An Embedded True Random Number Generator for FPGAs

Paul Kohlbrenner
Lockheed Martin
3201 Jermantown Road
Fairfax, VA 22030, USA
Paul.W.Kohlbrenner@lmco.com

Kris Gaj
George Mason University
4400 University Drive
Fairfax, VA 22030, USA
kgaj@gmu.edu

ABSTRACT

Field Programmable Gate Arrays (FPGAs) are an increasingly popular choice of platform for the implementation of cryptographic systems. Until recently, designers using FPGAs had less than optimal choices for a source of truly random bits. In this paper we extend a technique that uses on-chip jitter and PLLs to a much larger class of FPGAs that do not contain PLLs. Our design uses only the Configurable Logic Blocks (CLBs) common to all FPGAs, and has a self-testing capability. Using the intrinsic jitter contained in digital circuits, we produce random bits at speeds of up to 0.5 Mbits/second with good statistical characteristics. We discuss the engineering challenges of extracting random bits from digital circuits, and we report the results of running standard statistical tests (NIST) on the output generated by our system.

Categories and Subject Descriptors

G.3 [**Probability and Statistics**]: Random Number Generation; E.3 [**Data Encryption**].

General Terms

Algorithms, Design, Experimentation, Security.

Keywords

RNG, TRNG, Cryptographic, Random numbers, FPGA.

1. INTRODUCTION

The need for random numbers in cryptographic processes is ubiquitous. Initialization vectors, block padding, challenges, nonces, and, of course, keys are some of the cryptographic objects where a string of unpredictable bits is required. Often the same Random Number Generator (RNG) supplies bits for all of the above uses in a cryptographic system. Many of the bits generated by the RNG are transmitted in the clear and thus a passive attacker has ample opportunity to analyze the output of the RNG and can leverage any weaknesses found there. RNGs used for cryptographic processes must, therefore, be considered a critical part of the cryptographic system. A weakness or failure in the RNG can lead to a complete failure of the system [4].

One well-known example of a successful attack on a weak RNG is the infamous Netscape V2.0 browser attack [8]. The engineers at Netscape used the system clock as a source of randomness. This proved to be insecure. The RNG was used to generate the keys needed for the SSL protocol and thus the browser could not fulfill its promise of secure transfer of data.

1.1 Kinds of Random Number Generators

RNGs can be separated into two general categories [15]:

- **Pseudo Random Number Generators** (PRNGs): These generators are algorithms, which are initialized with an externally generated sequence and produce a much longer sequence that appears to be random. After being initialized with a seed value the internal state of the generator completely determines the next bit to be generated. Given the same seed value a PRNG will always produce the same sequence.
- **True Random Number Generators** (TRNGs): These generators base their output entirely on an underlying random physical process. Unlike their deterministic cousins there is no internal state kept in the generator and the output is based only on the physical process and not any previously produced bits. Often the raw bits generated by the physical source are biased (the probability of a '1' is not 0.5), and thus some bias reduction is necessary.

In many cases it is possible to combine the two kinds of RNGs and produce a useful hybrid [11] [20]. In cases where the output rate of a physical source of random bits is lower then the desired output bit rate of the RNG it is possible to periodically re-seed a PRNG with bits from the TRNG to achieve an acceptable output. As long as the PRNG's expansion of the physical bit source is complex enough that an attacker cannot feasibly reverse engineer it (e.g. a one-way hash function such as SHA1) the hybrid is considered cryptographically secure.

2. PREVIOUS WORK

Electronically generating random bits has been attempted for many years. In 1946 ATT was issued US patent 2406031 for a device that produced random bits on five-bit paper tape. The

FPGA'04, February 22–24, 2004, Monterey, California, USA
Copyright 2004 ACM 1-58113-829-6/04/0002…$5.00.

tapes were used to encrypt Teletype traffic (the source of randomness was a large container of white and black balls).

The source of randomness for our TRNG is two ring oscillators. Using free running oscillators as a source of randomness has a long history. An early use was to produce the bits for the RAND Corporation's 1955 book titled A Million Random Digits with 100,000 Normal Deviates [17]. Other uses include the TRNG described in [6] and modeling work covered in [16]. More recent examples of this technique include the TRNG built into Intel PC chipsets [10] and the TRNG used in the VIA Technologies Nehemiah processor core [2].

2.1 Field Programmable Gate Arrays

Field Programmable Gate Arrays (FPGAs) are emerging as an attractive platform for cryptographic implementations. Now fast enough and large enough to implement any cryptographic algorithm they offer benefits such as:

- Near-ASIC encryption speeds
- Algorithm and resource efficiencies
- In service algorithm modification
- Low development costs
- Parameter and algorithm eraser on intrusion detection

Until recently, FPGA designs that included a cryptographic component and required a source of random bits had limited options. The designer could use any of a variety of special purpose TRNG chips and make the necessary physical connections. However, these external interconnections are weak points that an attacker could observe and exploit. Or the designer could implement a PRNG in the FPGA and suffer the resulting degradation in security [1]. In [7] a third option was introduced. By carefully engineering the frequency of two clock signals the non-deterministic jitter present in all digital signals could be extracted.

Jitter is defined by the ITU-T as the variations in the significant instants of a clock or data signal [9]. Jitter in digital circuits has many sources including semiconductor noise, cross talk, power supply variations, and electro-magnetic fields in the operating environment. Semiconductor noise is the non-deterministic component that we based our construction on. There are several ways to characterize jitter. Period jitter, which is the measure of deviation in a clock's period from its average period, is shown in Figure 1 [21].

The extraction technique presented in [7] is to use one clock signal to sample the value of a second clock signal on each cycle. If the two clock frequencies are slightly different, the point sampled in the second signal will advance through the second signal's cycle. If the change is small enough it will eventually sample the second signal in the jitter zone. Thus the sampling will produce a large number of deterministic bits and at least one uncertain bit taken in the jitter zone. XORing the deterministic bits and the non-deterministic bit(s) produces a single random bit.

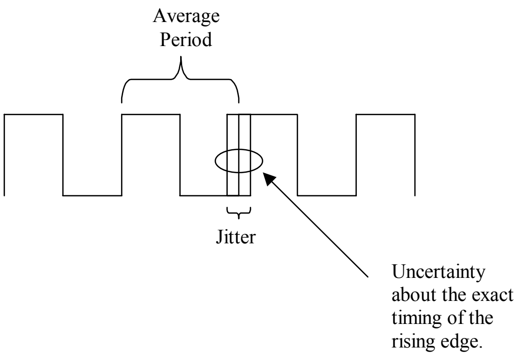

Figure 1 - Period Jitter

A Phase Locked Loop (PLL) present on Altera FPGAs was used to produce the two clock signals used in this technique. The PLL synthesized the new signals from the system clock. A PLL is a device that contains an oscillator whose frequency is adjusted such that there is no phase difference between it and the input clock signal. PLLs in FPGAs have two primary uses:

- Reduce clock skew in large clock distribution nets.
- Frequency synthesis.

Frequency synthesis is accomplished by modifying the oscillator signal before it is fed to the phase detector thereby causing the internal oscillator to increase or decrease the frequency of its output signal. Very fine control of the output frequency of the PLL is possible.

Xilinx is the largest manufacturer of FPGAs. With a 44% [22] share of the Programmable Logic Device (PLD) market segment and a broad line of FPGAs and other programmable logic devices Xilinx is often the choice of system architects. Unfortunately (for our application), Xilinx FPGAs mostly provide Delay Locked Loops (DLLs) instead of PLLs. A DLL inserts delay elements into the path of the clock signal until the phase difference of the incoming clock and a one cycle delayed clock is zero. While DLLs work well to reduce clock skew (their primary function), they cannot provide the fine control over frequency synthesis necessary for our application. For this technique to work the difference between the input and output frequency must be on the order of 0.1%; this is not possible with current DLL technology.

This paper seeks to extend the technique in [7] to PLL-less FPGAs.

3. OUR PROPOSED DESIGN

3.1 Overview

Our proposed design as shown in Figure 2 consists of two independent and identically configured ring oscillators, a sampling circuit, and a control circuit.

The two ring oscillators each supply a stream of pulses to the sampler unit. The frequency of the two clock signals is chosen to be close but not identical. The sampler unit uses one clock signal to sample the other clock signal. The stream of samples consists of a run of ones and a gap of zeros. The length of this run and gap is counted modulo 2 and output as a random bit.

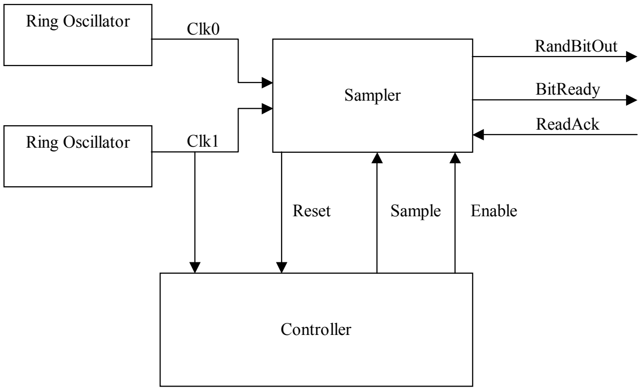

Figure 2 - Overall Design

3.2 The ring oscillators

The output from our ring oscillator is a stream of regular pulses.

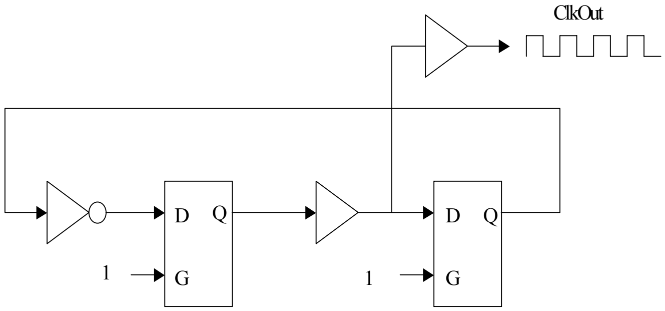

Figure 3 - Ring Oscillator Components

Our ring oscillator consists of a buffer, two transparent latches, and an inverter configured serially to feed back on itself. The buffer and the transparent latches add propagation delay. The sum of the propagation delays p_i through the various elements of the oscillator determines the nominal output frequency f of the circuit.

$$f = \frac{1}{2\sum_{i-0} p_i}$$

By taking care to have the end-to-end propagation delay of the circuit well above the inertial delays of the individual elements, the stability of the circuit can be assured.

We tested several different configurations for our oscillator. The output bit rate of our TRNG is directly related to the frequency of the ring oscillator, thus a high oscillator frequency was desirable. We believed that a target frequency for our oscillator of 150MHz would allow us to easily create counters and other control logic to test the design without the need for substantial logic optimization while still providing reasonable output bit rates. The ring oscillator propagation delay implied by 150MHz is:

$$\frac{1}{2*150{,}000{,}000} = 3.3ns$$

The propagation delay through the average gate or latch is approximately 0.4 ns and the wire delay from the output of the CLB back to the input of the CLB is about 1.0 ns. Thus we quickly settled on the design shown in Figure 3.

By design this configuration exactly fit in one Virtex CLB slice. The output of both latches is routed externally from the output of the CLB back to its inputs. The output of the oscillator is taken from the output of the buffer.

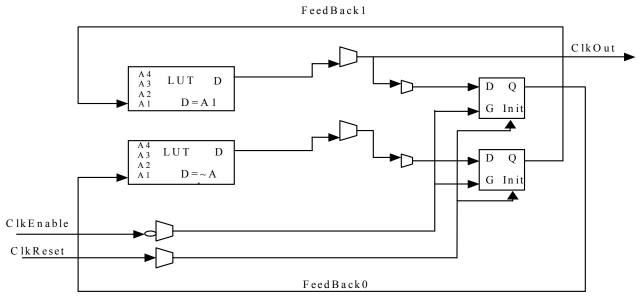

Figure 4 - Ring Oscillator CLB Layout

The end-to-end propagation delay through this particular circuit was found to be approximately 3.7 ns, which resulted in an output frequency of 130MHz. We note that the individual components of the circuit each experience a change of input every 3.7 ns, which is almost an order of magnitude greater then their inertial delays. In our testing, these ring oscillators were found to be very reliable. CLB ring oscillators, such as the one above, have an intrinsic natural difference in speed. These differences are due to small variations in the physical characteristics of the CLBs. We expand later in this paper on the effects of placement of the ring oscillator CLBs on the FPGA and the effects of temperature on the speed of the ring oscillator.

3.3 The Sampler

The sampler circuit extracts the jitter contained in the signals from the two ring oscillators.

As shown in Figure 5, at the input of the sampler circuit a D type flip flop uses the CLK1 signal to sample the CLK0 signal. The output of the sampling process (without jitter) is illustrated in Figure 6. The signal S0 will be high so long as the rising edges of CLK1 occur during the high portion of CLK0. Once CLK1's rising edge starts sampling the low portion of CLK0 the S0 signal will transition to a '0'.

Figure 7 shows what happens as the sampling point moves through a signal with jitter. The jitter in the CLK0 signal will be captured and expressed as a change in the cycle length of the S0 signal. In our sampler circuit we set up a one-bit counter to count cycles in the CLK1 signal (signal C0). By using the S0 signal to latch the value of the C0 counter we can convert the Least Significant Bit of the length of the S0 signal to a single random bit (RandOut). The S0 signal is also used to notify the user of the TRNG that a new random bit is ready. One key advantage to using this technique is that it captures the essential random element (the cycle length uncertainty) and very simply presents it as a single random bit without having to have a priori knowledge of the frequencies involved.

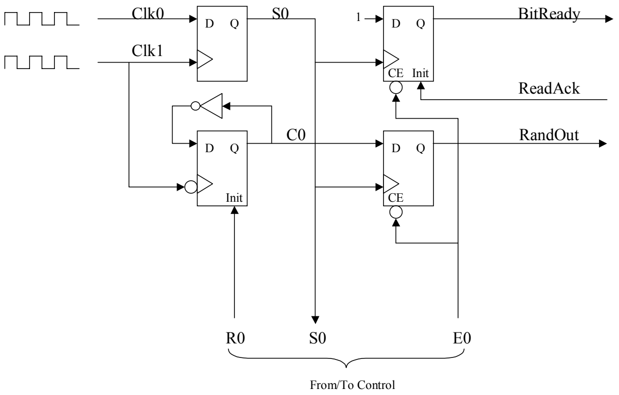

Figure 5 - Sampler Circuit Design

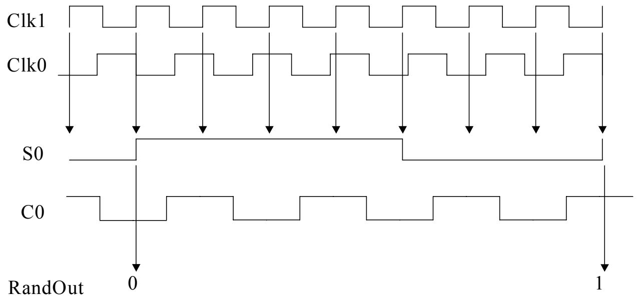

Figure 6 - Wave Diagrams for the Sampler Circuit

3.4 The control circuits

The description in the previous section only considered the jitter on the CLK0 signal. Since our ring oscillators are identically constructed it is reasonable to assume that they have similar amounts of jitter. In cases where the difference in the cycle lengths of the two clock signals is very small it is possible for the S0 signal to transition several times before settling down to a stable value. Without the control circuit, the sizes of these small S0 cycles will be counted and presented as (very correlated) random bits.

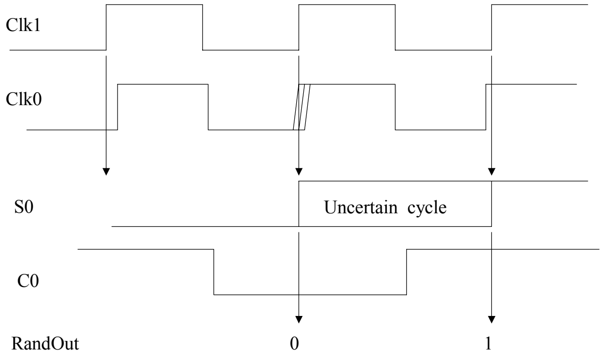

Figure 7 – Sampler Circuit's Behavior With Jitter

The output of the control circuit drives the Clock Enable (CE) inputs of the control and output flip-flops. The control circuit disables these devices immediately after a random bit is latched into the output flip flop. The control circuit enables the clock inputs on these two devices only after it has counted a preset number of CLK1 cycles that have sampled the low portion of the CLK0 signal. In this way it forces the sampler circuit to ignore the short S0 cycles that occur on both the rising and falling edge of S0.

The control circuit also resets the one bit counter after each random bit is latched. We do this to eliminate any correlation between successive bits.

An important secondary benefit of the control circuit is that it prevents any output from the TRNG if the difference between the cycle lengths of the two ring oscillators is too great. If the difference in the cycle time of CLK0 and CLK1 is greater then the width of the jitter zone then some S0 cycles will not contain a sample that includes jitter. Larger cycle time differences also produce fewer samples of CLK0. The control circuit will never enable the output flip-flops if there are too few cycles of '0' in the S0 signal. Detecting the failure of the internal source of randomness is a required function of a TRNG [19].

4. RING OSCILLATOR ISSUES

4.1 Good Ratios

A result of our research was the discovery of a wide variation in the intrinsic speed of ring oscillators in an FPGA. We found a 7% difference between the normalized speeds of the slowest CLB and the fastest CLB. Our technique requires a pair of ring oscillator with closely matched frequencies. Thus not all pairs of ring oscillator are suitable. One difficulty in measuring the speed of a ring oscillator is that they are very sensitive to the temperature of the FPGA. We found that by simultaneously measuring the speed of a reference ring oscillator and the ring oscillator under test we could normalize the speeds and build a database of CLB speeds. Using this database we can predict which pairs of CLBs would produce ring oscillators with a desired speed ratio.

The frequency of a ring oscillator tends to wander as the temperature of the chip varies. Even on a mostly empty FPGA there is a several second period in which the speed of the ring oscillator decreases as the area around the ring oscillator CLB heats up. In fact, this vary trait is used in [13] to measure the temperature of various parts of an FPGA. It is important that frequency of the two ring oscillators does not wander apart due to temperature differences on the chip. For this reason we found that it is important to place the two ring oscillators close to each other.

In order to overcome this placement sensitivity we created a design that consisted of four ring oscillators that were individually sampled by a fifth ring oscillator. The four bits produced by the sampler circuits were XOR'ed to produce a single output bit. In a test that involved placing and testing the circuit in 70 locations across the FPGA we saw evidence of poor statistical properties in only four of the placements (a 95% success rate). We expect that a larger number of ring oscillators can further increase the probability of a successful first-time placement.

4.2 Evidence of Jitter

During the development of this technique we considered the possibility that the ring oscillators do not produce signals with jitter. The apparent random output would then be just a complicated, but deterministic, combination of the two signals. Depending on the exact relationship of the signals from the two ring oscillators the sampler circuit would either produce S0 cycles of a single length (resulting in an output of either all zeros or all ones) or S0 cycles that alternated between two lengths (resulting in both ones and zeros but in a repeating pattern) [12]. To explore this argument we built the sampler circuit described earlier and added a counter to record the full length of the stream of S0 bits produced by the sampler flip flop. The resulting analysis of this data is presented below. Using a pair of ring oscillators with an average frequency of 130MHz and a cycle difference of 35 ps we obtained the distribution of cycle lengths shown in Figure 8.

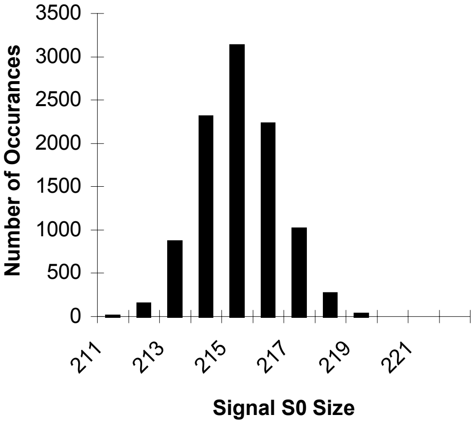

Figure 8 - Signal S0 Cycle Lengths for a Frequency Difference of 35ps

What is most striking about Figure 8 is that it contains nine different cycle lengths.

A second set of data, presented in Figure 9, provides an even more dramatic demonstration of the underlying randomness being extracted. The distribution in Figure 9 was produced with 130MHz oscillator with a cycle length difference of 22ps.

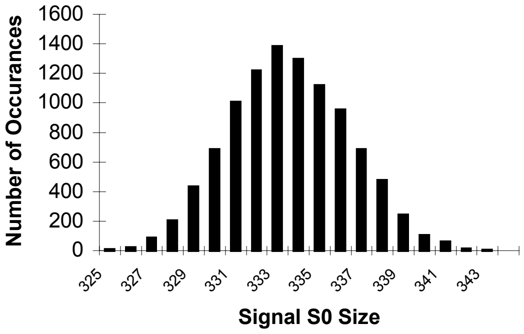

Figure 9 - Signal S0 Cycle Lengths for a Frequency Difference of 22ps

We note that the cycle length of the S0 signal in Figure 9 ranges from 322 cycles to 342 cycles and is centered around 332 cycles. This kind of variation could only be produced by a variation in the length of the ring oscillator cycles themselves (aka jitter).

4.3 Bias in the Output

Ring oscillator combinations that produce a small number of cycle length differences (such as in Figure 8 above) sometimes have a detectable bias in the resulting random bit output. While the bias is quite small (approximately 0.5%) this is enough to cause failures in the standard randomness tests. The source of bias is the limited number of different bit lengths of the S0 signal. Our extraction technique assigns a one to even length S0 cycles and a zero to odd length S0 cycles. Thus the ratio of odd to even length S0 cycles is directly reflected in the bias of the resulting random bit stream.

A second source of bias is the occasional meta-stable output from the sampling flip-flop in the sampler circuit. When a flip-flop is forced to latch a changing value its output sometimes becomes metastable, that is it is neither a zero nor a one. This property is a source of randomness in its own right and is being investigated as the basis for a TRNG [5]. In our case it appears the current state of the output flip-flop in the sampler circuit influenced the resolution of the meta-stable state. By adding a buffer to the S0 signal we were able to attenuate this source of bias.

Cryptographically secure TRNGs should have no detectable bias. Consequently we recommend all TRNGs constructed using this technique use a bias reduction method.

There are several well-documented ways to reduce bias [3]. Two popular ones are:

- XOR Reduction – With this method successive pairs of bits are XORed together. This will reduce the bias of uncorrelated bits in the following way:

Assume the probability of a one is p and therefore the probability of a zero is $(1-p)$. The probability that the XOR process will produce a one is

$$P(X=1)=2p(1-p)=2p-2p^2$$

and the probability that the XOR process will produce a zero is:

$$P(X=0)=p^2+(1-p)^2=2p^2-2p+1$$

Table 1 shows the power of this technique:

Table 1 - XOR Bias Improvement

Original Probability		New Probability	
One	Zero	One	Zero
0.7	0.3	0.42	0.58
0.6	0.4	0.48	0.52
0.55	0.45	0.495	0.505
0.51	0.49	0.4998	0.5002

It is possible to XOR more than two bits and obtain an even greater improvement. The downside of this technique is that the output bit rate is reduced. If the bits are correlated the XOR technique should not be used as the output bias will be substantially increased.

- A von Neumann corrector [4] (a.k.a. a von Neumann Whitener) – For this technique consecutive non-overlapping pairs of bits are examined, if they are different output the first bit, if they are the same discard both and output nothing. This completely eliminates the bias but at the cost of a potentially significantly reduced output bit rate. It will also fail if there are correlations between successive bits.

In our tests we saw no evidence of correlations in the output stream and thus we used an XOR reduction technique [3]. Although we performed the bias reduction on the bits we collected on the host computer, in future implementations we will design this additional step into the hardware itself.

4.4 Bit generation speeds

Our technique is specifically designed to overcome the variability of ring oscillator frequencies. By using the low-order bit of the size of the S0 cycle as our random bit our machine will work even as the speeds of the ring oscillator change. The output rate of the TRNG is dependent on both the mean frequency of CLK1 F_{c1} and the difference between the cycle times of the ring oscillator T_{c0} and T_{c1}.

$$OutputRate=\left(\frac{|T_{c0}-T_{c1}|}{T_{c0}}\right)F_{c1}$$

For example, a pair of 130MHz ring oscillator with a cycle time difference of 35 ps:

$$\left(\frac{|35ps|}{7692ps}\right)130{,}000{,}000=591{,}523bps$$

Note that the XOR bias reduction will reduce this rate by a factor of two.

5. TESTING

The design and testing of this TRNG was done on a SLAAC-1V FPGA test system. The SLAAC-1V board contains three Xilinx Virtex XCV1000 FPGAs, which can communicate with each other and with the host computer. The SLAAC board host machine is a Dell Optiplex running RedHat Linux. Our VHDL development system is Windows based. We use Synplify V7.2 and the Xilinx ISE-4.2 tool set to produce bit streams for the SLAAC board.

The ring oscillator pairs used to produce the graphs in the previous section were each used to produce a 1Gbit (128 MByte) sample file of random bits.

There are several substantial test suites for testing TRNGs [14] [18] [19].

We present results from the US National Institute of Standards and Technology (NIST) Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications [18].

The NIST test suite produces a summary report for each file of random bits it tests. The table that follows is a result of running the NIST suite over the set of data produced by our TRNG. The tables consist of ten columns labeled C1 through C10, A P-VALUE column, a PROPORTION column, and a column containing the name of the test for that row.

Each test in the NIST suite is run over a large number of sets of bits from the file to be tested. The statistic that is generated from each of these runs is called a P-Value and it represents ... *the probability that a perfect random number generator would have produced a sequence less random than the sequence that was tested.* [18] For example, if you got a P-Value of 0.95 this would mean that 95% of the sequences produced by an ideal RNG would look less random than your sequence Thus, very small P-values are bad.

With these kinds of tests one expects to get a range of P-values (in fact, it is bad if you don't get a range). The range from 0 to 1 is divided into ten bins, labeled in this report C1 through C10. The number in each of these columns represents the number of tests that had a P-value in the corresponding range. We would expect that a perfect RNG would have P-values evenly spread over the range 0 to 1. The column labeled P-VALUE is a chisquare test on the preceding spread of P-values over the range 0 to 1. It is a P-value of P-values. The documentation that accompanies the suite indicates that: “If P-Value [the number in the column labeled P-VALUE] >= 0.001, then the sequences can be considered to be uniformly distributed”.

The PROPORTION column indicates the number of P-values that were above the 0.01 confidence interval. It is acceptable for a few individual tests to fail. The test suite will indicate a problem by flagging the PROPORTION number with an "*".

In our case, none of these tests indicated failure

Table 2 - NIST Results

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------


------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
117 114 122 108 103 105 104  91 109  99  0.657545   0.9888    Frequency
114  90 120 113 106 101 107 120 108  93  0.452068   0.9879    Block-Frequency
118 107 113 130  97 112  93 106  99  97  0.278926   0.9888    Cusum
111 121 113  94 118 121  92  92 114  96  0.178887   0.9907    Cusum
121  89 101 116 112  93 118 118 111  93  0.198956   0.9944    Runs
108 115 108  97 105 115  99 122 100 103  0.789171   0.9907    Long-Run
102 113 116 107 101  85 119 119  99 111  0.381587   0.9916    Rank
 97 133 122 130 106 101  90 104 104  85  0.008761   0.9944    FFT
101 102  98 111 115 102 120 105 107 111  0.906423   0.9879    Aperiodic-Template
105  93  96 124 116 110 100 110  92 126  0.183510   0.9925    Periodic-Template
123 109 111 102  98 106 110 103 104 106  0.917455   0.9860    Universal
123 107  96 106  99 100 125 122  98  96  0.236353   0.9888    Apen
 67  65  71  60  60  63  70  53  59  65  0.890466   0.9842    Random-Excursion
 60  61  70  68  77  56  64  54  63  60  0.666838   0.9921    Random-Excursion-V
100 101 117 121 110 105  95 101 111 111  0.773062   0.9888    Serial
102  81 130 142  96 102 103 127 106  83  0.000105   0.9907    Lempel-Ziv
107 108 113 111 104 101 101 111 115 101  0.984963   0.9841    Linear-Complexity
```

6. FUTURE WORK

During the design of our TRNGs we noted several opportunities to increase the bit rate of the device. One obvious change would be to increase the speed of the ring oscillators. This has the disadvantage of requiring oscillators more closely matched in clock period. A second idea we had was to generate random bits from both the rising edge of the S0 signal (as we do now) and the falling edge of the signal. This would double the output of the TRNG.

Increasing the number of ring oscillators that produce bits appears likely to overcome the problem with finding matched CLB slices. There is some shared logic in using groups of ring oscillators and thus we found that each additional ring oscillator added to a design consumes four CLB slices.

Finally, by adding a counter to the S0 signal it would be possible to create a real-time “noise-failure” alarm that would allow the TRNG to signal a failure in the randomness extraction mechanism.

7. CONCLUSION

We believe that our construction is a useful addition to the expanding use of FPGAs in cryptographic systems. Being able to fully contain a TRNG within the FPGA increases the overall security of the system. By not requiring special resources within the FPGA (e.g. a PLL) we increase the universe of designs that can make use of this way of extracting jitter to make random bits.

Finally, our design has a built in mechanism to halt bit output on failure of the source of randomness.

8. REFERENCES

[1] Chu, P., P., Jones, R., E., Design Techniques of FPGA Based Random Number Generator, Military and Aerospace Applications of Programmable Devices and Technologies Conference, The Johns Hopkins University- Applied Physics Laboratory, September 1999.

[2] Cryptography Research Inc., Evaluation of VIA C3 Nehemiah Random Number Generator. Technical Report, Revision Dated: February 27,2003, Available at: http://www.cryptography.com/resources/whitepapers/index.html

[3] Davies, R., Exclusive Or (XOR) and Hardware Random Number Generators. Feb 28, 2002, Available at: http://www.robertnz.net.

[4] Eastlake, D., Crocker, S., Schiller, J., 1994. Randomness Recommendations for Security - RFC 1750, Available at: http://www.faqs.org.

[5] Epstein, M., Hars, L., Krasinski, R., Rosner, M., Zheng, H., Design and Implementation of a True Random Number Generator Based on Digital Circuit Artifacts, Proceedings of the 5th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2003), Springer-Verlag, LNCS 2779 (2003).

[6] Fairfield, R., C., Mortenson, R., L., Coulthart, K., B., An LSI random number generator (RNG), Advances in Cryptology - Proceedings of Crypto 84, Springer-Verlag, LNCS 196 (1985), Eds: G.R. Blakley and D. Chaum, pp. 203-230.

[7] Fischer, V., Drutarovský, M., True Random Number Generator Embedded in Reconfigurable Hardware, Proceedings of the 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002), Springer-Verlag, LNCS 2523 (2002).

[8] Goldberg, I., Wagner, D., Randomness And the Netscape Browser, Dr. Dobb's Journal, January 1996.

[9] International Telecommunication Union. Series G: Transmission Systems and Media: Definitions and terminology for synchronization networks - ITU-T Recommendation G.810, (08/96).

[10] Jun, B., Kocher, P., The Intel Random Number Generator, White Paper Prepared For Intel Corporation. by Cryptography Research Inc. Available at: http://www.cryptography.com/resources/whitepapers/index.html

[11] Kelsey, J., Schneier, B., Ferguson, N., Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator, Sixth Annual Workshop on Selected Areas in Cryptography, Springer-Verlag, LNCS Volume 1758/2000.

[12] Kohlbrenner, P., The Design and Analyses of a True Random Number Generator in a Field Programmable Gate Array, MS Thesis, ECE Department, George Mason University, Dec 2003, available from the author at paul@pk40.com.

[13] Lopez-Buedo, S., Riviere, P., Pernas, P., Boemo, E., Run-time Reconfiguration to Check Temperature in Custom Computers: An Application of Jbits technology, Field-Programmable Logic and Applications - Reconfigurable Computing Is Going Mainstream 12th International Conference, (FPL 2002), Springer-Verlag LNCS Volume 2438 / 2002

[14] Marsaglia, G., Diehard: A battery of tests for random number generators, 1985, http://stat.fsu.edu/~geo/diehard.html.

[15] Menezes, A., van Oorschot, P., Vanstone, S., Handbook of Applied Cryptography, 1997, CRC Press.

[16] Petrie, C., S., Connelly, J., A., Modeling and simulation of oscillator-based random number generators, IEEE International Symposium on Circuits and Systems, 1996. ISCAS '96 'Connecting the World', Volume: 4, May 1996 Pages: 324 -327 vol.4

[17] RAND Corporation, A Million Random Digits with 100,000 Normal Deviates., 1956, The Free Press.

[18] Rukhin et al., A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, NIST Special Publication 800-22 (revised May 15 2002).

[19] Schindler, W., Killmann, W., Evaluation Criteria for True (Physical) Random Number Generators used in Cryptographic Applications, Proceedings of the 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002), Springer-Verlag, LNCS 2523 (2002).

[20] Tsoi, K., H., Leung, K., H., Leong, P., H., W., Compact FPGA-Based True and Pseudo Random Number Generators, IEEE Symposium on FPGA-Based Custom Computing Machines (FCCM 2003).

[21] Xilinx, 2002, Superior Jitter Management With DLLs. Virtex Tech Topic, VTT013(v1.2).

[22] Xilinx, 2002, Annual Report and Form 10-k. http://media.corporateir.net/media.les/NSD/XLNX/annual2002/ar02/letter%.htm.