IMPLEMENTING TRUE RANDOM NUMBER GENERATORS IN FPGAS BY CHIP FILLING

Octavian Creţ, Radu Tudoran, Alin Suciu and Tamas Györfi
Technical University of Cluj-Napoca, 15, C. Daicoviciu street, Cluj-Napoca, Romania

Keywords: True random number generators, FPGA, Electrostatic and magnetic interferences.

Abstract: This paper presents a new method for implementing TRNGs in FPGA devices. The design is based on filling the chip close to its maximal capacity and exploiting the interconnection network as intensely as possible. This way, there are strong chances for the design to exhibit a nondeterministic behavior. Our design is a computationally intensive core that generates 64-bit numbers, accumulated into a normal, fixed-point accumulator. From the 64-bit words only those bits are extracted that exhibit the maximal entropy. They are then post-processed using the classical XOR-based bias elimination method. The resulting TRNG provides high quality random numbers; other advantages of this new method are its stability and the fact that the design encapsulates all its components in one chip. An explanation of the observed phenomenon is proposed, based on electromagnetic interferences inside the chip and cross talk. A method for developing new designs based on this approach is also proposed.

1 INTRODUCTION

The increasing need for random numbers is due to the emergence of many application fields where these numbers are indispensable. Random numbers are useful for a variety of purposes, such as generating data encryption keys, simulating and modeling complex phenomena and for selecting random samples from larger data sets. When discussing single numbers, a random number is one that is drawn from a set of possible values, each of which is equally probable, i.e., a uniform distribution. When discussing a sequence of random numbers, each number drawn must be statistically independent from the others.

A supplementary constraint on the random numbers is the throughput at which they are generated. This is also a condition for proving the randomness itself. In some cases, if a generator only yields a few numbers, it is almost impossible to check their statistical properties, because the number of output values is insufficient.

Although it is impossible to prove that a given sequence is random, it is possible to conclude that the sequence is *not* random. This is done by performing some statistical tests grouped in batteries like Diehard (Marsaglia 1996), NIST (Rukhin et al. 2001) and more recently TestU01 (L'Ecuyer and Simard, 2007).

Most true random number generator (TRNG) systems rely on physical phenomena to capture randomness. But after capturing entropy by some analog device, the signals must be sampled and digitized to become useful bits. This hybrid approach has a set of disadvantages: low speed, poor throughput, high vulnerability against harmful attacks (this aspect is important especially in cryptographic applications).

It is thus by far preferable to have the whole system in one digital chip. But it is difficult to get a digital system to do something "by chance". For instance, a computer is (or should be, if not broken) completely predictable.

There is a lot of ongoing research for improving and obtaining new methods to generate true random numbers, based on software (Drutarovsky and Galajda, 2007) and hardware (Gentle, 2004) strategies. There are three main approaches to generating random numbers using a computer, with quite different characteristics:

- *Pseudo Random Number Generators* (PRNGs) are algorithms that use mathematical formulae or simply pre-calculated tables to produce sequences of numbers that appear random. A good deal of research has gone into pseudo-

Cret O., Tudoran R., Suciu A. and Györfi T. (2009).
IMPLEMENTING TRUE RANDOM NUMBER GENERATORS IN FPGAS BY CHIP FILLING.
In *Proceedings of the International Conference on Security and Cryptography*, pages 167-174
DOI: 10.5220/0002177001670174

random number theory. If the algorithm is complex and the period of the PRNG sequence is long enough, it can produce quality numbers. The advantage is that they are easy to implement in software; the major disadvantage is their predictability: for equal external seeds the output sequence will always be the same.

- *Unpredictable Random Number Generators* (URNGs) are algorithms that basically rely on unpredictable human-computer interaction to capture entropy. Such examples are the mouse movement on a given area by a human operator or the amount of time between keystrokes. Even though the source is not a truly entropic one (a good knowledge of the operator's habits can ease the prediction of the next event), the results generated show a good quality and this type of methods are used in several cryptographic products (e.g. PGP). By combining several such entropy sources the results can be improved.
- *True Random Number Generators* (TRNGs) produce the random numbers based on some physical phenomena (e.g. radiation, jitter, ring oscillators, internal noise etc.) that are digitized. They do not have an internal state like the PRNGs and the next generated bit is based entirely on the physical process. In general these bits are not uniformly distributed (the probability of a '1' is not 50%) so they require some post processing in order to reduce the bias.

The main difference between PRNGs and TRNGs can be seen as follows: in a PRNG anyone can predict with 100% accuracy the next state based on the current state, while in a TRNG nobody can do that (even its designer).

At this moment, three main techniques were reported in the literature for creating TRNGs:

- *Ring oscillators* (*ROs*): basically, this method is exploiting the *jitter* of a clock signal in a purely digital design (Kohlbrenner and Gaj, 2004), (Schellekens et al. 2006);
- *Direct amplification of the noise* that is intrinsic in analog signals: this method relies on the amplification of the shot noise, the thermal noise, the atmospheric noise or the nuclear decay. The noise is amplified and then, using comparators and analog-to-digital converters, bits are "extracted" from it (Jun and Kocher, 1999);
- *Chaos-based TRNGs*: this method is based on a well-defined deterministic analog signal that exhibits chaos. Existing implementations exploit Markov's chaotic sources theory (Drutarovsky and Galajda, 2006) and use mixed analog-digital chips.

In this paper we report an empirical discovery: the possibility to obtain high quality TRNGs in FPGAs by almost completely filling the chip with active logic and intensely using the interconnection network. This way, a design that should work deterministically starts exhibiting a nondeterministic behavior. We measure the quality of the random bits obtained this way and propose an explanation of the observed phenomena. Finally, we propose a new method for implementing TRNGs in FPGAs based on these observations.

2 RELATED WORK

There are systematic efforts for creating RNGs both in software and hardware. Few TRNGs exist nowadays in software, because of the problems they face with and due to the reduced number of true sources of randomness. Such problems are for instance the throughput – for generators which rely on user's actions, or the strong dependency on some computer hardware components – e.g. the network card activity. Even a generator which does not depend on such inefficient factors – like the one proposed in (Colesa et al., 2008) – faces difficulties regarding the dependency of the output sequence on the environment of the workstation on which it was obtained.

The hardware methods proved to behave better than the software ones, but they also face sometimes problems like portability or sensibility to physical environment factors.

FPGAs are emerging as an attractive platform for cryptographic implementations, offering benefits such as:

- compactness: the generator is embedded in a single chip;
- fully digital: the ideal source of entropy is a digital signal, not a mixed analog-digital one;
- low development costs;
- reduced time-to market.

One of the first FPGA-based implementations was proposed in (Fischer and Drutarovsky, 2002), where the oscillator jitter was used as the entropy source. The system was based on PLLs (Phase Locked Loop, a component available in Altera FPGAs and used for frequency synthesis and clock skew reduction) in which very fine control of the

output frequency is possible. Unfortunately, this implementation is limited to Altera FPGAs.

A reliable FPGA vendor-independent implementation of TRNGs is based on *ring oscillators* (*ROs*) and on the exploitation of *jitter*. An RO is a circuit composed of an odd number of inverters connected to form a ring (i.e. the output of the last inverter is connected to the input of the first one, in a "circular" manner). The RO can also contain Latches, to allow the fine tuning of the propagation delays inside it. In all implementations, at least a pair of ROs is necessary.

There are two main methods of using ROs for building TRNGs:

- One of the ROs is sampling the other one – this method was proposed in (Kohlbrenner and Gaj, 2004)
- The ROs are working independently, in parallel, and the entropy is collected from them by means of an XOR gate – this method was proposed by (Schellekens et al., 2006) and further developed by (Klein et al., 2008).

3 A NEW METHOD FOR CREATING ENTROPY

The method we introduce here was empirically discovered while working at another design. Even though the underlying phenomenon is not new, the possibility to exploit it in a coherent and useful manner was a surprise.

It is common knowledge between experienced designers that problems can arise when a design fills the FPGA chip almost completely. After 90% of logic resources consumption, strange manifestations can appear: a design that used to work fine becomes totally unpredictable after extending it with more logic blocks. In short, the design becomes nondeterministic after passing a "fill threshold" in the amount of logic resources used.

The concept of "design usage percentage" usually refers to the amount of active logic that is used in the design. In our opinion, for this design one must also consider the concept of "interconnection network usage", with a particular accent on the number of repeaters that are used. The two concepts are related, but the latter is more important in TRNGs design using the method presented in this paper.

Unfortunately, it is not clear where this threshold is positioned, because there are also many designs that behave normally (i.e., completely deterministic, as expected) even if almost 100% of the FPGA chip is used. However, the nondeterministic behavior around 90% of chip usage is a phenomenon that occurs too often to be ignored.

We discovered such a behavior while working on a complex design that implies creating a multi-stage pipeline of arithmetic floating-point operators. This design's block structure is shown in Figure 1.

The floating-point operators used are well-tested, completely reliable (i.e. deterministic) ones, as developed in the FloPoCo project (FloPoCo, 2007). The design was aimed at computing some physical parameters of a medical device for stimulating the human nervous tissue (Creț et al. 2008). We used a Digilent XUP board featuring a Virtex2Pro30 FPGA device and the Xilinx ISE 8.1 software.

At the beginning, the architecture was completely deterministic (all components were tested separately and they clearly manifested a deterministic behavior).

This design implements a very computationally intensive algorithm and the pipeline produces a new number in each clock cycle. Because of some research we did on the final results' precision (De Dinechin et al., 2008), we tried to improve the final accumulator, so we designed a new one.

Of course, the results obtained by the new accumulator had to be compared with those produced by the old one. That is why we added the new accumulator in the design, on 64 bits instead of 32, working in parallel with the old one (Figure 1).

At this point, the "fill threshold" mentioned above was reached, and the design became nondeterministic. The same *.bit* file downloaded into the FPGA device produced each time a different final result, both from the old accumulator and from the new one.

To observe rigorously this phenomenon, we first tested this 150 times: at each time the final result was different. As the final result is on 64 bits, we observed that differences in the final results appeared after the two most significant hexadecimal digits. The same test was done on several identical boards, and in all cases, without exception, the results were different at each run.

It is important to mention that as soon as the additional accumulator (the improved one) was removed, the design became deterministic again. When adding it back, the design became nondeterministic, etc.

Trying to understand what happens, we have created a very simple debugging environment (Figure 2).

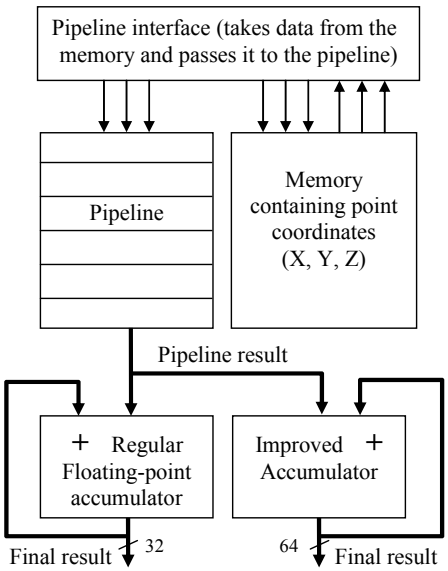

Figure 1: Architecture of the first design.

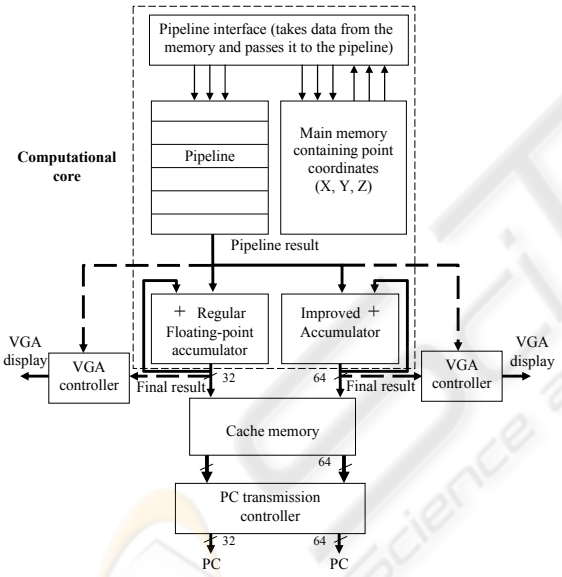

Figure 2: The simple debugging environment.

The intermediate results were captured in the PC and also monitored on a VGA display. The operating mode was simple: the computational core and the transmission module work simultaneously. But the former is faster than the latter; therefore, after the computational core fills the cache memory, it must be stalled until the transmission module empties the cache memory and transmits all data to the PC. After the cache memory is emptied, the computational core is restarted. So, the transmission module works all the time, while the computational core must periodically be stalled. Still, the VGA display allows monitoring the results produced by the computational core before being cached.

During debugging the following abnormal behavior was observed:

- The same one-bit signal was displayed on two different rows of the VGA display: it was '0' on the first row and '1' on the second one, as shown in Figure 3.
- During a functioning cycle while the computational core was stalled, the signals displayed on the VGA screen changed at apparently random moments of time, even though the computational core did not receive clock impulses. Of course, this means the corresponding registers modify their content in the absence of a clock impulse.
- We have displayed the value of the "pipeline result" signal sent to the improved accumulator and to the initial accumulator, and it was not the same value.

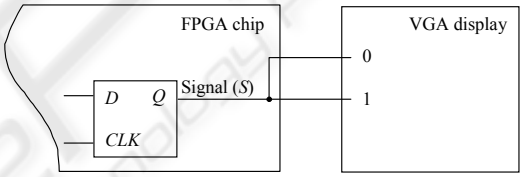

Figure 3: Example of abnormal behavior: the signal S appears to be simultaneously '0' and '1'.

In our opinion, these phenomena are due to cross talk and other electrostatic or magnetic interferences that appear in the interconnection network, as detailed in Section 7.

4 EXPLOITING THE NONDETERMINISM

We reflected about how to use in a creative manner this phenomenon.

As we were doing some research on random numbers (TRNGs) (Suciu, 2007) we asked ourselves if there is any way we could exploit this and capture all the intermediate numbers generated by the computational core, not only the final result. If these numbers have all the necessary statistical properties, the system could be successfully used as a TRNG, because the throughput is relatively high.

Another advantage is the fact that the numbers produced by the computational core and accumulated in the improved accumulator are 64-bit

wide, aspect that allows selecting from them only the groups that change the most.

We had to select numbers that fulfill the following conditions:

- Show significant differences from a run to another – since the design is nondeterministic, how can one be sure that the numbers, even if they pass the statistical tests, are different in consecutive runs?
- Show good statistical properties – i.e. do they pass all the tests from the universally accepted test batteries?

The first question was capital for the quality of the TRNG. We compared successive runs and discovered that at the beginning the numbers generated by the computational core are identical, but after less than 2% of the generated test file they started being different and continued to be different until the end of the execution. We can consider this first period (from the beginning until the first interference appears) a *warm-up* of the TRNG.

After series of testing their statistical properties, we have concluded that not all the bits show the non deterministic behavior with the same rate. The phenomenon that determines the nondeterministic behavior of the design seems to affect more often the middle bits of the Accumulator from 39 to 8 (Figure 4). These will be the TRNG's raw random numbers, because they obtain the best scores when tested.

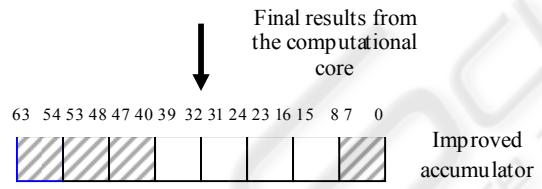

Figure 4: Bits with the best statistical properties from the randomness point of view.

An interesting aspect is that nondeterminism can not be located in the architecture: it is impossible to say that it appears in the computational core, in the accumulators or in the PC transmission module. We have tested several variants of PC transmission modules (RS-232 and Ethernet), and several accumulators (we added more "improved accumulators" in the design, in parallel with the ones from Figure 2, but we didn't operate changes at the level of the computational core). Every time, for all combinations, the nondeterministic character of the whole design was preserved.

5 BUILDING A HIGH QUALITY TRNG

Based on the above described phenomenon, we developed a hardware TRNG. The randomness shown by the nondeterministic design is still not enough to pass rigorous statistical tests like NIST and TestU01. We can thus consider these numbers to be raw numbers with high entropy, but the level of entropy is still insufficient to make them acceptable as high quality random numbers. Therefore, we had to add a post-processing unit to improve the quality of the numbers produced by the architecture.

The most popular post-processing methods nowadays are the von Neumann (Jun and Kocher, 1999) and the XORing (Fischer and Drutarovsky, 2002) methods. We opted for the XOR method, which takes n bits and XORs them together, thus producing one bit. XORing is a popular post-processing method offering multiple advantages, like bias reduction and entropy development (by XORing deterministic with non-deterministic bits, the result will be non-deterministic).

If XORing is applied on more than two bits, even greater improvements can be achieved. The drawback of XORing is that it reduces the bit generation rate by a factor of n (where n is the number of bits XORed).

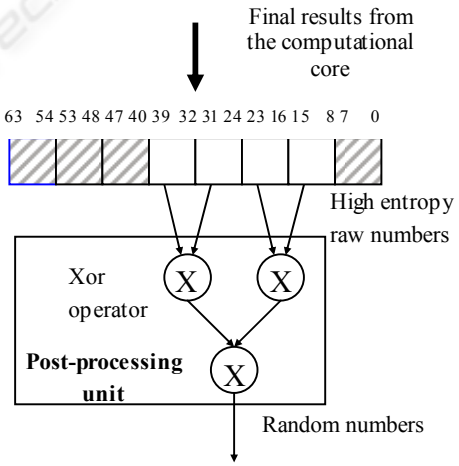

Figure 5: The XOR-based post-processing unit.

The design was thus extended with a XOR-based post-processing block, as shown in Figure 5. As one can notice, only the groups of bits having the best chances to pass the statistical tests were selected to enter the post-processing unit. We have used only the results produced by the improved accumulator.

After the end of the exploratory steps, the debugging environment (the VGA display) was no

longer necessary. We noticed that removing the debugging environment and adding the post-processing unit does not affect the system's nondeterministic character – Figure 6.

6 TESTING AND RESULTS

The proposed TRNG possesses a strong physical source of entropy that is completely reliable and shows some interesting additional features.

We have developed this TRNG on a Virtex 2 Pro FPGA device. The random behavior was observed on 150 tests of the same architecture (the same .BIT file was downloaded on the FPGA chip and the application was launched 150 times); all tests yielded different final results (a short excerpt is presented in Table 1).

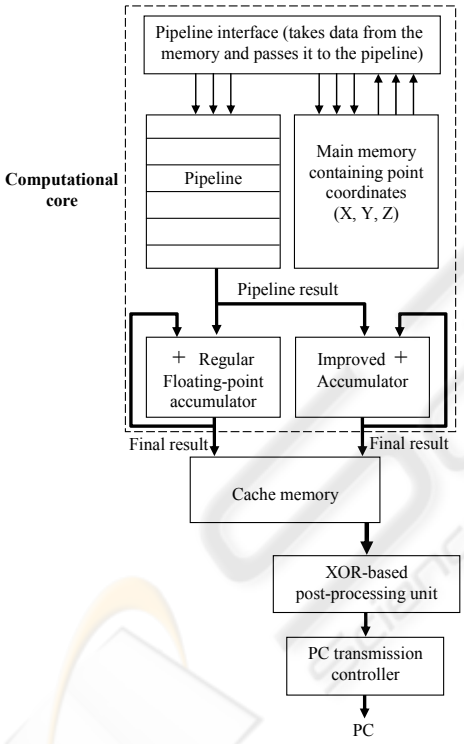

Figure 6: Complete scheme of the TRNG.

All tests were done on several Digilent XUP V2P30 boards (so this phenomenon does not occur only on one particular FPGA chip). The working frequency was of only 45MHz, which excludes overclocking as a source of entropy.

We observed that the design is nondeterministic: both the final value obtained in the improved accumulator and the sequence of intermediate numbers are different from a run to another.

Table 1: Short excerpt of the results generated by the nondeterministic design.

Test no.	Final value in the improved accumulator	Final value in the initial accumulator
1	43EF5C23E955001C	4BB7D363
2	43EA36059DDAFEFF	4BB7D99D
3	43E3F5FEFBA49059	4BB7BD16
…	…	…
150	43D664FABB240B66	4BB7A1DE

We have also compared consecutive runs (separated by a physical shut-down of the generator) and computed the similarities between the numbers occurring on the same position in each sequence. If the generator is non-deterministic, then the proportion of identical bits from different runs should be equal with the probability that two random bits are equal, i.e. 50%. In Figure 7 we present this proportion.

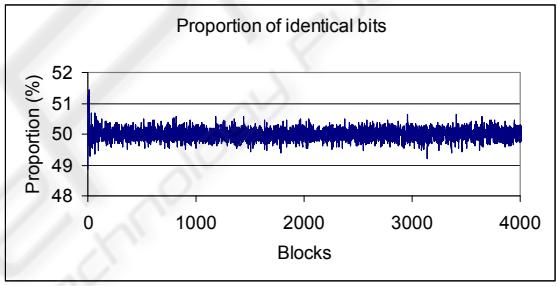

Figure 7: Proportion of identical bits in consecutive runs (separated by a shutdown).

This proves that the TRNG does not tend towards the same sequence of numbers, giving the possibility of using it for an infinite period of time. The generator has some coincidences at the beginning, but after a very short warm up period of approximately 100K numbers (with very small variations from a run to another), the sequence of values is clearly different. The length of the warm up period is very small compared to the running period of the generator.

This offers a great entropy source since neither the occurrence of the phenomenon or its amplitude (locally or globally) cannot be predicted.

For validating the results the NIST (Rukhin et al. 2001) and TestU01 (L'Ecuyer and Simard, 2007) tests were used. We have applied these tests on more than 40 sequences of outputs, testing different lengths (between 256 KB to 200 MB). The outputs were obtained from several Virtex 2 Pro FPGA boards, since the architecture does not rely on any isolated malfunction of one particular board. More

than 98% of the sequences passed all the NIST tests (there were situations when some instances of the tests weren't passed, but the number of failed instances is limited – 5 out of 186 instances), and also all the TestU01 tests. The failed instances were not the same, and also the number of instances varies, so there isn't any pattern in the TRNG which causes this. Statistically, even a TRNG can produce sequences of outputs on which isolated tests can fail, because of what looks to be a pattern, but since this doesn't repeat on multiple executions (different outputs), this phenomenon is considered normal and doesn't question the quality of the RNG.

7 SOURCE OF RANDOMNESS

Our method consists in filling the chip closed to its maximum capacity (using almost all its logic and interconnection resources). This way, the interconnection network will be used close to its maximal routing capacity. It is well known that the interconnection network occupies about 90% of the physical space in an FPGA chip; if used close to its maximal capacity, the electrostatic field can produce serious interferences. Figure 8 shows the device utilization summary produced by Xilinx tools.

Selected Device:	2vp30ff896-6			
Number of:				
Slices:	13380	out of	13380	100%
Flip Flops:	15350	out of	27392	56%
4 input LUTs:	24156	out of	25427	88%
BRAMs:	130	out of	136	95%

Figure 8: Synthesis report of the design.

We believe that the nondeterminism is caused by electrostatic and/or electromagnetic interferences generated by the full usage of the interconnection network. It could be a cross talk between parallel lines as well as a global influence caused by the electrostatic field.

A strategy consisting of filling the FPGA chip close to its maximal capacity increases the chances for this phenomenon to take place. The happening of this phenomenon does not ensure that the values carried by the wire will always be modified. Its influence can also be just a local one, so two different receivers can read distinct values from the same connecting wire. The uncertainties which rise from this physical phenomenon ensure us with a very good source of entropy.

Another possible explanation (although we tend to give credit to the first one) would be that the maximal fan-out of the FPGA chip is exceeded because of the large number of Flip-Flops used in the design.

We believe this could constitute a novel methodology for designing TRNGs in FPGAs:

1. Create a design that performs intense computations (preferably a pipeline on at least 64 bits) and produces a new number in each clock cycle.
2. Make this design big enough to fill the FPGA chip close to its maximal capacity (use as many slices and Flip-Flops as possible).
3. Try to use the interconnection network at its full capacity. This can be done by setting the routing effort to "low" in the synthesis tools and by an adequate usage of the placement constraints (LOC, RLOC etc.). Use the "one sender-multiple receivers" model.

If the fill threshold is reached, the design has good chances to become a high quality TRNG.

The classical methods based on ROs are very sensitive to a fine tuning of the ROs, which should have almost equal periods (Kohlbrenner and Gaj, 2004). It is known that ROs are sensitive to temperature; this can seriously affect the quality of the generated random numbers. On the contrary, in the present design the entropy level increases with the temperature or any other extreme condition which could influence the chip (like radiations, etc.).

8 CONCLUSIONS AND FURTHER WORK

We have presented a new way of implementing TRNGs in FPGA devices. Our design is based on filling the chip close to its maximal capacity, from the Flip-Flops and slices point of view, and exploiting the interconnection network as intensely as possible. This way, the design becomes nondeterministic.

The design is a computationally intensive core that produces 64-bit numbers, accumulated into a normal, fixed-point accumulator. From the 64-bit words we extract only those bits that exhibit the maximal entropy and post-process them using the classical XOR-based bias elimination method. We consider this an interesting way of exploiting a phenomenon that otherwise is neglected or avoided by most designers.

The resulting TRNG was proven to provide high quality random numbers and we also believe it has the advantage of resisting to extreme functioning conditions (temperatures and radiations), which can only increase its quality. Other advantages of this new method are its stability and the fact that the design encapsulates all its components in one chip, thus increasing the generator's security. Since it does not depend on any external factors, an attacker cannot intervene in any way to study it in order to make any prediction about the source of randomness.

The design (in the form of a .BIT file) can be freely downloaded from (Suciu, 2007) together with an Installation Guide, so it can be tested by anyone who wants to convince him/herself about its nondeterministic behavior.

We have also proposed a method for developing new designs based on this approach, which are FPGA vendor independent. The only drawback of this method is that the FPGA chip will be used at its full capacity, which will make it impossible to implement anything else in the same chip.

Future work will focus on constructing a generic, device-independent architecture which could be applied to any FPGA by only modifying the generic variables in order to completely fill the chip. Another research direction will be to compare this TRNG with other generators, when exposed to external factors (temperature variations, radiations, current fluctuations) to determine the stability of each method.

ACKNOWLEDGEMENTS

This work was supported by the Romanian National Centre for Program Management (CNMP) under grant nr. 11-020/2007 (the CryptoRand project).

REFERENCES

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

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, 2001. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. NIST Special Publication 800-22 (with revisions dated May 15, 2001). http://csrc.nist.gov/rng/SP800-22b.pdf.

L'Ecuyer, P. and Simard, R., 2007. TestU01: A C library for empirical testing of random number generators. In ACM Transactions on Mathematical Software, 33(4):22.

Drutarovsky, M. and Galajda, P., 2007. A robust chaos-based true random number generator embedded in reconfigurable switched-capacitor hardware. In Radioelektronika, April 2007.

Gentle, E. J., 2004. Random Number Generation and Monte Carlo Methods. Springer.

Kohlbrenner, P. and Gaj, K., 2004. An Embedded True Random Number Generator for FPGAs. In Proceedings of the ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays, Monterey, California, pp. 71-78.

Schellekens, D., Preneel, B. and Verbauwhede, I., 2006. FPGA Vendor Agnostic True Random Number Generator. In Proceedings of the International Conference on Field Programmable Logic and Applications, Madrid, pp. 1-6.

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

Drutarovsky, M. and Galajda, P., 2006. Chaos-based true random number generator embedded in a mixed-signal reconfigurable hardware. Journal of Electrical Engineering, vol. 57, no. 4, pp. 218-225.

Coleşa, A., Tudoran, R. and Bănescu, S., 2008. Software Random Number Generation Based on Race Conditions. In Proceedings of the 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing SYNASC'08, Timişoara, Romania.

Fischer, V. and Drutarovsky, 2002. True random number generator embedded in reconfigurable hardware. In Proceedings of the Cryptographic Hardware and Embedded Systems Workshop (CHES), pp. 415-430.

Klein, C., Creţ, O. and Suciu, A., 2008. Design and Implementation of a High Quality and High Throughput TRNG in FPGA. In Proceedings of DASIP'08 Conference on Design and Architectures for Signal and Image Processing, Université libre de Bruxelles, Belgium, pp. 52-56.

FloPoCo project, 2007: http://www.ens-lyon.fr/LIP/Arenaire/Ware/FloPoCo/ .

Creţ, O., Trestian, I., De Dinechin, F., Darabant, L., Tudoran, R. and Văcariu, L., 2008. Accelerating The Computation of The Physical Parameters Involved in Transcranial Magnetic Stimulation Using FPGA Devices. In *Romanian Journal of Information, Science and Technology*, vol. 10, no.4, pp. 361-379.

De Dinechin, F., Detrey, J., Creţ, O. and Tudoran, R., 2008. When FPGAs are better at floating-point than microprocessors. Sixteenth ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, California.

Suciu, A., 2007. The CryptoRand project http://crypto-rand.utcluj.ro.