High Speed True Random Number Generators in Xilinx FPGAs

Catalin Baetoniu[1]

[1] Xilinx, Inc., 2100 Logic Drive
San Jose, CA 95124-3400, USA

catalin.baetoniu@xilinx.com

Abstract. This paper introduces a method of generating true random numbers using only digital logic. It shows that asynchronous sampling the output of ring oscillators can be used as a source of random numbers in digital hardware and that Linear Hybrid Cellular Automata can be used as scramblers to substantially improve the quality of the random data. A highly efficient and scalable FPGA implementation of the proposed circuit has been tested in hardware and shown to generate 32-bit true random numbers at rates over 16Gbps. The generated random numbers have been tested with the DieHard test suite and passed all 15 randomness tests.

1 Introduction

Generating cryptographic quality random numbers either in software or in hardware is not a simple task. Producing such random numbers at very high data rates makes the challenge even harder. For many applications pseudo-random numbers can be used and many good implementations exist that can produce sequences of such numbers at high data rates, both in software and in hardware. However, for certain cryptography applications, true random numbers are required and they have to pass very rigorous tests to be used confidently

It is generally accepted that true random numbers cannot be generated with a software algorithm and for hardware implementations a physical source of analog noise is required as a primary source of randomness. This paper will present a solution to this problem that uses only digital logic and does not rely on any external analog components that could become subject to an attack and compromise the security of the generated numbers. A very efficient implementation in Xilinx FPGAs that can produce 32-bit random numbers at rates of up to 500MHz is presented.

2 Ring Oscillators as a Source of Digital Random Data

By their very nature, pure digital circuits cannot derive randomness from noise in the amplitude domain of a signal but there is a potential source of randomness in the time domain if we use asynchronous logic circuits. The simplest such circuit one could build is a ring oscillator (Fig. 1 shows a few examples). One or more logic gates connected in a loop will oscillate if the total number of logic inversions is odd. In particular, an inverter with its output connected back to its own input will oscillate at a very high frequency (in the GHz range for modern digital logic). A two input XOR gate can use the second input to turn on and off the ring oscillator. When the control input is high we have an inverting one-gate ring oscillator. When the control input is low the output will be locked either high or low.

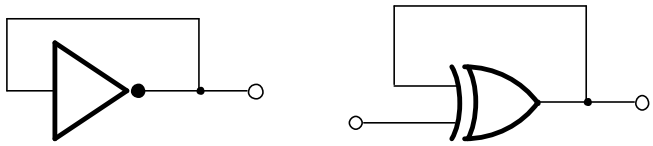

Fig. 1. Simple ring oscillators can be built using inverters and XOR logic gates

The frequency of the output of the ring oscillator is determined by the gate propagation delay and by routing delays. It is not very stable and is affected by temperature and voltage variations and by system noise. By sampling the output of a ring oscillator with another asynchronous clock like in Fig. 2 we get a truly random binary value.

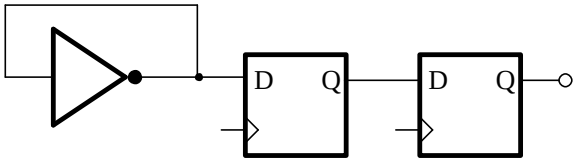

Fig. 2. A sampled ring oscillator will produce a random bit value (double sampling is required to avoid metastability)

There are a few problems with this approach. The first is that the sampling flip-flop can become metastable so its value cannot be used reliably in more than one place. Double sampling the data like in Fig. 2 can easily solve this problem, providing a stable and valid logic value. The metastabiltity itself is a good source of random data and for once it is an asset rather than a liability. The second and more important problem is the correlation between consecutive samples. Unless the sampling rate is a few orders of magnitude slower than the ring oscillator frequency the output data will not be truly random and producing high data rate streams of random data becomes difficult. If the output of the ring oscillator doesn't have perfect symmetry (and it never has) the probabilities of the output random bit of being 0 or 1 will not be equal. We would also like to produce a word of random data (typically 32 bits)

per clock, not just one random bit at a time. We can use two techniques, generalized ring oscillators and Linear Hybrid Cellular Automata to address all these issues.

2.1 Generalized Ring Oscillators

By interconnecting multiple XOR based ring oscillators together as shown in Fig. 3, a generalized ring oscillator with an n-bit output can be created. The proposed circuit topology uses n XOR based ring oscillators and connects each one with its two neighbors creating a ring of rings. This keeps all the interconnections local and makes the circuit scalable, achieving the highest possible oscillation frequency, independent on the actual size of the ring. Another important aspect is the fact that one of the n XOR gates must in fact be an inverting XNOR, as this will prevent the oscillator from becoming trapped into a constant value state. This makes sure no such stable state exists and is similar to the requirement of having an odd number of logic inversions in a simple ring oscillator.

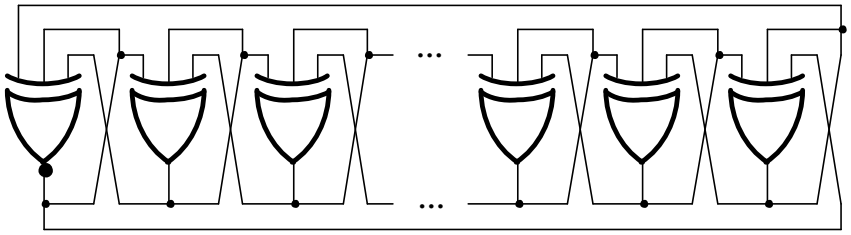

Fig. 3. A generalized ring oscillator will produce a random n-bit word by exhibiting chaotic behaviour

Due to the slight differences between the logic delays of different paths trough the XOR gates and to the noise present in any digital circuit the output of this ring oscillator will exhibit chaotic behaviour. Fig. 4 is an oscilloscope screen capture of two adjacent output bits of a 32-bit ring implemented in a Xilinx FPGA. The output pins used to probe the signals have limited bandwidth and do not show accurately the very high speed signals inside the device but it is clear that they toggle randomly very fast (in the 500MHz to 1GHz range) and that there is little autocorrelation in the time domain or correlation between adjacent bits. If the output of a generalized n-bit ring oscillator is sampled at a rate lower than the natural oscillating frequency, it will produce random n-bit numbers. While indeed random, the quality of these numbers is not necessarily acceptable for cryptographic use and they will need further processing to be useful for these types of applications.

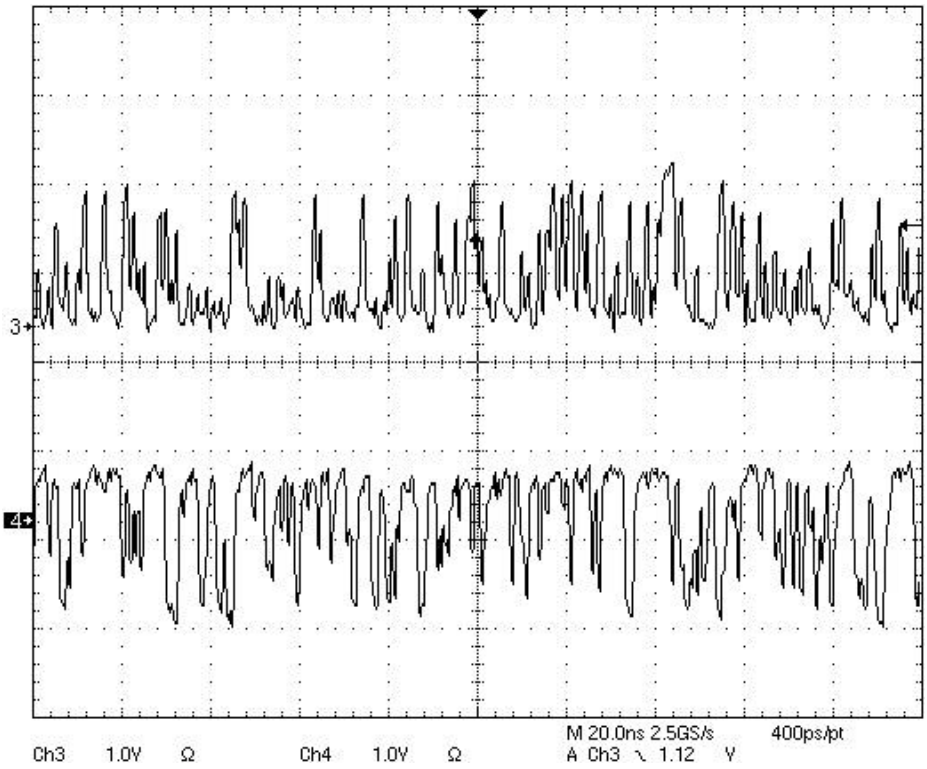

Fig. 4. Oscilloscope screen shot capture of two adjacent bits of a 32-bit generalized ring oscillator running in a Xilinx XC2VP7-6 FPGA. The output pins bandwidth severely limits the very high frequency oscillations that occur in the ring oscillator but indicate the chaotic nature of the process and the lack of correlation between data bits

2.2 Linear Hybrid Cellular Automata

Linear Hybrid Cellular Automata (LHCA) are a generalization of Linear Feedback Shift Registers (LFSR). With a complexity similar to that of an LFSR, an LHCA will produce a pseudo-random n-bit word every clock instead of just one bit per clock, as an LFSR will do. Parallel implementations of LFSRs exist that will generate n pseudo-random bits every clock period but they are large and slow. An LHCA can achieve the same result and the circuit will be much smaller and faster as LHCAs are scalable while LFSRs, and especially their parallel versions, are not.

Good introductions to LHCAs can be found in the published literature [1], we will simply state here some of their basic properties. Both LFSRs and LHCAs are linear feedback state machines (only XOR gates are used to determine the next state). At the same time, since the future state of each bit is a function only of itself and its two neighbors, an LHCA can also be seen as a 1-D cellular automaton [3]. The

automaton is hybrid since not all cells have the same behaviour, as the d_i coefficients can be either 0 or 1. The leftmost and rightmost bits have only one neighbor because there are no maximum length circular LHCAs.

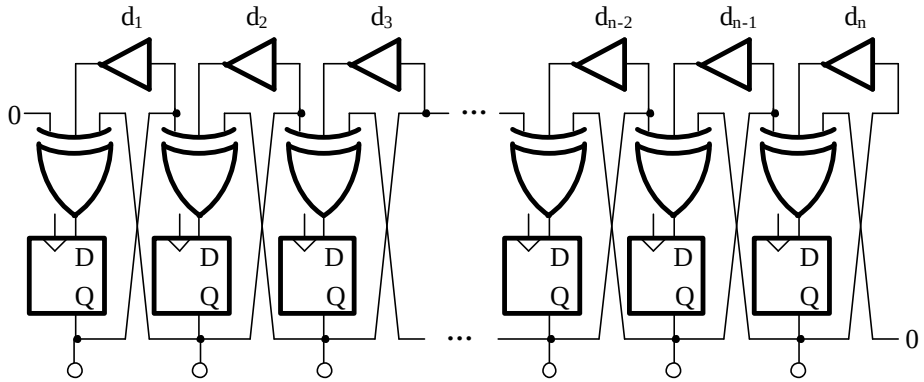

Fig. 5. An LHCA is a linear feedback state machine with the structure of a 1-D cellular automaton

LHCAs also share the same fundamental property with LFSRs – if the characteristic polynomial of the state transfer matrix is primitive they will both generate maximum length pseudo-random sequences. There is a one to one correspondence between maximum length LFSRs and maximum length LHCAs and for each primitive characteristic polynomial there is a certain set of coefficients that must be used. The d_i coefficients for all the LHCAs of order 2 to 8 as well as one example for sizes from 9 to 64 are given in the Appendix in Table 2.

2.3 Random Number Generation using Generalized Ring Oscillators and LHCAs

A generalized ring oscillator will produce random numbers but the probability distribution is not uniform and if the sampling rate is too high there will be significant correlation between samples. The output of an LHCA has very uniform probability distribution but it is not random and consecutive samples are strongly correlated. A combination of these two blocks that uses an LHCA to scramble the output of a generalized ring oscillator can produce very good quality random numbers at a very high data rate.

The proposed random number generator is shown in Fig. 6 and has a number of features that make it ideal for use in cryptographic applications implemented in an FPGA. It is scalable and uses only two flip-flops and two 4-input look-up tables for every output bit. By using floorplanning and directed routing constraints the design will maintain the same high speed and consistent operation independent of its size.

Since no external components are used as a source of randomness it is inherently more secure than other hardware implementations that exploit external sources of noise for this purpose as it eliminates one avenue of attack that would try to manipulate the random number generator to compromise a cryptographic application.

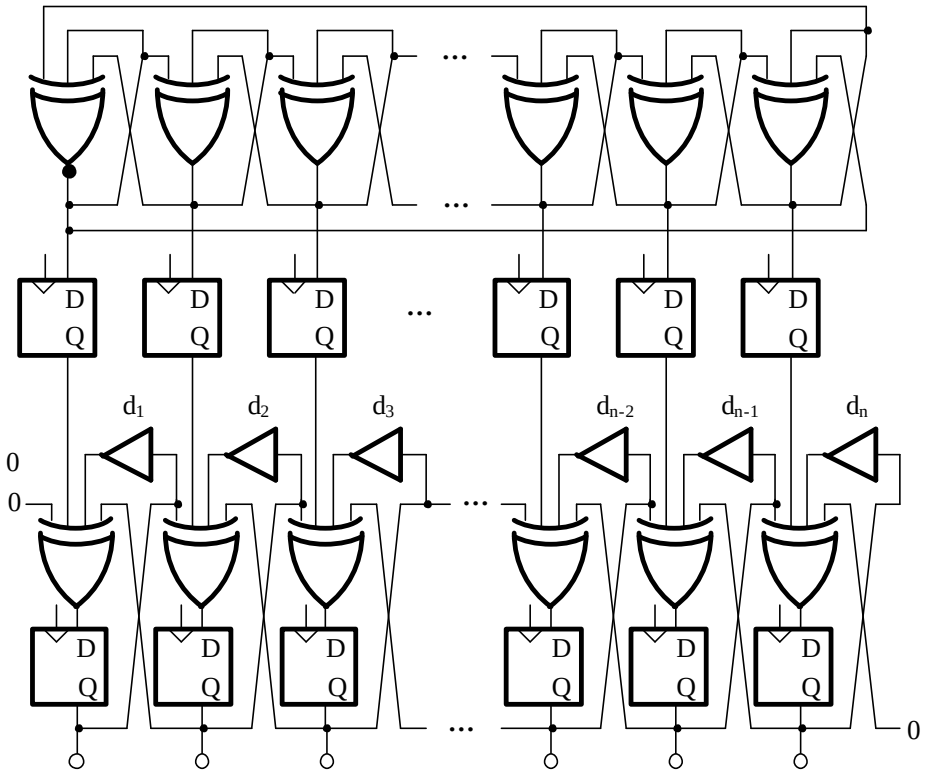

Fig. 6. A random number generator based on a generalized ring oscillator as a source of randomness and an LHCA used as a scrambler

Good quality random numbers at very high data rates can be produced, much faster than other implementations based on different random number generation mechanisms. For example, a 32-bit version running at 500MHz will generate random numbers at a rate of 16Gbps and since the design is scalable, even higher rates are possible if a larger ring oscillator and LHCA are used.

3 Testing the Randomness of the Generated Numbers

To test the quality of the random numbers produced by the proposed design the following setup has been used. A 32-bit version of the circuit shown in Fig. 6 was implemented in an XC2VP7-6 Virtex2Pro Xilinx FPGA, [4]. [5]. The generalized ring oscillator was sampled at two different frequencies, 500MHz and 125MHz, and passed through a 32-bit LHCA scrambler. The following sets of 11MBytes of data were collected and tested with the DieHard [2] random number test suite:

LHCA – the output of the LHCA with a zero input signal applied instead of the ring oscillator output (this is in fact a pseudo-random number sequence)

RING500 – the ring oscillator sampled at 500MHz

RND500 – the ring oscillator sampled at 500MHz and scrambled by the LHCA

RND125 – the ring oscillator sampled at 125MHz and scrambled by the LHCA

The DieHard test suite consists of 15 different randomness tests and Table 1 shows the result of these tests when applied to the four 11MB data sets.

Table 1. DieHard Random Number Test Suite Results.

Data Set	DieHard Tests Failed	DieHard Tests Passed
LHCA (11MB)	13	2
RING500 (11MB)	4	11
RND500 (11MB)	3	12
RND125 (11MB)	0	15

As expected, the LHCA data set fails the DieHard test completely since it is not random at all. The RING500 passes only some of the tests as the output probability distribution is not uniform and there is some sample correlation due to the very high data-sampling rate. The RND500 test, which is virtually a combination of the previous two cases generates much better results and is probably acceptable for most applications. Reducing the data-sampling rate from 500MHz to 125MHz produces a data set RND125 that passes all the DieHard tests and is probably suitable for the most demanding cryptographic applications.

4 Conclusions

This paper introduces a new random number generator design based on a generalization of ring oscillators that uses only digital logic gates and no external analog components. While the numbers generated by the proposed circuit are random and could be used in some applications, they do not have uniform distribution and are not good enough for cryptographic use. The use of a Linear Hybrid Cellular Automaton as a scrambler improves substantially the quality of the random numbers generated. The resulting design is able to generate good quality random numbers at very high data

rates, is scalable and is ideal for FPGA implementations. The design has been tested in a Xilinx FPGA and the output successfully passes all 15 tests in the DieHard random number test suite.

References

1. K. Cattell, S. Zhang, X. Sun, M. Serra, J. C. Muzio and D. M. Miller: One-Dimensional Linear Hybrid Cellular Automata: Their Synthesis, Properties, and Applications in VLSI Testing. In: http://csr.csc.uvic.ca/~mserra/Publications/CApaper.pdf

2. Marsaglia, G. Diehard: A battery of tests for randomness, 1996 http://stat.fsu.edu/~geo/diehard.html
3. S. Wolfram: Random Sequence Generation by Cellular Automata in Advances in Applied Mathematics, 7:123-169, 1986
4. Xilinx, Inc.: Virtex-II Pro Platform FPGAs: Complete Data Sheet, DS083, December 10, 2003, http://direct.xilinx.com/bvdocs/publications/ds083.pdf
5. Xilinx, Inc.: Virtex-II Pro Platform FPGA User Guide, UG012 (v2.4), June 30, 2003, http://direct.xilinx.com/bvdocs/userguides/ug012.pdf

Appendix: Coefficients for Maximum Length LHCAs

Table 2. Diagonal Coefficients and Primitive Polynomials for all LHCAs of order 2 to 8 and one example for sizes 9 to 64

Order	LHCA diagonal elements	Characteristic polynomial (primitive)
2	0x1	0x7
3	0x1	0xD
3	0x3	0xB
4	0x5	0x13
4	0xB	0x19
5	0x01	0x37
5	0x03	0x2F
5	0x06	0x29
5	0x07	0x3B
5	0x0F	0x25
5	0x13	0x3D
6	0x01	0x73
6	0x06	0x43
6	0x15	0x6D
6	0x16	0x61
6	0x1D	0x5B

6	0x25	0x67
7	0x04	0xC1
7	0x09	0xB9
7	0x0B	0xEF
7	0x0E	0xF7
7	0x12	0xA7
7	0x15	0xE5
7	0x17	0x91
7	0x21	0xBF
7	0x26	0xFD
7	0x2B	0x9D
7	0x2E	0x89
7	0x37	0xD3
7	0x3D	0xD5
7	0x45	0xF1
7	0x47	0x8F
7	0x4D	0x83
7	0x5B	0xCB
7	0x6F	0xAB
8	0x06	0x11D
8	0x0F	0x169
8	0x2A	0x18D
8	0x2D	0x165
8	0x36	0x14D
8	0x39	0x171
8	0x45	0x1A9
8	0x4B	0x163
8	0x5B	0x1CF
8	0x5D	0x1C3
8	0x5F	0x12B
8	0x77	0x12D
8	0x93	0x15F
8	0xAB	0x1F5
8	0xCB	0x1E7
8	0xEF	0x187
9	0x1F9	0x3E9
10	0x3D7	0x557
11	0x7F9	0xC89
12	0xFC1	0x18EF
13	0x1FF1	0x2B23
14	0x3FE9	0x6447
15	0x7FFB	0xABBB
16	0xFFE5	0x1E923
17	0x1FFEF	0x22075
18	0x3FFF7	0x7E5E9

19	0x7FFFB	0xA1A1B
20	0xFFFF9	0x10ABAB
21	0x1FFFCD	0x29D6C7
22	0x3FFFEF	0x79F2F1
23	0x7FFFF3	0xC27979
24	0xFFFFF9	0x11B0B0B
25	0x1FFFFDB	0x3C6E6E7
26	0x3FFFFF9	0x5200707
27	0x7FFFFE3	0x8331919
28	0xFFFFFFB	0x19DD1919
29	0x1FFFFF8D	0x32725151
30	0x3FFFFFC1	0x7CE17373
31	0x7FFFFFE5	0x84898585
32	0xFFFFFFBB	0x106B30E0F
33	0x1FFFFFFCF	0x3F376F2F1
34	0x3FFFFFFEB	0x55877585D
35	0x7FFFFFF3	0xCE5EEE5E9
36	0xFFFFFFED	0x11C4DDC4D5
37	0x1FFFFFFD9	0x2986BB86AF
38	0x3FFFFFFF9	0x5720772077
39	0x7FFFFFFFB	0xAB10BB10BB
40	0xFFFFFFFBD	0x10DEBEFEBE7
41	0x1FFFFFFFEF	0x22255755575
42	0x3FFFFFFF9F	0x50020272025
43	0x7FFFFFFFFB	0xA10011B011B
44	0xFFFFFFFF73	0x1FD806C206C1
45	0x1FFFFFFFFF9	0x3D4400790079
46	0x3FFFFFFFF6B	0x60ECFD71FD71
47	0x7FFFFFFFF9B	0x83FF116D116D
48	0xFFFFFFFFFFB	0x18589000D000D
49	0x1FFFFFFFFFAB	0x284DB04590459
50	0x3FFFFFFFFFE7	0x5072700250025
51	0x7FFFFFFFFFE7	0xCFBE200150015
52	0xFFFFFFFFFE9F	0x1F98AB0F230F23
53	0x1FFFFFFFFFEF	0x20070200250025
54	0x3FFFFFFFFFF9	0x57202000570057
55	0x7FFFFFFFFFDB	0xCD161000CB00CB
56	0xFFFFFFFFFF31	0x189013301BB01BB
57	0x1FFFFFFFFFFA9	0x322C8CF03250325
58	0x3FFFFFFFFFEF7	0x543B39B056B056B
59	0x7FFFFFFFFFF97	0x88C0D0C088D088D
60	0xFFFFFFFFFFFB7	0x115D9191115D115D
61	0x1FFFFFFFFFFFD9	0x29AF3D3D29AF29AF
62	0x3FFFFFFFFFFFC9	0x42B34A4A42B342B3
63	0x7FFFFFFFFFFFE5	0x8489858584898489

63	0x7FFFFFFFFFFFFE5	0x8489858584898489
64	0xFFFFFFFFFFFFFFB	0x18588858585888589