A meta-level true random number generator

Bernhard Fechner*

Department of Mathematics and Computer Science,
Parallel Computing and VLSI Group,
University of Hagen,
Universitätsstraße 1, 58084 Hagen, Germany
E-mail: Bernhard.Fechner@fernuni-hagen.de
*Corresponding author

Andre Osterloh

BTC AG,
Escherweg 5, 26121 Oldenburg, Germany
E-mail: andre.osterloh@btc-ag.com

Abstract: True random number generators (TRNGs) are extensively used in cryptography, simulations and statistics. In this work, we introduce, extend and analyse the concept of the randomised bit cell (RBC), introducing a second meta-level of randomisation, being able to simultaneously produce random numbers and detect active non-intrusive attacks. The concept is extended by using a corrector. Meta-stability is one way to generate true random numbers. By using electromagnetic radiation, a flip-flop (ff) in a meta-stable state can be manipulated to a known state. We clarify and comprehend open issues of meta-stable ffs such as power consumption and electromagnetic field strength. The experimental comparison though a software simulation with a standard TRNG yields a 17.69 times better distribution of zeros and ones while the TRNGs are under attack at the expense of a delay which is proportional to the quality of the random source.

Keywords: meta-level; true random number generator; TRNG; corrector.

Reference to this paper should be made as follows: Fechner, B. and Osterloh, A. (2010) 'A meta-level true random number generator', *Int. J. Critical Computer-Based Systems*, Vol. 1, Nos. 1/2/3, pp.267–279.

Biographical notes: Bernhard Fechner has an accident insurance education. He obtained his Master's and PhD in Computer Science from the University of Hagen. He worked several years as a Consultant and was involved in the conception and realisation of the first telematic and virtual computer architecture lab for the University of Hagen, Germany. He is an Assistant Professor at the Chair of Parallel Computing and VLSI (Prof. Dr. J. Keller). His research interests include (fault-tolerant) micro-architectures, their realisation and computer graphics.

Andre Osterloh received his Master's in Computer Science from the Carl von Ossietzky University Oldenburg and his PhD in Computer Science from the Technical University of Ilmenau. He worked several years as a Software Engineer in two consulting firms. Furthermore, he worked for more than ten years as a Lecturer and Scientist for the Universities of Oldenburg, Ilmenau, Hagen and Darmstadt. He is a Software-Engineer at BTC AG Renewable Energies and a member of the eTelligence team.

1 Introduction

Random numbers are needed and used in our everyday lives. The generation of random numbers is an essential component to generate passwords, session keys for secure socket layer (SSL). They are also used in simulations such as nuclear physics, weather forecasts, traffic simulations or Monte-Carlo approximations. The quality of a random number generator (RNG) substantially determines the security of the underlying communication. A manipulation leads to predictable random numbers and causes unsafe communication channels. We have to clarify what a *good* random number is. In our case, a good random number is a binary number where zeros and ones are discretely and uniformly distributed. A discrete random variable X with a finite number of characteristics x = {0,1} is uniformly distributed, if the probability of each characteristic $x = x_i$ ($i = 1,2$) is equal:

$$P(X = x) = \begin{cases} \frac{1}{2} \text{ for } x = x_i, \\ 0 \text{ else.} \end{cases}$$

P is the probability mass function of X, the entropy is

$$H(X) = E(I(X)) = p(x_2)I(x_2) + p(x_1)I(x_1) = -\left(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}\right)$$

Since $H(X) = 1$ the entropy is maximal – a fair distribution.

A future sequence of bits must not be predicted. Thus, from the knowledge on time t_0, we cannot forecast any characteristic on time $t_1 > t_0$. Let $S(t_0)$ be the sequence of characteristics on time t_0. If we cannot deduct $S(t_1)$ from $S(t_0)$ with any function f, we have a random result. Thus, if: $S(t_1) = f(S(t_0))$ we have no randomisation.

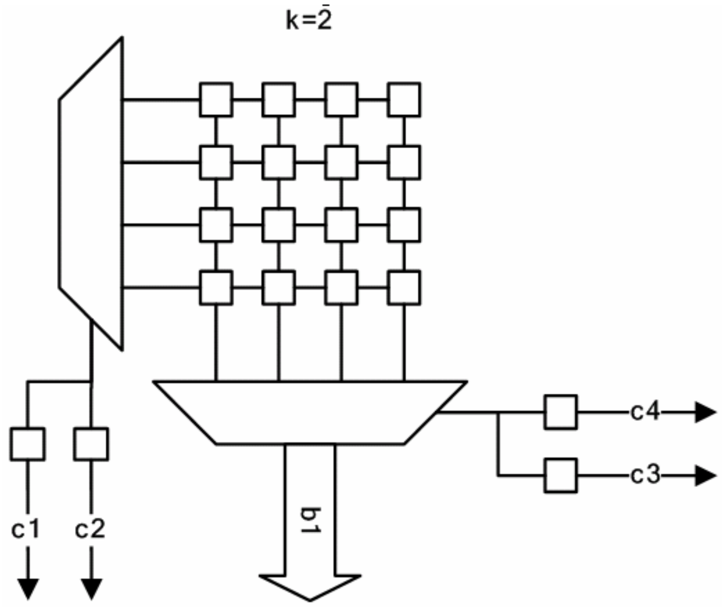
This article is organised as follows: in Section 2, we review related work. In Section 3, we introduce the concept of the randomised bit cell (RBC). In Section 4, we discuss important measures like the mean time between failures (MTBF), open issues with electromagnetic radiation and power consumption of RBCs based on meta-stable flip-flops (ffs). We extend the RBC to that of the smoothing bit cell (SBC) in Section 5 by using a corrector and theoretically examine and compare the SBC with other concepts. Section 6 presents the results of various software fault-injection experiments we conducted to show the different distributions of characteristics. Section 7 concludes the article.

2 Related work

In Chapter 5 of Menezes et al. (1997), it is shown that it is impossible to give a mathematical proof whether a RNG creates real random numbers or not. The only possibility to test a RNG is to use statistical tests to check whether the generator has one or more of the desired properties. If the generator fails in a test the result could be discarded or – in the case of an online test – a warning could be given. But even if all tests are successful, this is only a probabilistic proof that the generated random sequence has some desired properties of a real random sequence. A statistical test is a method that tests whether a statistical hypothesis holds or not. A statistical hypothesis is an assumption about the distribution of a random source. In Neuenschwander (2004, p.89), a list of constraints is given which could be tested. In the case of RNGs, such a hypothesis is the assumption that the produced random sequence is uniformly distributed and independent. A simple example for a statistical test of a RNG is to count the number of zeros in the generated random sequence. Common statistical tests are the frequency, serial, poker, autocorrelation, run and long run test which are described in Knuth (1997) and Beker and Piper (1982).

This work deals with TRNG and is an extension and abstraction to the paper presented by Fechner (2008). In contrary to pseudo RNGs (PRNGs), e.g., based on linear congruency (Knuth, 1997) or the Mersenne Twister (Matsumoto and Kurita, 1992), TRNGs are able to produce a non-periodic sequence of bits. Numerous TRNGs have been developed, some of them with industrial background, e.g., the RNG based on thermal noise within the Intel i810 chipset (Hoffman, 1998). Other TRNGs are based on radioactive decay (http://www.fourmilab.ch/hotbits), atmospheric noise (http://www.random.org/randomness), ring oscillators (Sunar et al., 2007), the photoelectric effect or other quantum phenomena (http://random.irb.hr/). In Walker and Foo (2001), an analog-digital converter is used to produce random numbers from the output of a ff being in a meta-stable state although it is unclear on which voltage levels the ff will toggle. We are aware of the fact, that a TRNG can be subordinated to physical influences. Therefore, one should carry out tests like DIEHARD (Marsaglia, 1996) or NIST tests (Rukhin et al., 2000) to evaluate the RNG. These tests require a huge amount of random numbers to gain statistical significance and thus a long time to determine the *actual quality* of the RNG. For fast online tests as in this work, others methods have to be regarded.

Figure 1 Randomised bit cell (RBC)

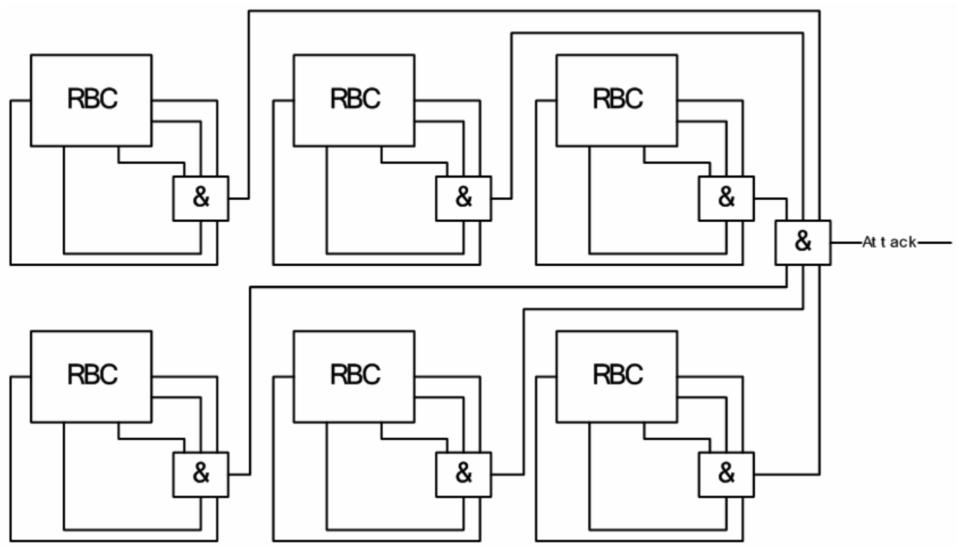


3 Randomised bit cells

Figure 1 shows the basic RBC. All square elements are random state generators (RSGs), producing random sequences of zeroes and ones. In the next section, we discuss an implementation of RSGs, ffs which have a random state due to a previous meta-stable period. Naturally, in this case, clock signals have to be introduced. The main idea is clear: the RSGs controlling the decoders are generating random numbers, selecting random elements out of the bit cell matrix which are also randomised. Thus, the TRNG is completely self-sustaining. In the example in Figure 1 we have five random outputs b1 and c1–c4. The main output is a single bit b1. The bits c1–c4, b1 are used for attack detection. If we assume the number of RSGs as 2^{2k} arranged in a $(2^k, 2^k)$-matrix, we have 2k bits for the selection of a single bit cell. The RBC with *k* selection bits is called RBC_k.

The advantage of the scheme in Figure 1 is that, the second level (the RSGs at the inputs of the decoder) introduces a meta-level of randomisation. If an attacker is able to disturb the contents of the decoder RSGs, there is still a randomisation, because a dedicated bit cell will be selected, which is again randomised. If the attacker tries to influence the function of a single bit cell, the randomisation to select the bit cell will decrease the probability for a successful attack. This probability computes to $\frac{1}{2^{2k}}$. One could think of XORing the outputs of the RSGs to a single value. Here, the circuit depth is growing with *k* due to faniin problems. The logical depth of the RBC is always the same. A single and most common case of a non-invasive attack is left: the influence of all RSGs within the RBC. For example, energy in the form of electromagnetic radiation will toggle the RSGs to a fixed and known state. We can detect these attacks by comparing the outputs c1–c4 and b1. Figure 2 shows a way to detect manipulations by comparing RBC signals. Every output within the array of RBCs is compared locally.

Figure 2 An n-bit RNG with attack detection



Note, that the number of false positives can be reduced if a history of c1–c4, b1 values is introduced. If we have a global matching, an attack is signalled. Note, that we are also able to detect local manipulations. A mechanism to detect permanent faults through manipulation of the attack signal is use dual-rail-logic (Fechner and Osterloh, 2008). In this case, every stuck-at fault is detected by comparing both signals by using an external circuit within the perfection core.

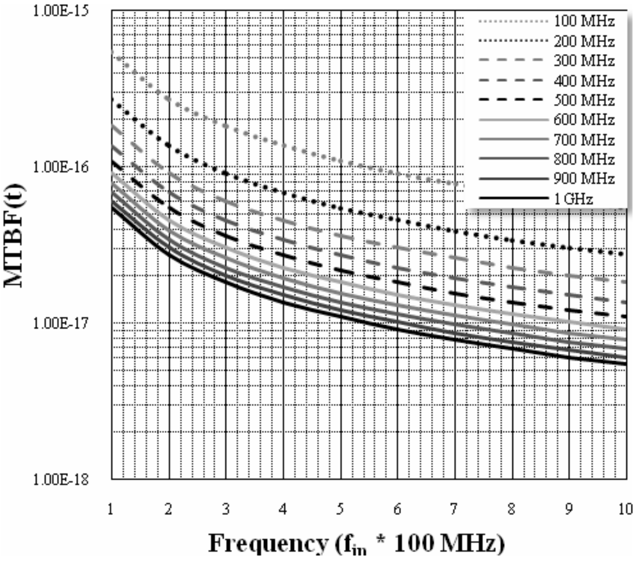
4 Open issues with meta-stability

One way to generate random numbers is the violation of setup and hold times within a ff. This leads to an unpredictable behaviour called *meta-stable state*. For the generation of n-bit random numbers in this work, we use the final state of a ff being metastable once. The probability that the ff will be meta-stable is an exponentially decreasing function (Philips, 1989).

Three things must be known about meta-stability (Philips, 1989):

- What is the likelihood T_0 that the ff will enter a meta-stable state?
- How long will it be expected to remain in that state (τ)?
- What is the propagation delay of the ff?

Figure 3 MTBF of meta-stable ffs



The MTBF of a ff in a meta-stable state results from empirical observations (Walker and Foo, 2001) to

$$MTBF(t) = \frac{e^{\frac{t}{\tau}}}{\left(T_0 \cdot f_{clk} \cdot f_{in}\right)},$$

where t is the time, a digital output can remain in a meta-stable state without causing a synchroniser failure, f_{clk} the system clock frequency and f_{in} the frequency of the incoming data. In addition – among many others – the technology and ambient temperature determine how long it takes the ff to settle down. Remember, that in contrary to e.g., Chaney (1983), we want to have meta-stable states to produce random numbers. Thus, the aim is a short MTBF. Figure 3 depicts the MTBF (f_{clk} from top to bottom 100 MHz to 1 GHz in steps of 100 MHz), whereas the frequencies f_{in} were varied from 100 MHz up to 1 GHz in steps of 100 MHz.

Before we continue, we have to discuss an open issue with meta-stability. The ffs being in a meta-stable state cause electromagnetic radiation which could influence other ffs. The wavelength $\lambda = \frac{c}{f}$ in metres of an electromagnetic wave is dependent from the phase speed (here: the speed of light in vacuum c =299792458 m/s) and the frequency f. If the wavelength is larger than the chip or the distance of ffs assumed to interact, there will be no issue. Today, feature sizes of 45 nm and below are implemented. As an actual example, we take a six-transistor SRAM in 26 nm process technology that consumes as less as 0.122 µm^2 area (http://electronicsweekly.com). Assuming that the signal will travel exactly one side of the area of the ff, the maximal frequency such a circuit can toggle at is $\sqrt{0.122 \cdot 10^{-6}\text{m}} = \frac{299792458 \text{ m/s}}{f}$. Solving the equation results to $f \approx$ 36.84 THz. In the real world, the frequency will be much lower, since it is influenced from temperature, technology and other effects. The probability that ffs can influence each other depends on the energy of the signal depending on distance and frequency of the ffs. If the distance is large enough, ffs cannot interact. The magnitude of the electric field $E_{far,near}$ decreases with the distance, according to the solution of Maxwell's equations for an ideal magnetic dipole, modified to show the field magnitude (Johnson and Graham, 1993) in the far field $\left|E_{far}\right| = \frac{\eta_0 IA\pi}{r\lambda^2}\sqrt{1 + \frac{\lambda^2}{r^2(2\pi)^2}}\sin\theta$ V/m, where $\eta_0 = 120\pi \approx 377\Omega$ is the intrinsic impedance of the free space, l the current in amperes, $A = 0.122 \cdot 10^{-6}\text{m}^2$ the area, r the distance from the centre, $\theta = \frac{\pi}{2}$ the angle from the z-axis and λ the wavelength in meters. Note, that we have chosen $\sin\theta = \sin\left(\frac{\pi}{2}\right) = 1$ to produce maximal field strength. In the near field this simplifies (Hall, 2000) to $\left|E_{near}\right| = 4.8\frac{IA\lambda}{r^3}$ V/m. Figure 4 and Figure 5 show the field strength of the mentioned SRAM in V/m for the near and far field, respectively. The wavelength λ in Hz was set to {10^9, 10^{10}, 10^{11}, 10^{12}} (from top to bottom) to cover the terahertz range, r the distance was varied between A and $2 \cdot A$.

The energy is reduced with the distance and decreasing wavelength. By using current technology, the electrical field of meta-stable ffs is not strong enough to induce energy to toggle faults in adjacent ffs.

Figure 4 The field strength of a six-transistor SRAM in V/m (near field)

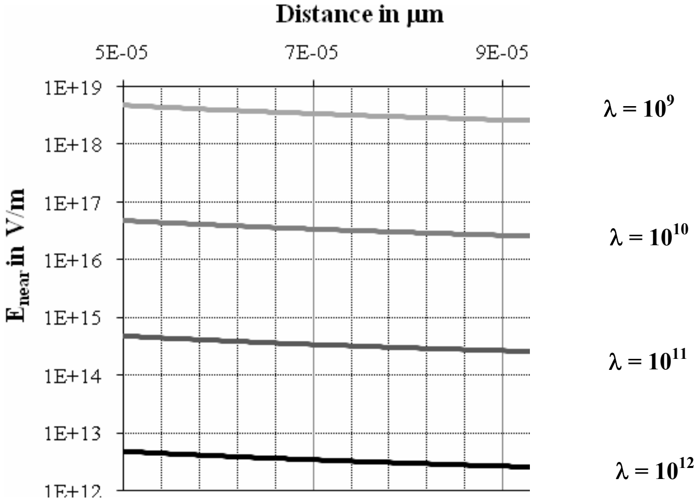
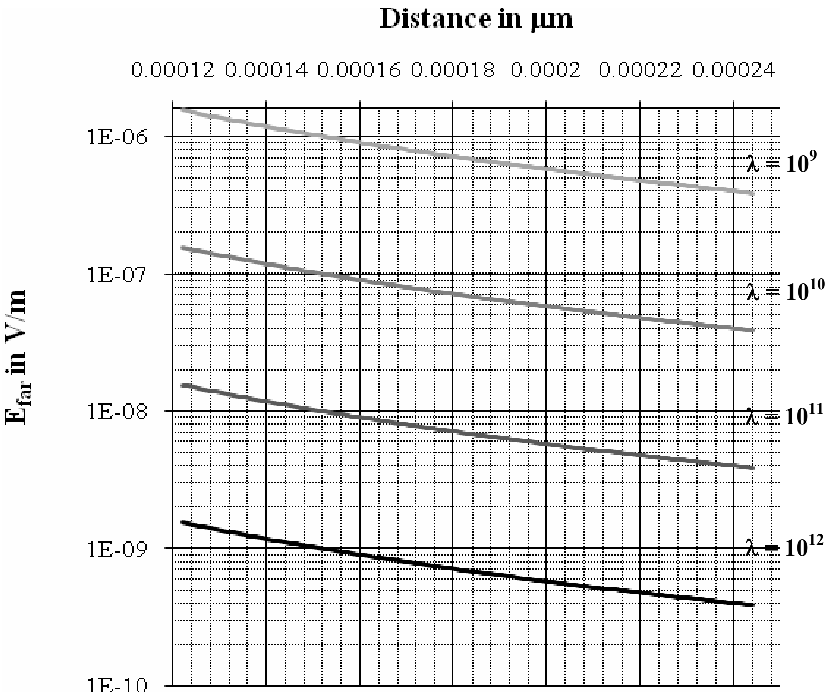


Figure 5 The field strength of a six-transistor SRAM in V/m (far field)



5 Improving the quality of randomness with the SBC

In this section, we assume that our random source is not perfect, i.e., the probability that the source produces a zero differs from the probability that it produces a one. Such effects can occur due to the production process or by external circumstances (e.g., a non-intrusive attack). In our fault model, we assume that the attacker is able to change the state of one or more ffs systematically. Therefore, we propose the online injection of a burst of one to n stuck-at-0 or stuck-at-1 faults. We assume stuck-at-0 and stuck-at-1 faults to occur with equal probabilities. The observation interval is the time to generate a single random number of length n. We introduce the SBC to reduce such effects and discuss its assets and drawbacks. The SBC is a simple extension of a RBC by using a corrector. In our case, we use a von-Neumann corrector (Jun and Kocher, 1999). To avoid misunderstandings, we call ones and zeros produced by the random source zero and one-bit (random bits) and the zeros and ones produced by the proposed method as zeros and ones (random numbers). Let $0 < p0$, $p1 < 1$, $p0 + p1 = 1$ the probability that the random source produces a zero or one, respectively. We get the probabilities listed in Table 1 for sequences of length two.

Table 1 Sequences and probabilities

Sequence	*Probability*
00	$p0^2$
01	$p0p1$
10	$p1p0$
11	$p1^2$

Note that the probability for ‘01’ is the same as for ‘10’. We exclude the sequences ‘00’ and ‘11’. The drawback is that, in case of long sequences of zero or one-bit no random numbers are generated. The expected number of bits, we have to wait to create a zero or a one is

$$\sum_{n=0}^{\infty} 2nq(1-q)^{n-1} = {}^{2}\!/_{q},$$

where $q = 2p0p1$. If the source produces a zero-bit with probability 2/5, i.e., $q = 6/25$, the expected number of bits we have to wait for is $25/6 \approx 4.167$. If we assume a very bad random source producing a zero-bit with probability 10^{-k}, the expected number is approximately 10^k. An advantage is that, we are able to detect stuck-at-0 and stuck-at-1 faults or alleviate manipulations of the random source through a timer, measuring the output frequency.

Now we discuss whether this approach can be extended to a sequence of length $n > 2$. The main idea in case of $n = 2$ was that we have to identify sequences with equal probabilities. The only possibility to do this is to take sequences where the number of zeros and ones is equal. Hence, we have to restrict our attention to even n. The probability that a sequence of length n exactly contains $n/2$ zeros is

$$\binom{n}{n/2}(p0)^{\frac{n}{2}}(p1)^{\frac{n}{2}} = (2p0p1)^{\frac{n}{2}}.$$

Thus, we get $q = (2p0p1)^{\frac{n}{2}}$ in this case. Thus, $n = 2$ is the best choice.

The number of bits necessary to produce a one or a zero could be seen as a measure for the time needed to produce a random number. To reduce this time, we can produce random numbers of k random sources in a synchronised and parallel fashion. The question is how long, i.e., how many steps we have to wait until we get a one or a zero. First, we have to describe how zeros and ones are produced and what will be a *step* in our model. In one step, each of the k sources produces one bit. If at least one of k sources separately produces '01' or '10' after two steps, we have an output of zero or one. In case that all k sources produce '00' or '11', we have no output. The probability that there is no output after two steps is $\prod_{i=1}^{k}(1-2p0_i p1_i)$, where $p0_i$ ($p1_i$) is the probability that the random source i produces a zero (one). The expected number of steps we have to wait is

$$\frac{2}{1-\prod_{i=1}^{k}(1-2p0_i p1_i)} = \frac{2}{1-\prod_{i=1}^{k} q_i},$$

where $q_i = 1 - 2p0_i p1_i$. If we assume two random sources with $q_1 = q_2 = 6/25$, the expected number of steps we have to wait is approximately 2.122.

Now we assume that we have one faulty ff and we want to determine the probability that the produced random number is affected by this fault (i.e., a worst-case fault). Obviously the result is affected with probability one. Table 2 shows the probabilities for this case.

Table 2 Worst-case results

Number of ffs in matrix	*Total number of ffs*	*Probability*	*Name*
4	6 = 4 + 2	1/6	RBC_1
16	20 = 16 + 4	1/20	RBC_2
2^{2k}	$2^{2k} + 2k$	$1/(2^{2k} + 2k)$	RBC_k

For the case where we have k ffs in parallel, the probability depends on the way we implement the choice when two or more random numbers are produced by the k random sources. Under the assumption that one from k random numbers is chosen with probability $1/k$ and that $p0 = p0_i$ and $p1 = p1_i$ for all i, we get a probability of

$$\sum_{i=0}^{k-1}\binom{k-1}{i}\frac{1}{i+1}p^i(1-p)^{k-1-i},$$

where $p = 2p0p1$.

6 Experimental results

For a qualitative comparison we conducted software fault-injection experiments. Faults were injected according to the fault-model in Section 5. For all methods, we injected burst faults at random positions, since this is likely to match a physical injection modelling an attack. Therefore, a burst of length n is able to influence n or less bits.

Numerous combinations of RSGs, RBCs and SBCs are possible. For a better overview, we introduce the nomenclature shown in Table 3.

Table 3 Nomenclature for fault-injection experiments

	Decoder
D0	Non-smoothing
D1	Smoothing
--	No decoder
	Contents
N	Normal meta-stable ffs
R	RBCs
S	Smoothed, RBCs

The description is read from top to bottom. For example: --*N* means an array of normal RSGs, naturally without decoder. Table 4 shows the abbreviations and the meaning of the structures considered for the fault-injection. Note, that the matrix with normal/smoothing decoder and normal RSGs is not reasonable, since the decoder inputs will be unclear.

Table 4 Simulated structures

Abbrev.	*Meaning*
__N	Array with normal RSGs, reference
__S	Array with smoothed, RBCs
D0R	Array with RBCs and normal decoder
D0S	Matrix with normal decoder and SBCs
D1R	Matrix with smoothing decoder and RBCs
D1S	Matrix with smoothing decoder and SBCs

As mentioned, the SBC uses an additional bit holding the previous random result. If the current result differs from the last content, we output a random bit according to a fixed coding like (10)→0 or (01)→1, whereas the values on the right occur later. Other results are retained. Figure 6 shows the development of distributions for all structures from Table 4, 1–39 stuck-at-0 bursts, a 16 × 16 (k = 8) matrix, averaged over 1000 fault-injection runs. We do not show the results for other values of k, because the results were very similar. The array holding the structures had the length of 40 bits, since the typical length for session keys ranges from 40–128 bit or 160 bit (Baugher et al., 2004). For larger session keys, the results can be easily interpolated since they are very regular. The number of zeros and ones was counted for each method. The one-bit frequencies were divided by the zero-bit frequencies, resulting in the data points in Figure 6.

We compute the standard deviation of all 1/0 distributions for all matrix sizes {2, 4, 8, 16, 32}. The D0R gives a 17.69 times better distribution than the reference. Figure 7 shows the improvement of the standard deviation in comparison with the __N reference for different matrix sizes N. Note, that N is the input size of the decoder. If N = 2, the actual matrix size is $2^N = 4$.

Figure 6 Development of distributions over 1–39 burst faults

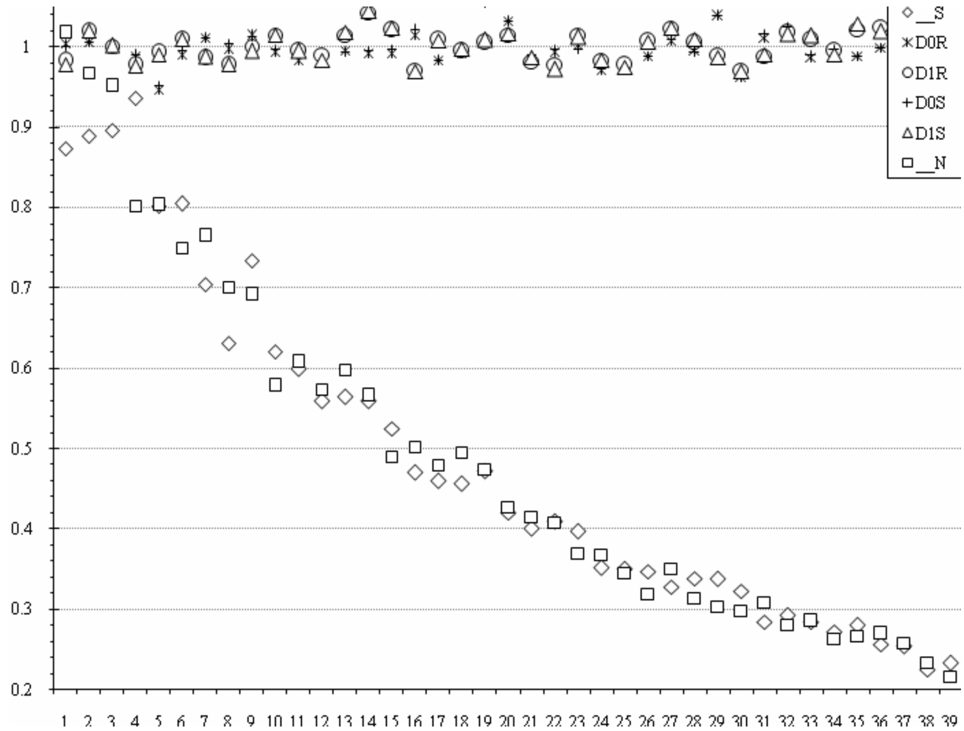
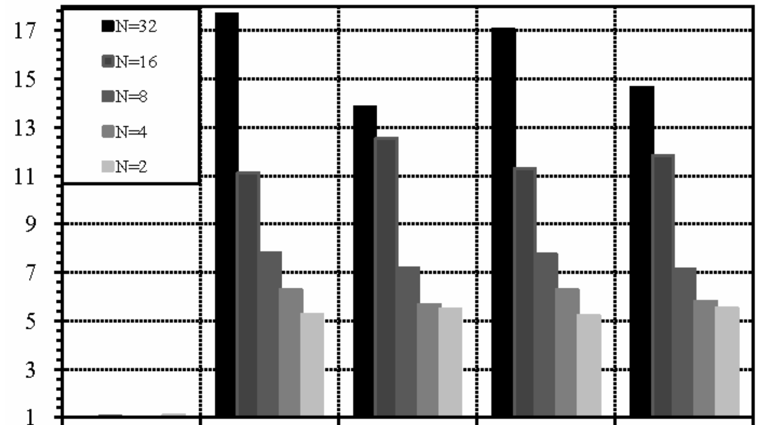


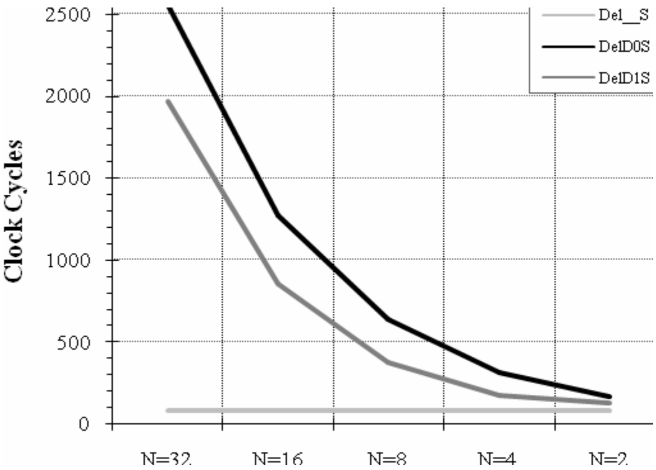
Figure 7 Standard deviations for different matrix sizes



	__S	D0R	D1R	D0S	D1S
N=32	1,016015512	17,69320369	13,84386144	17,04411876	14,64528151
N=16	1,05194408	11,1186231	12,5320228	11,3055875	11,8245243
N=8	0,99362498	7,82352315	7,17576084	7,74241785	7,15012357
N=4	1,0373117	6,28851547	5,68486007	6,27691899	5,8048894
N=2	1,10513495	5,26127921	5,49933041	5,19641315	5,52108589

The better distribution comes at the cost of the delay mentioned in the previous section. Figure 8 shows the delay of the SBC structures in clock cycles, relative to the MTBF of the RSGs used. Naturally, the delay of a linear structure like __S is constant since the RSGs can work in parallel. The delay Del__S, DelD0S, Del D1S of the matrix structures D0S and D1S grows linear with the size. D0S has the worst delay.

Figure 8 Delay of SBC structures



7 Conclusions

In this work, we introduced the abstract concept of a meta-level design for TRNGs, the RBC. The concept was extended by using a corrector to the SBC. Both concepts and combinations were experimentally studied and compared with the reference, an array of RSG. In addition, the SBC was theoretically examined. The results show that the SBC gives an equal distribution of zeros and ones to the expense of a delay equal to $^{1}\!/_{p0p1}$, where $p0$ and $p1$ is the probability that a physical random source produces a zero or one-bit. For size $N = 2$, D1S gives the best distributions during the injection of $1 - n$ stuck-at faults. On average, the D0R with $N = 32$ has the best distribution in comparison with the reference, maximal 17.69 times better. The interesting result is that the SBCs only have relevance for small sizes. Bigger sizes do not result in a better distribution, but more delay. Thus, we can recommend that one should use the D0R for bigger sizes and the D1S or D0S for very small sizes where the delay does not matter. As future work, we consider a hardware implementation to get figures for the footprint and heat dissipation of different RBCs.

References

Baugher, M., McGrew, D., Naslund, M., Carrara, E. and Norrman, K. (2004) *The Secure Real-time Transport Protocol (SRTP)*, Request for Comments 3711, The Internet Society.

Beker, h. and Piper, F. (1982) *Cipher Systems*, Northwood Books, London.

Chaney, T.J. (1983) 'Measured flip-flop responses to marginal triggering", *IEEE Transactions on Computers*, December, Vol. C-32, No. 12, pp.1207–1209.

Fechner, B. and Osterloh, A. (2008) 'A true random number generator with built-in attack detection', *Proc. of the 2008 3rd Int'l. Conf. on Dependability of Computer Systems*, pp.111–118.

Hall, S,H. (2000) *High Speed Digital System Design: A Handbook of Interconnect Theory and Design Practices*, Wiley, 0-471-36090-2.

Hoffman, E. (1998) *Random Number Generator*, 5706218 USA.

http://www.electronicsweekly.com/Articles/2008/10/28/44806/umc-produces-first-28nm-sram-chips.htm (accessed on 11/12/08).

http://www.fourmilab.ch/hotbits/ (accessed on 11/12/08).

http://www.random.org/randomness/ (accessed on 11/12/08).

Johnson, J.W. and Graham, H. (1993) *High Speed Digital Design: A Handbook of Black Magic*, Prentice Hall, 978-0133957242.

Knuth, D. (1997) 'The art of computer programming', *Seminumerical Algorithms*, 3rd ed., Vol. 2, Addison-Wesley, ISBN0-201-89684-2.

Jun, B. and Kocher, P. (1999) 'The Intel random number generator', s.l., Intel, April.

Marsaglia, G. (1996) 'DIEHARD: a battery of tests of randomness', The Marsaglia random number CDROM including the 'Diehard battery of tests of randomness, available at: http://www.stat.fsu.edu/pub/diehard/, accessed on 11/22/07.

Matsumoto, M. and Kurita, Y. (1992) 'Twisted GFSR generators', *ACM Trans. Model. Comput. Simul.*, Vol. 2, No. 3, pp.179–194.

Menezes, A.J., Vanoorschot, P.C. and Vanstone, S.A. (1997) *Handbook of Applied Cryptography*, CRC Press.

Neuenschwander, D. (2004) *Probabiblistic and Statistic Methods in Cryptology*, Springer, LNCS 3028.

Philips (1989) 'A metastability primer', Application Note 219. s.l., Philips Semiconductors.

Quantum Random Bit Generator Service (2007) http://random.irb.hr/ (accessed on 12/12/07).

Rukhin, A., Soto, J., Nechvatal, J. Smid, M., Barker, E, Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J. and Vo, S. (2000) 'A statistical test suite for random and pseudorandom number generators for cryptographic applications', *Special Publication*, Vol. 800, No. 22

Sunar, B., Martin, W.J. and Stinson, D.R. (2007) 'A provable secure true random number generator with build-in tolerance to active attacks', *IEEE Transactions on Computers*, Vol. 56, No. 1.

Walker, S. and Foo, S. (2001) 'Evaluating metastability in electronic circuits for random number generation', *Workshop on VLSI.*