High performance physical random number generator

K.H. Tsoi, K.H. Leung and P.H.W. Leong

Abstract: A field programmable gate array (FPGA) -based implementation of a physical random number generator (PRNG) is presented. The PRNG uses an alternating step generator construction to decorrelate an oscillator-phase-noise-based physical random source. The resulting design can be implemented completely in digital technology, requires no external components, is very small in area, achieves very high throughput and has good statistical properties. The PRNG was implemented on an FPGA device and tested using the NIST, Diehard and TestU01 random number test suites.

1 Introduction

Random number generators (RNGs) are an important primitive widely used in simulation and cryptography. A physical random number generator (PRNG) derives its output from a physical noise source and its output is non-deterministic in nature. Given the importance of random number generation, surprisingly few hardware implementations of PRNGs have been reported. There are three commonly used techniques in the literature, namely oscillator sampling, direct amplification and discrete time chaos. In the oscillator sampling approach, period variation (i.e. oscillator jitter) in a low-frequency clock of low quality factor (Q) is exploited by using it to sample a high-frequency clock. The direct amplification technique digitises thermal or shot noise, using an amplifier and comparator. Finally, chaotic systems can be used to produce PRNGs.

A high performance PRNG which uses a physical random source to control two linear feedback shift registers (LFSRs) in a manner similar to that of an alternating step generator (ASG) stream cipher is proposed. This approach combines some of the benefits of both approaches and achieves high throughput, small area and good randomness properties. The same approach could be applied to combine other weak physical random number generators with a stream or block cipher.

In 1984, Fairfield *et al.* [1] developed the first integrated RNG based on oscillator phase noise. In the design, a high-frequency oscillator was sampled using a low-frequency oscillator. After removing duty cycle biases via a parity filter, the flip flop output was fed into a LFSR-based scrambler. The design generated 27 bps using a 1000 Hz low-frequency clock. The Intel RNG is part of the Intel 8xx chipset starting with the Intel 810 and is implemented in the Intel 82802 firmware hub device. It uses amplified thermal noise to drive a voltage controlled oscillator (VCO), and oscillator sampling is used to detect the phase noise of the VCO to produce a digital random source [2].

We have previously reported an FPGA design which employs oscillator sampling [3]. In this design, a low-frequency RC oscillator was used to sample an internal high-frequency clock. The design requires only three external passive components to control the time constant of the resistor-capacitor (RC) oscillator. Phase noise in the RC oscillator produced randomised output which was filtered through a parity filter. A disadvantage of this approach is that the output rate is limited by the speed of the RC oscillator and in order to pass the NIST and Diehard tests, the maximum rate was 4.7 kbps. The only other FPGA-based implementation was one by Fischer and Drutarovsky [4] which used a variation of oscillator sampling. Their design was based on the randomness of jitter in an analogue phase locked loop (PLL) and a decimator was used to ensure that at least one sample affecting jitter was included in every output data. The design was implemented on an Altera APEX EP20K200-2X FPGA with a 33.3 MHz external clock. With an 88.245 MHz internal clock, it can generate 69 kbps. For FPGAs such as the Altera APEX E and APEX II devices which have internal PLLs, this approach requires no external components. The disadvantage of this approach is that few FPGAs have this feature.

PRNGs based on chaotic systems can lead to very compact complementary metal oxide semiconductor (CMOS) implementations. In 2001, Stojanovski *et al.* [5] implemented an analog chaos-based RNG in a 0.8 μm CMOS process utilising switched current techniques. The estimated output bit rate of this design was 1 Mbps. Gerosa *et al.* [6] also implemented a RNG based on a chaotic system. Their design with a pipelined ADC (analog-to-digital converter) occupied 2.2 mm^2 silicon area and the design can generate 8-bit data using a 20 MHz clock. Petrie and Connelly combined oscillator sampling, direct amplification and discrete time chaos to produce an analog very large scale integration (VLSI) chip which was robust to power supply noise and substrate signal coupling [7]. Implemented in 2 μm CMOS, the chip could produce random numbers at 1.4 Mbps. The design occupied an area of 1.5 mm^2 and dissipated 3.9 mW of power.

In comparison with the approaches described above, the design presented in this paper, an output rate of 400 Mbps can be achieved on a Xilinx XCV300-8 devices and the design occupies approximately 130 Xilinx Virtex slices. Furthermore, it can be implemented entirely in digital technology with no external components.

© The Institution of Engineering and Technology 2007

doi:10.1049/iet-cdt:20050173

Paper first received 30th October 2005 and in revised form 10th January 2007

The authors are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, NT, Hong Kong

E-mail: phwl@cse.cuhk.edu.hk

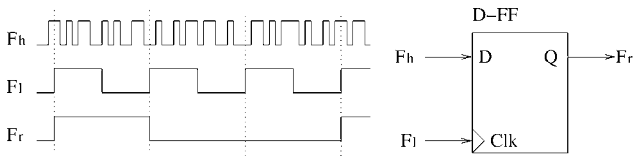

Fig. 1 *Oscillator sampling using D-type flip-flop*

The rest of the paper is organised as follows: the architecture of the PRNG and its FPGA implementation are presented in Section 3. The performance of the design and the quality of the resulting output is reported and evaluated in Section 4. Conclusions are drawn in Section 5.

2 Background

2.1 Oscillator sampling based physical noise source

Oscillator sampling based noise sources typically use a low-frequency clock (F_l) with large phase noise to sample an accurate high-frequency clock (F_h), producing an output (F_r) as shown in Fig. 1. If the phase noise of F_l is of the same order as the period of the high-frequency clock, an output which is random is obtained [1]. However, since the output rate of this approach is that of the low-frequency clock, the output rate of this PRNG is determined by the frequency of F_l. If the frequency of F_l is increased to improve the output rate, the phase noise usually decreases, leading to correlations in the output.

There are several factors which affect the randomness of the output [1]. The first is that the duty cycle of F_h may not be 50%. In this situation, F_r will have unequal probability of being zero or one. An N-bit parity filter [1, 8] can be used to deskew a non-uniform distribution. If the ratio of ones to zeroes in the raw random bitstream is p:q then the probability that the parity will be one or zero is the sum of the odd or even terms of the binomial expansion of $(p+q)^N$. This sum can be evaluated to calculate the probability of a one at the output of the parity filter and is $1/2[(p+q)^N + (p-q)^N]$. Since $p+q=1$, this expression reduces to $1/2[1+(p-q)^N]$. As N increases, this expression tends to 0.5.

The second factor is the selection of clock frequency. The period of the generated clock changes because of oscillator phase noise. If the variation in F_l's period is not large enough, there will be correlation between bits and so the value of the output can be predicted to some extent from the previous values. Previous research has suggested that the standard deviation of the period variation of F_l should at least be 0.75 times the period of F_h to reduce bit to bit correlation [1]. Increasing F_h and reducing F_l leads to more randomness.

A third factor affecting the quality of the RNG is the random source itself. As both periodic and aperiodic electromagnetic noise exists inside a computer system, there may be correlation in the output sequence as the result of coupling of periodic noise from the power supply, clocks, crosstalk, thermal effects and so on. This issue is not addressed in this work.

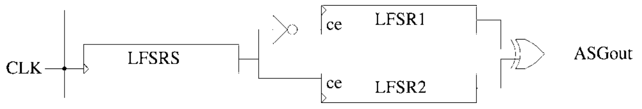

Fig. 2 *Alternating step generator*

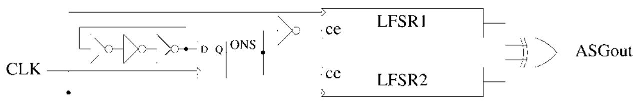

Fig. 3 *Proposed PRNG circuit*

2.2 Alternating step generator

The ASG is constructed from three LFSRs as shown in Fig. 2 [9, 10]. The binary output of the selection LFSR (LFSRS in the figure) is used to select whether LFSR1 or LFSR2 is clocked. The output of the ASG is the exclusive-or (XOR) of the output of LFSR1 and LFSR2. The characteristic polynomials of LFSR1 and LFSR2 are irreducible and different. In addition, the greatest common divisor of the periods of LFSR1 and LFSR2 should be equal to 1.

Several attacks on the ASG have been proposed. If the connection polynomials of LFSR1 and LFSR2 are primitive trinomials, the generator can be attacked using the linear syndrome method [11]. In our design, a high Hamming weight polynomial was chosen to prevent this attack. Golic proposed an attack based on the edit distance [12]. This attack requires computing the edit distance for every possible pair of initial states of LFSR1 and LFSR2 and is hence not practical for large shift register lengths (approximately 127 in our case).

3 Architecture and implementation

In the proposed approach, a physical noise source, hereafter called the oscillator noise source (ONS), is produced by oscillator sampling as shown in Fig. 3. The high-frequency clock, F_h, is generated using a three-inverter ring oscillator implemented in a single Xilinx Virtex slice, whereas the low-frequency oscillator input comes from the system clock (133 MHz) in our tested configuration. These two signals are combined using an edge-triggered D-type flip-flop to produce a non-deterministic but correlated random output. This output is used instead of the selection LFSR of an ASG.

In order to achieve a high output rate, the ONS should produce outputs at the same rate as the system clock. This is normally derived from a crystal-controlled oscillator and has low phase noise. Hence the system clock should be connected to the clock input of the D type flip-flop (as shown in Fig. 3), and a high-frequency oscillator connected to the D input. For the FPGA implementation, a high-frequency ring oscillator was used. Ring oscillators are commonly used for PLLs, clock recovery circuits and frequency synthesisers, but have high phase noise compared with circuits employing passive resonant components [13]. Thus they combine the advantages for this application

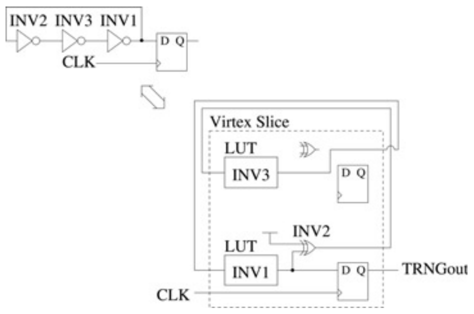

Fig. 4 *Xilinx virtex ring oscillator implementation*

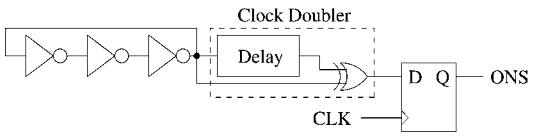

Fig. 5 *Clock doubler circuit*

of being implementable entirely within an FPGA with that of high phase noise.

It is desirable to make the frequency of the ring oscillator as high as possible in order to reduce the correlation resulting from sampling the ring oscillator with the system clock. A naive implementation would require three lookup tables (LUTs) and hence 1.5 Xilinx Virtex slices [14]. The FPGA implementation used an additional two-input XOR gate present in the Xilinx Virtex slice to reduce the implementation to 1 Virtex slice as shown in Fig. 4. This has the advantage of higher speed because wiring is reduced and the XOR gate is faster than a LUT.

The LFSRs were implemented using the SRL16 [14] feature of the Xilinx Virtex chip which enables a 1–16 stage shift register to be implemented in a single LUT.

3.1 Clock doubler

As discussed in Section 2, increasing the high-frequency clock, F_h, improves the randomness of the ONS output. It is possible to apply a clock doubler to the output of the ring oscillator as shown in Fig. 5. The Poker test in the NIST testsuite [15] was used to observe the effect of different delay values for the clock doubler, and the results are shown in Fig. 6. This test is quantitative and a low figure implies better randomness. The Poker test is passed if the result is between 1.03 and 57.4 [10]. As can be seen, small and large values of the delay do not result in clock doubling and the Poker test results are large. The test results show a significant improvement for delay values of approximately 2.5 ns as reported by the Xilinx timing analyser. Table 1 shows a comparison of the best Poker test results with and without a clock doubler. Note that although the clock doubler offers an improvement, the ONS output does not pass the Poker test.

4 Results

An implementation of the PRNG was synthesised and implemented using the Xilinx ISE 8.2i software. The LFSRs were implemented as a chain of 16 bit shift register

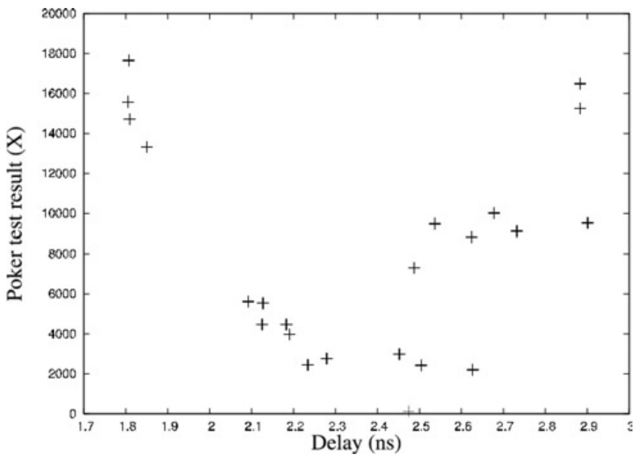

Fig. 6 *Poker test results as a function of the clock doubler delay*

Table 1: Comparison of poker test results with and without a clock doubler

Delay, ns	Poker test result
0	1579.77
2.474	124.013

primitives (called SRL16 blocks) in the FPGA device to achieve high performance and density. The FPGA platform used was a Pilchard FPGA card [16], which employs the synchronous dynamic random access memory (SDRAM) bus instead of the peripheral component interconnect (PCI) bus used in conventional FPGA boards. The FPGA device used was a Xilinx Virtex XCV300E-8 device. The LFSRs were chosen so as to have a random irreducible connection polynomial of degrees 127 and 129 with approximately the same number of 0 and 1 coefficients [9, 10]. Several different polynomials were tested. The initial states of the LFSRs were random numbers with approximately an equal number of 1's and 0's.

Table 2 summarises the resource utilisation and performance of the PRNG including a host interface to read back the data. The high-frequency clock of the PRNG can operate at over 400 MHz, but experiments described in this paper used a 133 MHz clock so that the output sequence could be collected via the SDRAM interface of the host computer. As reported by the Xilinx timing analysis tool, the minimum ring oscillator frequency was 800 MHz.

Since the ONS output of the clock doubler improves on the randomness, results reported below are without the clock doubler (i.e. the delay was set to 0). It was also verified that the implementation passes the below tests when an appropriate delay for the clock doubler was added. This increases confidence that the design will operate correctly even if the delay of the clock doubler is set to an inappropriate value.

4.1 NIST test suite

For the NIST test suite (version 1.5), all parameters were set according to the recommendations in [17] and the test sequences were 1 Mbit in size. The sample size, that is test sequences used in the tests, was 100. Table 3 summarises the NIST test results for the PRNG. The significance level α was chosen to be the default of 0.01 (99% confidence) so a test has passed if the p-value is larger than this number. The *pass rate* is proportion of the 100 binary sequences that passed the test. It can be seen that the PRNG passes all NIST tests.

4.2 Diehard test suite

Although the Diehard test suite is one of the most comprehensive publically available sets of randomness tests, unfortunately there are no well-defined pass criteria. Intel assumed that the entire suite passes with a 95% confidence interval for p-values between 0.0001 and 0.9999 [18], and this method was used for our testing. The Diehard test results are summarised in Table 4. If multiple p-values are

Table 2: Implementation summary (Xilinx XCV300E-8)

Design	Period, ns	Slices (% XCV300)	BRAM (% XCV300)
PRNG	7.482	129 (4%)	4 (12%)

Table 3: NIST RNG test result summary for the PRNG

Test	*p*-value	Pass rate
frequency	0.145326	0.9900
block frequency	0.657933	0.9700
cusum-forward	0.383827	1.0000
cusum-reverse	0.867692	1.0000
runs	0.289667	0.9700
long run	0.759756	0.9900
rank	0.514124	0.9900
FFT	0.779188	1.0000
aperiodic templates	0.657933	0.9600
periodic templates	0.289667	0.9900
universal	0.162606	1.0000
approximate entropy	0.924076	0.9900
random excursions	0.637119	0.9565
serial1	0.534146	1.0000
serial2	0.262249	1.0000
lempel Ziv	0.616305	0.9900
linear complexity	0.637119	1.0000

in the result, the worst case value is presented. The PRNG random sequence passes the Diehard test.

4.3 TestU01 test suite

TestU01 [19] is a set of C libraries for RNG performance evaluation. We developed programs to test our RNG results using this library. The random data were stored in a file and then read in as an external RNG source. The RNG passes the *Rabbit*, *Alphabit*, *SmallCrush* and *Crush* test batteries.

Table 4: Diehard RNG test result summary

Test	*p*-value
birthday spacings	0.310619
overlapping 5-Permutation (chisqr 66.743792)	0.994677
overlapping 5-Permutation (chisqr 107.948832)	0.253086
binary rank (31 × 31)	0.155
binary rank (32 × 32)	0.080
binary rank (6 × 8)	0.051318
bitstream	0.008018
OPSO	0.996754
OQSO	0.011809
DNA	0.050285
steam count-the-1	0.066896
byte count-the-1	0.040476
Parking lot	0.921990
min. distance	0.496703
3D spheres	0.016095
squeeze	0.456598
overlapping sums	0.080856
runs up	0.053444
runs down	0.738119
craps	0.985720

5 Conclusion

A new RNG was introduced. This circuit combines a physical random number source with a high speed stream cipher to produce a physical noise source based RNG with small area, high output rate and good statistical properties. This RNG would be suitable for simulation and cryptographic applications although for the latter, caution should be taken since the design is new, and it may be possible to attack the ASG construction given that the ONS is weakly correlated.

6 References

1 Fairfield, R.C., Mortenson, R.L., and Coulthart, K.B.: 'An LSI random number generator (RNG)'. Advances in Cryptography: Proc. of Crypto 84, 1984, (Springer-Verlag), pp. 203–230 LNCS 0196

2 Jun, B., and Kocher, P. White paper by Cryptographic Research Inc., 'The Intel random number generator', 1999, ftp://download.intel.com/design/security/rng/CRIwp.pdf

3 Tsoi, K., Leung, K., and Leong, P.: 'Compact FPGA-based true and pseudo random number generators'. Proc. IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM), 2003, pp. 51–61

4 Fischer, V., and Drutarovsky, M.: 'True random number generator embedded in reconfigurable hardware'. Proc. Cryptographic Hardware and Embedded Systems Workshop (CHES), 2002, pp. 415–430

5 Stojanovski, T., Pil, J., and Kocarev, L.: 'Chaos-based random number generators. Part II: practical realization', *IEEE Trans. Circuits Syst. – I: Fundam. Theory Appl.*, 2001, **48**, pp. 382–385

6 Gerosa, A., Bernardini, R., and Pietri, S.: 'A fully integrated 8-bit, 20 MHz, truly random numbers generator, based on a chaotic system'. SSMSD. 2001 Southwest Symp. on Mixed-Signal Design, 2001, pp. 87–92

7 Petrie, C., and Connelly, J.: 'A noise-based IC random number generator for applications in cryptography', *IEEE J. Solid-State Circuits*, 2000, **47**, (5), pp. 615–621

8 Eastlake, D., Crocker, S., and Schiller, J.: 'Randomness recommendations for security', *Network Working Group*, 1994 **RFC 1750**

9 Gunther, C.: 'Alternating step generators controlled by de Bruijn sequences'. Advances in Cryptology: Proc. Eurocrypt, 1988, vol. 87, pp. 5–14

10 Menezes, A., van Oorschot, P., and Vanstone, S.: 'Handbook of applied cryptography' (CRC Press, 1997)

11 Zheng, K., Yeng, C., and Rao, T.: 'An improved linear syndrome algorithm in cryptanalysis with applications'. Advances in Cryptology: Crypto '90, 1991 **LNCS 537**, pp. 34–47

12 Golic, J., and Menicocci, R.: 'Edit distance correlation attack on the alternating step generator'. Advances in Cryptology: Crypto '97, 1998, pp. 499–512

13 Razavi, B.: 'A study of phase noise in CMOS oscillators', *IEEE J. Solid-State Circuits*, 1996, **31**, (3), pp. 331–343

14 Xilinx Virtex 2.5 V field programmable gate arrays 2000

15 U.S. Department of Commerce, Security Requirements for Cryptographic Modules. Federal Information Processing Standards Publication FIPS 140-1 1994

16 Leong, P.H.W., Leong, M.P., Cheung, O.Y.H., Tung, T., Kwok, C.M., Wong, M.Y., and Lee, K.H.: 'Pilchard – a reconfigurable computing platform with memory slot interface'. Proc. IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM), 2001, pp. 170–179

17 Rukhin el., A. NIST Special Publication 800-22, 'A statistical test suit for random and pseudorandom number generators for cryptographic applications', 2001

18 Intel Platform Security Division: 'The Intel random number generator'. Intel technical brief, 1999. ftp://download.intel.com/design/security/rng/techbrief.pdf

19 L'Ecuyer, P., and Simard, R.: 'TestU01: A C library for empirical testing of random number generators', *ACM Trans. Math. Software*, 2007, **33**, 4, To be published