FPGA VENDOR AGNOSTIC TRUE RANDOM NUMBER GENERATOR

Dries Schellekens, Bart Preneel, and Ingrid Verbauwhede

Katholieke Universiteit Leuven
Department Electrical Engineering - ESAT/SCD-COSIC
Kasteelpark Arenberg 10
B-3001 Heverlee, Belgium
{dschelle,preneel,iverbauw}@esat.kuleuven.be

ABSTRACT

This paper describes a solution for the generation of true random numbers in a purely digital fashion; making it suitable for any FPGA type, because no FPGA vendor specific features (e.g., like phase-locked loop) or external analog components are required. Our solution is based on a framework for a provable secure true random number generator recently proposed by Sunar, Martin and Stinson. It uses a large amount of ring oscillators with identical ring lengths as a fast noise source – but with some deterministic bits – and eliminates the non-random samples by appropriate post-processing based on resilient functions. This results in a slower bit stream with high entropy. Our FPGA implementation achieves a random bit throughput of more than 2 Mbps, remains fairly compact (needing minimally 110 ring oscillators of 3 inverters) and is highly portable.

Key words: true random number generators, ring oscillators, jitter, resilient functions

1. INTRODUCTION

The security of many cryptographic systems depends upon the generation of nonrecurring and/or unpredictable quantities; these random numbers are, for instance, used for the generation of session keys, challenges in cryptographic protocols or padding of plain text messages [1]. While the nonrecurrence property can easily be fulfilled (e.g., by using a linear congruential generator or linear feedback shift register with large enough period), unpredictability is more difficult to assure.

Pseudo-random number generators (PRNGs) typically use some cryptographic mechanism to deterministically produce a sequence of random numbers from a randomly chosen seed; the random seed could, for example, be used as the key of a stream cipher. The generator however still has to be seeded by a random number produced by a hardware or software *true random number generator* (TRNG). Hardware-based generators exploit randomness which occurs in physical phenomena; nuclear decay and electronic noise are two examples of usable processes. A software-based TRNG on the other hand tries to derive a seed by mixing (usually with a compression function) different entropy sources (e.g., system clock, mouse movement, network statistics); this approach is sometimes called a *hybrid generator* to distinguish it from the physical – hardware – TRNG.

Historically, physical TRNG were only available in specialized cryptographic devices (smart cards, hardware security modules, cryptographic accelerators, ...), but they are appearing in more mainstream products like trusted platform modules, Intel motherboard chipsets [2] and VIA processors [3]. The implementation details of these solutions are mostly not published – because of their commercial value – but three different techniques seem widely used to generate a random bitstream: sampling jittered oscillators, chaotic circuits, and direct amplification of resistor or PN junction noise.

If the availability of high quality random numbers is required in an FPGA application, the latter two techniques are impossible, because they depend on analog components. For instance, good RNGs are becoming a necessity to protect hardware implementations of cryptographic operations against side channel analysis [4] – think of masked AES implementations [5] or blinded modular exponentiation [4].

Conventional solutions for digital TRNGs will typically extract randomness from the unpredictable jitter of oscillators. The design proposed in [6] samples the jitter of the clock signal synthesized by the analog phase-locked loop embedded in some FPGAs. Implementations of this concept have been demonstrated on Altera and Actel FPGAs, but this approach cannot be ported to others FPGAs. Xilinx FPGAs for instance use digital delay lines, instead of PLLs, to provide on-chip clock synthesis. The solution proposed in [7] relies on an external analog oscillator circuit, making it clearly unusable for tamper resistant and reliable FPGA applications, especially because disconnecting the external circuit stops the operation of the RNG. The preferable way to create a jittered oscillation is to rely on ring oscillators, an odd number of inverters in a ring configuration. The de-

1-4244-0 312-X/06/$20.00

sign of Kohlbrenner and Gaj [8] uses two ring oscillators to sample each other, but the placement of these oscillators into CLBs is tricky because the oscillation frequency needs to be closely matched; to overcome this placement sensitivity the authors suggest using more than two ring oscillators. In [9] Golić proposes two promising variations on classical ring oscillators using concepts from linear feedback shift register (LFSR) design: Fibonacci and Galois ring oscillators. These new asynchronous circuits with feedback provide a mixture of true randomness (jitter in ring oscillators) and pseudo-randomness (LFSR). The design was experimentally implemented on FPGA, but real details lack; the implementation is said to satisfy standard statistical tests, if very powerful post-processing (an irregularly clocked 64-bit LFSR) is applied.

The design parameters of most proposals (such as sampling frequency, amount of post-processing, and thus the resulting bitstream throughput) are mainly determined by trial and error until the random bitstream passes the statistical tests provided by the NIST [10] or DIEHARD [11] test suite. Mathematical modeling of the entropy source and justification of the post-processing is mostly absent. In this respect, the approach taken by Sunar, Martin and Stinson is completely different [12]. They start by modeling the entropy collection process and use this model to specify the design parameters, i.e. the number and length of ring oscillators and the requirements for the post-processor.

In the article we describe an FPGA realization of the proposal from Sunar et al. All the components of the TRNG design are given in Section 2. Section 3 and 4 describe the noise source and post-processing circuit used in our design. The implementation results are discussed in Section 5.

2. GENERIC ARCHITECTURE

Every physical random number generator will follow the generic architecture depicted in Fig. 1, where the definitions of [13, 14] are adopted.

Normally, the random noise source generates an analog signal $n(t)$ which is the result of some non-deterministic physical phenomenon. The analog noise signal is digitized (e.g., by a comparator), yielding the so-called digitized analog signals $s[i]$, briefly denoted as *das-random numbers*. The non-deterministic source and the digitizer together form the digitized noise source.

The digitized noise signal $s[i]$ is fed into a post-processor unit which then produces a sequence of m-bit random words $r[i]$, the so-called *internal random numbers*. The post-processing algorithm has two goals. First of all, the post-processor needs to adjust the probability distribution of the raw random bits $s[i]$, thus compensating statistical imperfections inside the entropy source or in the digitizer (e.g., offset of a voltage comparator). The probability distribution of the internal random words $r[i]$ is much closer to a uniform distribution than that of the das-random bitstream $s[i]$.

In the second place, the post-processing step is also used to increase the entropy per bit of the internal random words $r[i]$ by applying a compression function on the input stream $s[i]$, resulting in a lower speed output stream with increased randomness. This will be especially important if a noise source is used, that has a low entropy per bit. Compression also provides tolerance against environmental changes and tampering.

Two popular post-processors to reduce bias are the von Neumann corrector [2] and XOR corrector [6]. The XOR corrector just takes the exclusive-or of pairs of input bits; consequentially, the input stream is compressed with a factor 2. The von Neumann corrector also looks at pairs of input bits, but uses the first one if the bits are different and otherwise throws them away. The resultant stream will have a variable bit rate, but on average the compression factor will be 4. Then again, in other proposals more complicated post-processing algorithms than these simple correctors may be needed, for example a cryptographic hash function or an extractor function [15].

When the random number generator is integrated into a cryptographic module with security certification, an extra unit performing statistical tests is part of the design. Traditionally, NIST has specified some tests for the internal random numbers $r[i]$: a continuous test (subsequent random number should always be different) and statistical tests that need to be performed on startup of the device (monobit, poker, runs, long runs)[1]. However, by testing the internal random numbers, only the non-recurrence can be verified, and not the unpredictability; a PRNG will also generate a uniform probability distribution and pass the statistical tests, but it does not produce any entropy – like a TRNG. The guidelines provided by German IT security certification authority BSI list a number of tests that need to be performed on the das-random numbers $s[i]$, before the post-processing [13]. Evaluation of designs that use a low entropy noise source and that rely on high compression in the post-processor, is more difficult, because the standard BSI criteria will not be met.[2] To address this problem, the concept of a stateless random bit generator is introduced in [14]. For this class of generators, the verification of a minimum entropy limit could be performed directly on the post-processed random numbers.

3. NOISE SOURCE

The design proposed by Sunar et al. in [12] aims at generating random numbers in a purely digital way and conse-

[1] In the latest revision of the FIPS 140-2 document these statistical tests are no longer mentioned.

[2] To not discriminate against these designs, the BSI has foreseen "alternative criteria".

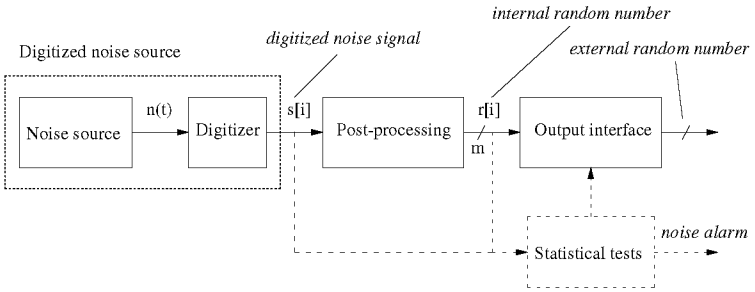

Fig. 1. Generic architecture for a random number generator

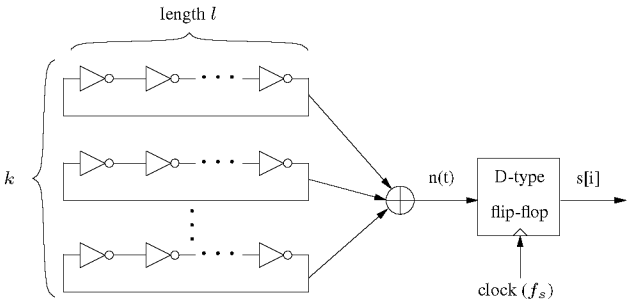

Fig. 2. Noise source based on ring oscillators

quently uses digital *ring oscillators* as its noise source (see Fig.2). If an odd number of inverters are connected in a ring configuration, the output of any of the inverters will oscillate from a logical zero to a logic one and back, owing to the instable nature of the circuit. If one inverter is replaced with a NAND gate, the ring oscillator can be disabled for instance to reduce power consumption. At any point in the ring a period square wave can be observed. Ideally, the period of this wave linearly depends on the number of inverters (i.e., the *ring length*) and the delay of a single inverter. In practice, there is some random variation on the moment the signal switches; this phenomenon is commonly called *jitter*. The goal of a digital TRNG is to harvest this entropy source by sampling the uncertain transition zones and not the deterministic part of the waveform. Generally, two approaches exist to extract randomness from jitter: sampling the output of a ring oscillator using the output signal of another oscillator (*coupled oscillators*) or combining the signals of a number of ring oscillators. The framework we build upon, uses the latter mechanism. The exclusive-or of k ring oscillators with length $l_1, \ldots, l_k$ is used as $n(t)$ and this signal is sampled at a regular clock frequency f_s using a D-type flip-flop creating a das-random bitstream $s[i]$.

By modeling the combined signal with a combinatorial model Sunar et al. find some important results. In contrast to what some have proposed, choosing the length of the ring oscillators $l_1, \ldots, l_k$ pairwise relatively prime does not yield better results than using identical ring lengths; thus from now on we assume that $l_i = l$ for all rings. The statistical model also allows to determine the required number of ring oscillators to fill the entire spectrum of the signal $n(t)$ with jittered transition zones. Justification of the expected entropy of the noise source is also provided.

Table 2. The jitter width as a percentage of the period depends on the ring length l (measured on a Xilinx Virtex XCV800) [16].

length l	25	41	57	67	83	101
jitter/period (%)	1.46	0.91	0.67	0.57	0.56	0.49

Nonetheless, they conclude that populating the whole spectrum with jitter events is undesirable, because too many ring oscillators would be necessary. Hence they suggest allowing a fraction of the signal $n(t)$ to be deterministic, but compensating for this in the post-processor. The *fill rate* f denotes the portion of spectrum that will be random. Table 1 gives the minimum number k of rings necessary (with a confidence of 99%) for certain fill rates.[3] This number also depends on the amount of jitter per period present in the ring oscillators. If the *jitter width* (i.e. the standard deviation for the jitter random variable) is larger, fewer ring oscillators are needed to obtain the same fill rate. Also note that an exponential effort must be invested to obtain a constant factor improvement in the fill rate. This fact confirms the observation that $f = 1$ is too expensive and that lower fill rates will yield a more effective design, provided that post-processing can efficiently get rid of the non-random bits.

The proportion of the jitter width to the entire period of oscillation is an important property for an implementation of this framework. The ratio depends on the technology used and hence needs to be determined experimentally or

[3]The results slightly differ from those in the paper of Sunar et al. We believe this is because they used another mathematical program producing the inaccurate calculations. All our calculations have been performed with MAGMA (`http://magma.maths.usyd.edu.au/calc/`) and have been verified with GP/PARI (`http://pari.math.u-bordeaux.fr/`).

Table 1. The number of rings k necessary so that with 99% confidence at least a fraction f of bits produced by the noise source are non-deterministic.

jitter/	fill rate f									
period	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95
4%	45	53	59	70	79	94	107	133	158	231
2%	83	96	110	127	146	169	198	236	292	393
1%	158	182	210	241	277	320	374	445	548	733

Table 3. The period of ring oscillator depends on the ring length l (measured on a Xilinx Virtex XC2VP30).

length l	1	3	5	7	9	13	19
period (ns)	2.7	3.0	5.0	6.6	7.5	10	15
length l	25	31	41	57	67	83	101
period (ns)	20	25	38	51	58	72	90

by simulations. The jitter of FPGA ring oscillators has been measured in [16, 17]. For a Xilinx Virtex FPGA (XCV800) the jitter width ranges from 30 to 45 ps.[4] The measurements were only done for long ring length (25 up to 101 inverters), because the waveform of shorter rings was not square – presumably because of the output capacitance when sending the oscillation to an output pin of the FPGA. For this Xilinx Virtex FPGA the mean period of the ring oscillation was also quantified and as expected it depends linearly on the ring length l and on the (technology specific) delay of a single inverter: $T \approx 1.77 \cdot l - 0.11$. The measurements of the jitter width proportional to the period are listed in Table 2. It seems that this percentage increases for shorter ring lengths, because the jitter width does not decrease as fast as the period when shortening the ring oscillator.

Our implementation of the design will be realized on a Xilinx XUPV2P development board with a Virtex II Pro FPGA (XC2VP30). We have measured the period of ring oscillators[5] with different lengths; a simple linear regression of the measurements in Table 3 gives the following relation: $T \approx 0.88 \cdot l - 0.23$. This difference with the other measurements is due to the fact that this FPGA is fabricated in more modern CMOS technology (0.13 μm/1.5V as opposed to 0.22 μm/2.5V); the logic gates have become twice as fast. As it is not so easy to quantify the jitter width, specially for short rings, we assume that the jitter width is 2% of the period. In practice, ring oscillators located close to each other tend to lock phase. This is why it always important to use more post-processing than theoretically required and/or to overcompensate by adding extra ring oscillators.

[4]These measurements seems to be consistent with a post on the comp.arch.fpga newsgroup by a Xilinx engineer stating that jitter of an FPGA ring oscillator will be about 40 to 60 ps peak to peak.

[5]A latch needs to be added in the ring in order that the oscillator can to be synthesized by the Xilinx tools. This latch just adds some extra delay and thus increase the period a bit.

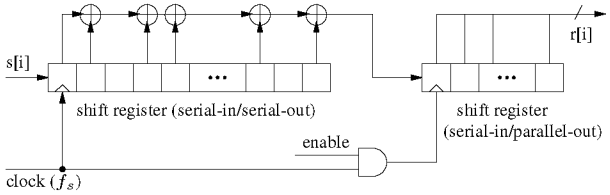

Fig. 3. Efficient implementation of post-processing algorithm based on cyclic codes

4. POST PROCESSING

By allowing the noise signal $n(t)$ to contain deterministic bits, the design becomes practical. For example, a fill rate of 0.95 necessitates at least 393 ring oscillators, while only 110 rings are required if a fill rate of 0.60 is allowed (see Table 1, still assuming 2% jitter). Sunar et al. recommend the usage of a *resilient function* to eliminate the non-random components in the post-processing unit. They show a simple technique to obtain suitable resilient functions from error-correcting codes. The internal m-bit random words $r[i]$ are calculated using n ($> m$) das-random bits $s[i]$ using

$$(\; r[i] \quad \ldots \quad r[i+m-1] \;) = (\; s[i] \quad \ldots \quad s[i+n-1] \;) \cdot \mathbf{G^T}$$

where $\mathbf{G}$ is a generator matrix for an [n,m,d] *linear code*. For *cyclic codes*, a special class of linear codes, the generator matrix will have the form

$$\begin{pmatrix} g_0 & 0 & \ldots & 0 \\ g_1 & g_0 & & 0 \\ \vdots & \vdots & \ddots & \\ g_{n-m-1} & g_{n-m-2} & \ldots & g_0 \\ 0 & g_{n-m-1} & \ldots & g_1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & & g_{n-m-1} \end{pmatrix}^T$$

This resilient function can be efficiently implemented using the circuit depicted in Fig. 3. The first $n - m$ cycles the input bits $s[i]$ are shifted into a register of $n - m$ bits. The next m cycles the output bits $r[i]$ are produced by XOR-ing a number of the bits of the shift register. The position of the taps is determined by the generator matrix

of the cyclic code. The resilient function has a compression factor of $\frac{m}{n}$ and can correct $d-1$ "errors" in m input bits. The designer has to choose d such that $\frac{d-1}{m} > f$; that is, the resilient function can correct more non-random bits than are produced by the noise source. A good tool to assist in the selection of suitable linear codes is available on [18]. A trade-off between the hardware cost of the noise source (number of inverter gates needed) and the complexity of the post-processing (mainly flip-flops and XOR gates) can be exploited.

5. RESULTS

Sunar et al. end their paper with a brief description of a sample design, by picking some realistic values for the parameters of the framework. They propose to use rings of 13 inverters and assume based on the experimental results of [17] that these ring oscillators have a period of 25 ns (i.e., 40 MHz) and that the jitter has a standard deviation of 50 ps, so $\sigma = 0.02T$. The framework suggests that these parameters yield at least 0.97 bits of entropy per sampled bit. In order to achieve a fill rate f of at least 0.60 their calculation shows that 114 rings[6] are needed. The resilient function they suggest is based on a cyclic [256,16,113]-code [19]. This post-processing has a compression factor of $\frac{256}{16} = 16$ and can filter out 112 corrupted bits, while only $(1-0.6)\cdot 256 = 103$ out of 256 bits are expected be deterministic. When sampled at 40 MHz, a random bit rate of $\frac{40}{16} = 2.5$ Mbps is produced.

In our research, we have implemented this proposal on a Xilinx FPGA and we have verified that the theoretical background of the framework indeed holds in practice. In order to minimize the required hardware resources, we made a slight change to the original proposal. By using ring oscillators with a shorter length, namely $l = 3$, we significantly reduced the number of inverters needed in the noise source. However, it is a bit unclear whether this has a big influence on the available jitter; the measurements from [16, 17] seem to suggest that shorter rings result in more jitter per period and thus potentially more randomness. The downside of our adjustment is that fact that the ring oscillators will not create a good square waveform, so maybe the theoretical model no longer holds. From our own measurements (see Table 3) we know that ring length 3 gives a period of 3 ns (i.e., 333 MHz) and hence potentially a higher sampling frequency can be used. The fact that fewer inverters (around $13/3 \approx 4$ times) could lead to much higher random bitstream ($333/40 \approx 8$ times) appears rather suspicious. Because of this, we decided to keep the sampling frequency at 40 MHz. We also keep the choice of the [256,16,113]-code, because cyclic codes can implemented in a very efficient matter.

The original proposal (with $l = 13$) uses 1664 slices of the FPGA, while the minimal version only occupies 565 slices (see Table 4). In case we overestimated the jitter on the more modern Xilinx FPGA, it would be safer to use more rings; for example, 210 rings increases the slice count to 973 but allows the standard deviation of the jitter to be $\sigma = 0.01T$ (instead of 2% of the period).

[6]More precise calculations suggest $k = 110$ is sufficient (see Table 1).

Table 4. The resource usages of the design (on a Xilinx Virtex XC2VP30) depends on the number of rings k and the length l used as noise source; post-processing is done with a [256,16,113]-code and occupies 115 slices.

noise source	k	l	# slices
reference	110	13	1664 (12 %)
minimal	110	3	565 (4 %)
robust	210	3	973 (7 %)

We have checked our design with standard tests (NIST and DIEHARD) and confirmed that the statistical properties of the produced random numbers are fine. This however does not validate that the design indeed produces 0.97 bits entropy per output bit.[7]

It is also nice to point out that our minimized version satisfies the requirements of a stateless random bit generator recently defined in [14]: the noise source can be considered stateless as the sampling frequency (40 MHz) is sufficiently small compared to the noise source bandwidth (more than 300 MHz) and the post-processor is memoryless because sequentially produced internal random numbers $r[i]$ only depend on the das-random bits $s[i]$ and not on each other. Since our random number generator is stateless, and a detailed statistical model of the noise source is provided, we believe it should be possible to certify the design, given that appropriate (online) statistical tests on the post-processed random number are applied.

6. CONCLUSIONS AND FUTURE WORK

We have verified that the framework for the provable secure true random number generator proposed by Sunar et al. is efficiently implementable on any FPGA type. In order to reduce the hardware requirements we propose using shorter ring oscillators, but also suggest using more rings than theoretically necessary. Furthermore we have verified that an real implementation of design passes all common statistical tests.

Interesting future work would be to better measure the jitter of short ring oscillators and test the robustness of the FPGA implementation. What happens when the operating

[7]In the latest version of their paper Sunar et al. state that the entropy before the resilient function is $H \approx pfN$, so for their sample design 0.59 bits. To verify this experimentally, we tried compressing the das-random bitstreams with bzip2 but this was not possible. Other statistical tests, like Mauer's universal statistical test, however fail, showing that the das-random numbers contain statisticals defects that get corrected by the post-processor.

conditions of FPGA (say temperature) are changed? Can the frequency and jitter of the ring oscillators be influence by modulating the FPGA power supply with a frequency close to the oscillator frequency?

7. ACKNOWLEDGEMENTS

The authors would like to thank Christophe De Cannière, Berk Sunar and the anonymous reviewers for helpful comments and discussions. This work was supported in part by the FWO (G.0450.04), the IBBT and the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government.

8. REFERENCES

[1] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996. [Online]. Available: http://www.cacr.math.uwaterloo.ca/hac/

[2] B. Jun and P. Kocher, "The Intel Random Number Generator," Cryptography Research, Inc. White Paper prepared for Intel Corporation, 1999. [Online]. Available: http://www.cryptography.com/resources/whitepapers/IntelRNG.pdf

[3] "Evaluation of VIA C3 Nehemiah Random Number Generator," Cryptography Research, Inc. White Paper prepared for VIA Technologies, 2003. [Online]. Available: http://www.cryptography.com/resources/whitepapers/VIA_rng.pdf

[4] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, ser. Lecture Notes in Computer Science, N. Koblitz, Ed., vol. 1109. Springer, pp. 104–113.

[5] T. S. Messerges, "Securing the AES Finalists Against Power Analysis Attacks," in *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*, ser. Lecture Notes in Computer Science, B. Schneier, Ed., vol. 1978. Springer, 2001, pp. 150–164.

[6] V. Fischer and M. Drutarovský, "True Random Number Generator Embedded in Reconfigurable Hardware," in *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, ser. Lecture Notes in Computer Science, B. S. K. Jr., Çetin Kaya Koç, and C. Paar, Eds., vol. 2523. Springer, 2003, pp. 415–430.

[7] K. H. Tsoi, K. H. Leung, and P. H. W. Leong, "Compact FPGA-based True and Pseudo Random Number Generators," in *11th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2003), 8-11 April 2003, Napa, CA, Proceedings*. IEEE Computer Society, 2003, pp. 51–61.

[8] P. Kohlbrenner and K. Gaj, "An Embedded True Random Number Generator for FPGAs," in *Proceedings of the ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays, FPGA 2004, Monterey, California, USA, February 22-24, 2004*, R. Tessier and H. Schmit, Eds. ACM, 2004, pp. 71–78.

[9] J. D. Golić, "New Paradigms for Digital Generation and Post-Processing of Random Data," Cryptology ePrint Archive, Report 2004/254, 2004. [Online]. Available: http://eprint.iacr.org/2004/254.ps

[10] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo, "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," NIST Special Publication 800-22 (with revisions dated May 15, 2001). [Online]. Available: http://csrc.nist.gov/rng/SP800-22b.pdf

[11] G. Marsaglia, "DIEHARD: Battery of Tests of Randomness," 1996. [Online]. Available: http://stat.fsu.edu/pub/diehard/

[12] B. Sunar, W. J. Martin, and D. Stinson, "A Provably Secure True Random Number Generator with Built-in Tolerance to Active Attacks," 2005. [Online]. Available: http://www.cacr.math.uwaterloo.ca/~dstinson/papers/rng-IEEE.pdf

[13] W. Schindler and W. Killmann, "Evaluation Criteria for True (Physical) Random Number Generators Used in Cryptographic Applications," in *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, ser. Lecture Notes in Computer Science, B. S. K. Jr., Çetin Kaya Koç, and C. Paar, Eds., vol. 2523. Springer, 2003, pp. 431–449.

[14] M. Bucci and R. Luzzi, "Design of Testable Random Bit Generators," in *Cryptographic Hardware and Embedded Systems – CHES 2005, 7th International Workshop, Edinburgh, Scotland, August 29 - September 1, 2005, Proceedings*, ser. Lecture Notes in Computer Science, J. R. Rao and B. Sunar, Eds., vol. 3659. Springer, 2005, pp. 147–156.

[15] B. Barak, R. Shaltiel, and E. Tromer, "True Random Number Generators Secure in a Changing Environment," in *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, ser. Lecture Notes in Computer Science, C. D. Walter, Çetin Kaya Koç, and C. Paar, Eds., vol. 2779. Springer, 2003, pp. 166–180.

[16] B. J. Abcunas, S. P. Coughlin, G. T. Pedro, and D. C. Reisberg, "Evaluation of Random Number Generators on FPGAs," MQP Report, 2004. [Online]. Available: http://www.crypto.wpi.edu/Publications/Documents/MQP_jitter_1.pdf

[17] W. R. Coppock and C. R. Philbrook, "A Mathematical and Physical Analysis of Circuit Jitter with Application to Cryptographic Random Bit Generation," MQP Report, 2005. [Online]. Available: http://www.crypto.wpi.edu/Publications/Documents/MQP_jitter_2.pdf

[18] A. E. Brouwer, "Server for bounds on the minimum distance of q-ary linear codes." [Online]. Available: http://www.win.tue.nl/~aeb/voorlincod.html

[19] S. Duplichan, 2000. [Online]. Available: ftp://ftp.win.tue.nl/pub/home/aeb/math/codes/genmats/2/2.257.17.113