation Program (CMVP) for validating cryptographic modules to the Federal Information Processing Standards (FIPS) 140-1 which is a security requirement for cryptographic modules, and other FIPS cryptography based standards. The

CMVP is a collaborative effort between the NIST and the Communications Security Establishment Canada (CSEC). In 2001, the FIPS 140-2 was released for security requirement for cryptographic modules, and this test standard took over the FIPS 140-1 test standard [13].

United States and Canadian federal agencies accept modules conforming to the FIPS 140-1 and 140-2 for the protection of sensitive information. The secretary of state made adherence to the FIPS standard for the protection of sensitive data mandatory. This standard is applicable to all federal agencies that use cryptographic based security system to protect sensitive information in computer and telecommunication systems including the voice systems [14][15].

Here are the approved pseudo and true random number generators for the FIPS 140-2 [16].

Approved Pseudo Random Number Generators

1. Digital Signature Standard (DSS), Federal Information Processing Standards Publication 186-2, January 27, 2000 with Change Notice –Appendix 3.1.
2. Digital Signature Standard (DSS), Federal Information Processing Standards Publication 186-2, January 27, 2000 with Change Notice –Appendix 3.2.
3. American Bankers Association, Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA), ANSI X9.31-1998 - Appendix A.2.4.
4. American Bankers Association, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), ANSI X9.62-1998 – Annex A.4

5. Recommended Random Number Generator Based on ANSI X9.31 Appendix A.2.4 Using the 3-Key Triple DES and AES Algorithms, January 31, 2005.
6. Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised), Special Publication 800-90, March 2007.

Approved True Random Number Generators

There are no FIPS Approved true random number generators**.**

CHAPTER 5 TEST RESULTS

Long streams of the output numbers of the RNG were gathered and uploaded to a PC for testing the randomness of the random numbers. Once required random numbers were stored statistical tests were performed to determine whether they were random enough to pass the FIPS 140-1, FIPS 140-2 and NIST 800-22 cryptography standard test.

5.1 FIPS 140-1 and FIPS 140-2 Test Results for the RNG ASIC

As the RNG ASIC was verified for its correct functionality, generated random numbers were stored in a binary file and the statistical test suites FIPS 140-1 and FIPS 140-2 were run on the data. C code for FIPS 140-1 and FIPS-2 that was written by Samuel T. Mitchum was used to run binary files.

A total of 625 words of 32-bit (20,000 bits) random numbers were required to run the FIPS 140-1 and FIPS 140-2 test suites. One hundred binary files of 20,000 bits were run for these tests and 80% of them passed the tests successfully. The test results are shown in Figure 25 and Figure 26.

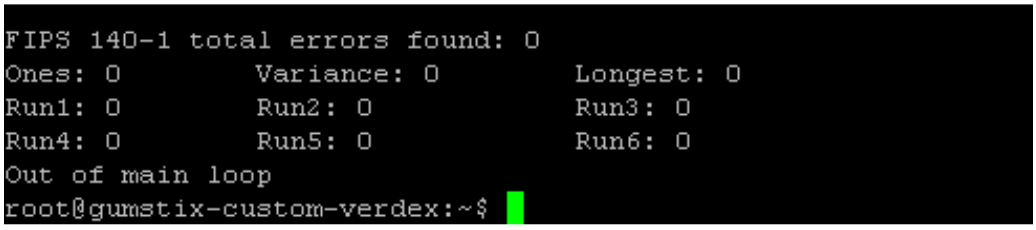


Figure 25: FIPS-140-1 on RNG Data

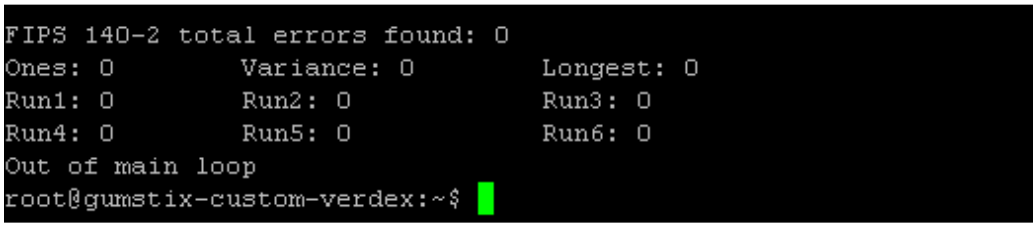


Figure 26: FIPS-140-2 on RNG Data

5.2 NIST 800-22 Test Results on Random Numbers

The NIST 800-22 statistical test suites were run on the data of 100 sets of 2,000,000 bits (total of 200,000,000 bits). Once a total of 200,000,000 (two billion) bits were stored in a binary file, it was given as an input file to the NIST 800-22 test suite software. After the statistical tests code ran successfully, it generated the 'final result analysis' report computing the resulting value for each test. Some of the tests generate multiple values so the average of the total values was computed as the final result for the test. The NIST 800-22 test results of the RNG data are tabulated in Table 6. The test that did not pass the NIST 800-22 is marked with an asterisk.

Test	TRN
Frequency	0.6800 *
Block Frequency	0.6700 *
Runs	0.6700 *
Longest Run of Ones	0.9700
Binary Matrix Rank	0.9400 *
Discrete Fourier Transform	0.9800
Non-overlapping Template	0.9262 *
Overlapping Template	0.9000 *
Maurer's Universal	0.9400 *
Linear Complexity	0.9700
Serial	0.9150 *
Approximate Entropy	0.6800 *
Cumulative Sums	0.6800 *
Random Excursions	0.9872
Random Excursions Variant	0.9954

Table 6: NIST 800-22 Test Results for RNG

The NIST 800-22 statistical tests are useful for studying and evaluating the randomness of the binary sequence produced by the generator. For 100 sets of data, each set having 2,000,000 bits, a passing score suggested by the NIST 800-22 is 96%; that is, the test fails if it scores 95.9% or below whereas it is considered to pass if it scores 96.0% or above.

This RNG passed 5 tests out of the total 15 statistical tests. Failing the above test indicates that the bit patterns found in the sequence are not as consistent as expected for truly

random sequence, meaning that the occurrence of some bit patterns was either more or less frequent than the average.

This chip failed the frequency test and block frequency test, meaning that the binary numbers generated from the chip did not have equal numbers of 1s and 0s in a stream or in the M-bit size block. As described earlier in section 4.2, the RNG outputs should have roughly equal numbers of 1s and 0s in a sequence to pass the frequency test and the block frequency test

Also, this RNG did not generate equal numbers of n-bit length runs (continuous streams of all zeros or ones) as expected as to be random, and failed the 'Runs test'.

The chip failed 'Binary Matrix Rank test', as it detects a deviation of the rank distribution from that corresponding to a random sequence.

The chip did not pass 'Non-overlapping Template' and 'Overlapping Template' tests, meaning that it generated either too many or very few pre-specified non-periodic patterns in the binary sequence.

It failed the 'Maurer's Universal Test,' which means that there were sequences that could be squeezed without losing information.

Failure of the 'Serial Test' indicated that occurrences of all possible overlapping patterns of particular bits in a long binary sequence were not consistent.

It did not pass the 'Approximate Entropy Test' meaning that there were several m-bit and (m+1) bit overlapping patterns present in the entire binary sequence.

Finally, it failed the 'Cumulative Sums Test,' meaning that random walk (partial sums with in binary stream after transforming original [0, 1] pattern into [-1, +1] pattern)

expedition was not as relative as expected for the random sequence. These tests are described in detail in chapter 4.

5.3 Whitening Random Numbers

The process of combining (XOR/XNOR) the outputs of the TRNG with the outputs of a LFSR is termed as 'whitening'. This procedure improves the randomness of the final outputted numbers and is widely used in cryptography. Figure 27 shows the block diagram of the whitening process.

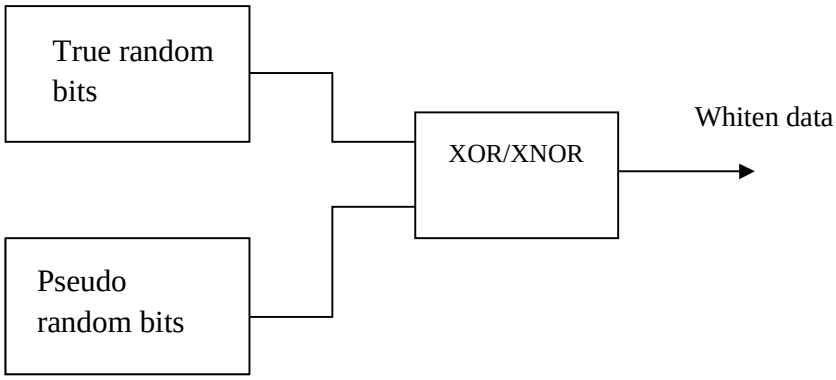


Figure 27: Whitening

'Whitening' is described in [17, section 2.4.2], which states "The linear combination can be realized by adding and subtracting or XORing the two sets. Equation 2 illustrates this principle.

Equation 2 Variance of TRNG XOR PRNG

$$V(x \hat{} y) = V(x) + V(y)$$

Since a TRNG is independent of any LFSR based PRNG by virtue of its definition, Equation 2 shows how the output of a TRNG can be statistically improved by XORing it with the output of an LFSR…….”

In this research, this approach was used to ensure that the RNG data passed all the statistical tests on the RNG data. The random output data were whitened and the whitened data were applied to the NIST 800-22 statistical suites. A 32- bit whitening PRNG has a 32-bit LFSR that clocks the LFSR exactly 32 times between samples. A design of such PRNG is described in [17, section 5.3.1]. Here, C program of whiten PRNG that was originally written by Samuel T. Mitchum was used to whiten the RNG data. The software generates pseudo-random numbers and takes binary file of random bits as input. It then combine (XOR/XNOR) the pseudo random numbers with the random numbers generated from the RNG chip to output final whiten data.

5.4 NIST 800-22 Test Results on Whitened Random Numbers

Whitening was done in two different ways. In the first method, half of the bits of the RNG (the odd bits 1, 3, 5…) were XORed and half of the bits (even bits 0, 2, 4...) were XNORed with the 32-bit maximal length LFSR, which is defined by ‘Xilinx Application’. This whitened RNG data were used to run the NIST 800-22 test suites and the results of the XOR/XNOR whitening data are tabulated in Table 7. The whitened data passed all the tests of NIST 800-22.

Test	Whitened TRN
Frequency	0.9900
Block Frequency	1.0000
Runs	1.0000
Longest Run of Ones	0.9800
Binary Matrix Rank	0.9900
Discrete Fourier Transform	1.0000
Non-overlapping Template	0.9895
Overlapping Template	0.9900
Maurer's Universal	0.9900
Linear Complexity	0.9900
Serial	0.9900
Approximate Entropy	1.0000
Cumulative Sums	0.9950
Random Excursions	0.9952
Random Excursions Variant	0.9929

Table 7: NIST 800-22 Test Results on Whitened RNG

Other deviation from this approach is that the RNG data were whitened by only XNORing the 32-bit random numbers of the RNG with the 32-bit outputs of the LFSR. Whitened data were given to the NIST 800-22 tests and the results are tabulated in Table 8. Whitened data XNORed with the RNG data passed all the tests of NIST 800-22 statistical suites.

Test	Whitened TRN
Frequency	0.9900
Block Frequency	0.9800
Runs	1.0000
Longest Run of Ones	0.9900
Binary Matrix Rank	1.0000
Discrete Fourier Transform	1.0000
Non-overlapping Template	0.9903
Overlapping Template	0.9900
Maurer's Universal	0.9900
Linear Complexity	0.9700
Serial	0.9950
Approximate Entropy	0.9900
Cumulative Sums	0.9900
Random Excursions	0.9848
Random Excursions Variant	0.9890

Table 8: NIST 800-22 Test Results on Whitened RNG

Looking over the test results of whitened data, it can be noted that there is not any significant difference with whitening using either only XNOR or both XOR/XNOR. In both approaches whitening data eliminates the inconsistency of the patterns present in the sequence, and improves the randomness in data to meet random standards sets by the NIST 800-22.

CHAPTER 6 CONCLUSION

6.1 Summary of Work

This thesis focuses on designing a digital true random number generator. The methodology used is based on standard digital blocks that can be synthesized from the VHDL description. The Mentor Graphics tools were used to design schematics and layouts of the custom IC. The MachTA simulation tool was used for post-layout simulation of full chip design. Once chip design was simulated successfully, it was sent to MOSIS for the fabrication.

The test board was designed to collect random numbers from the fabricated RNG IC. This board can be attached to any microprocessor. Here, it was attached to the gumstix, which had a pre-installed Linux operating system. Software was developed in C language to operate the fabricated RNG chip, perform functional testing, generate random numbers and upload generated random numbers to a PC.

Once the RNG IC was verified as functionally correct, it was run to generate random bits. The data set of 100 files, each having 2,000,000 bit (a total of 200 million bit), were collected and uploaded to a PC. The statistical test suites were run on the data to test the quality of randomness. Initially, the chip passed 5 tests of the NIST 800-22, to resolve this, whitening was performed on the RNG data to clear up the inconsistency in the output binary pattern. The whitened data did pass all 15 tests of NIST 800-22.Designing a VLSI chip from high-level VHDL code and using standard cells to implement the design was a good learning experience for real-world applications, as full-custom chip design is rare.

6.2 Future Work

This RNG design utilized the 19-bit and the 13-bit LFSR to generate 32-bit random numbers; the combination was chosen arbitrarily, and more interesting research can be done in the area of generating the 32-bit random numbers by using different combination of LFSRs. Another possibility for future work would be to design a unique ring oscillator for each LFSR to provide clocks such that ring oscillator operates at a different speed (fast/slow) or different in design (big/small). In this design similar ring oscillators were used to clock the LFSRs.

In this thesis, the test board in conjunction with the gumstix was used to generate and collect random numbers for testing the RNG IC. There are several other possibilities for interfacing the RNG chip with any other microcontroller or FPGA platform to generate and test random numbers.

REFERENCES

[1] Landon Curt Noll, Robert G. Mende, Sanjeev Sisodiya, "Method for Seeding a Pseudo-Random Number Generator with a Cryptographic Hash of a Digitization of a Chaotic System", US patent 5,732,138, January 29, 1996

[2] John Koeter, "What's an LFSR?", Texas Instrument, December 1996

[3] Benjamin Jun and Paul Kocher, "The Intel Random Number Generator", Cryptography Research, Inc. White Paper Prepared for Intel Corporation, April 22, 1999

[4] Greg Taylor, George Cox "Behind Intel's New Random-Number Generator" IEEE Spectrum, September 2011

[5] Greg Taylor, George Cox,"Intel Makes a Digital Coin Tosser for Future Processors", IEEE Spectrum, September 2011

[6] Mitchum, S and R. H. Klenke, "FPGA Based Digital True Random Number Generator", *2009*

[7] Mitchum, S.T.; Klenke, R.H., "Divergent path random number generators", Southeastcon, 2009. SOUTHEASTCON '09. IEEE, March 2009, Page(s): 109 – 114

[8] Pierre-Yvan Liardet, "Random Number Generating Circuit and Process" US Patent 6581078, January 4, 2000

[9] Katsunori Ikake, "Random Number Generator", US Patent 7124157, June 7, 2002

[10] Kohlbrenner, Paul and Alfke, Peter, "Efficient Shift Registers, LFSR Counters and Long Pseudo-Random Sequence Generators", *Xilinx Publication XAPP052,* July 7, 1996

[11] Mitchum, S.T.; Klenke, R.H., "Design and Fabrication of a Digitally Synthesized, Digitally Controlled Ring Oscillator", Proceedings of the Third IASTED International Conference on Circuits, Signals and Systems, October 24-26, 2005.

[12] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert, James Dray, San Vo, "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications" *National Institute of Standards and Technology publication* 800-22, revised: April 2010

[13] Computer Security Division Computer Security Resource Center,

"Cryptographic Module Validation Program" *National Institute of Standards and Technology available at* http://csrc.nist.gov/groups/STM/cmvp/

[14] FIPS PUB 140-1: Federal Information Processing Standards Publication *"Security Requirements for Cryptographic Modules", U.S. Department of Commerce and National Institute of Standards and Technology,* January 11, 1994

[15] FIPS PUB 140-2: Federal Information Processing Standards Publication (Supersedes FIPS PUB 140-1, 1994 January 11)*"Security Requirements for Cryptographic Modules", U.S. Department of Commerce, Technology Administration and National Institute of Standards and Technology,* December 03, 2002

[16] Annex C: Approved Random Number Generators for FIPS PUB 140-2, "Security Requirements for Cryptographic Modules" *U.S. Department of Commerce and National Institute of Standards and Technology,* July 26, 2011

[17] Samuel Theodore Mitchum, Junior, "Digital Design and Implementation of True Random Number Generators", December 2010