FPGA Implementation of Neighborhood-of-Four Cellular Automata Random Number Generators

Barry Shackleford, Motoo Tanaka, Richard J. Carter, and Greg Snider

Hewlett-Packard Laboratories
1501 Page Mill Rd.
Palo Alto, CA 94304 U.S.A.
{barry_shackleford, motoo_tanaka, dick_carter, greg_snider}@hp.com

ABSTRACT

Random number generators (RNGs) based upon neighborhood-of-four cellular automata (CA) with asymmetrical, non-local connections are explored. A number of RNGs that pass Marsaglia's rigorous DIEHARD suite of random number tests have been discovered. A neighborhood size of four allows a single CA cell to be implemented with a four-input lookup table and a one-bit register which are common building blocks in popular field programmable gate arrays (FPGAs). The investigated networks all had periodic (wrap around) boundary conditions with either 1-d, 2-d, or 3-d interconnection topologies. Trial designs of 64-bit networks using a Xilinx XCV1000-6 FPGA predict a maximum clock rate of 214 MHz to 230 MHz depending upon interconnection topology.

Keywords

random number generator, cellular automata, FPGA

1. INTRODUCTION

Since the beginning of the computer age, high-quality random numbers have played an important and expanding role in areas such as Monte Carlo simulations [1, 2], computer-based gaming, VLSI chip testing, and probabilistic computing methods like simulated annealing, genetic algorithms, and neural networks. As computers have become more powerful and simulations more ambitious, the demands on random number generators (RNGs) have likewise increased [3]. However, there is no single test to determine the quality of an RNG—a battery of different tests is required. As our RNG touchstone, we will use George Marsaglia's widely acknowledged battery of stringent [4] random number tests: DIEHARD [5-7].

Digital hardware designers have long relied on feedback shift register methods [8, 9] for the generation of random numbers. In 1986, Wolfram [10] suggested that cellular automata (CA), which rely on bit-level computation and local interconnection, could be used for efficient hardware implementation of random number generators due to their simplicity and regularity of design.

With the advent of VLSI design, it became advantageous to incorporate a part of the chip testing system on the chip itself. Initially, linear feedback shift registers were used to implement the random pattern generator portion of the built-in self-test. Hortensius *et al*. [11] showed that nonhomogeneous CA composed of two linear functions could generate superior random numbers to the linear feedback shift register and to the homogenous nonlinear CA proposed by Wolfram [10].

Large-scale field programmable gate arrays (FPGAs) now make it possible to build entire systems on a single FPGA chip [12]. The building block of many FPGA architectures is the electrically reprogrammable lookup table (LUT) and its associated single-bit register for data storage. The typical LUT fanin is four, which yields a 16-row truth table, whereas a CA with a radius of one (i.e., one neighboring cell left and right of center) has a neighborhood of three, which requires an eight-row truth table that wastes half of the LUT's capacity. A two-dimensional CA network, or alternatively, a one-dimensional CA network with a radius of two, requires a 32-row truth table, requiring two LUTs per CA bit position.

Our purpose was to explore the implementation of random number generators using four-input (i.e., neighborhood of four) cellular automata with asymmetrical, non-local connections. Using a neighborhood of four allows a single CA cell to be implemented efficiently in a typical four-input LUT, thus allowing high-quality, inexpensive RNGs to be implemented in FPGAs as system components. In conducting this investigation, we have discovered a number of 64-bit CA-based RNGs that pass the DIEHARD suite of random number tests.

In the next section, we will describe some of the previous work in CA-based RNGs. Then in Section 3, we will provide an overview of CA conventions and describe our extensions. Section 4 will describe our experimental procedure and Section 5 will show some detailed results.

2. PREVIOUS WORK

In 1986, Wolfram [10] described random sequence generation by simple 1-d cellular automata with a neighborhood size of three. The work focused on the properties of CA30 (so named by the decimal value of its eight-row truth table). Seven statistical tests were performed and the CA30 was shown to be superior to the linear feedback shift register in the generation of random sequences. Wolfram pointed out that very efficient hardware implementations should be possible for the CA30. Referring to a single site, he pointed out that for some critical applications, time sampling (allowing one or more additional state updates between reads) would improve the statistical quality of the sequence.

Hortensius *et al*. [11], in 1989, described the use of the CA30 as a random number generator in a VLSI implementation of a two-dimensional Ising computer. Later that year, Hortensius *et al*. [13],

FPGA'02, February 24-26, 2002, Monterey, California, USA.

described the application of CA-based RNGs to built-in self-test of VLSI chips. By using a combination of CA90 (XOR of left and right neighbors) and CA150 (XOR of left, right, and center cells) at various cell sites they were able to generate results superior to the CA30. They indicated that both time spacing and site spacing would improve the statistical quality of the random numbers. Later, in 1991 and 1994 respectively, Taslides *et al.* [14] and Nandi *et al.* [15] performed further work with the CA90/CA150 hybrid CA with regard to generation of random number streams for on-chip test.

In 1994, Chowdhury *et al*. [16] described a class of two-dimensional, neighborhood of five (i.e., $N=5$), cellular automata composed of linear cells (XOR function). The target application was random number generation for VLSI built-in self-test. They found that while the quality of random numbers was better for the 2-d CA, the fault coverage was not always better.

Since the number of hybrid (i.e., different truth table functions possible at each cell position) CA is very large (256^n for a neighborhood of three with uniform interconnections where n is the number of cells), Sipper and Tomassini [17], in 1996, described a process of evolving the CA truth tables *in situ* (a co-evolution process termed cellular programming). The lengths of the CA were 50 and 150 cells with the quality of the random numbers being comparable to existing CA-based RNGs.

In 1999, Tomassini *et al*. [18] introduced the use of Marsaglia's highly regarded DIEHARD random number test suite to provide a standardized means of comparing random number quality among CA-based RNGs. Using the DIEHARD suite, they showed that their previous work [17] now failed five of the 18 listed tests. However, with a time spacing of two (i.e., reading data on cycles i, $i+3$, $i+6$, …) the one-dimensional RNG passed all of the tests.

In 2000, Tomassini *et al*. [19] again employed cellular programming to evolve an 8×8 two-dimensional CA-based RNG with a neighborhood of five where the chromosome specification for a cell indicated either XOR or XNOR functionality and to which cells in the neighborhood it was connected. Although the DIEHARD p-values were not published, they indicated that 14 tests were passed without resorting to time spacing or site spacing. They also described the random generation of an 8×8 array using the same six-bit rules as the evolved RNG with the restriction that at least five of the six bits be 1s. They indicated that an additional four DIEHARD tests were passed. However, neither the p-values nor the exact configuration were published.

3. CELLULAR AUTOMATA

Cellular automata can be thought of as dynamical systems, discrete in both time and space [20]. The traditional view is that they are implemented as an array of cells with homogenous functionality, constrained to a regular lattice of some dimensionality. Most often, these lattices have been either one-dimensional strings/rings or two-dimensional planes/toroids. Strings and planes result from fixed boundary conditions and rings and toroids result from periodic boundary conditions. Each cell has a state that is updated as a function of local neighbor connections on each time step. For two-state cellular automata with a neighborhood size of N there are 2^{2^N} possible implementations.

Cellular automata apparently came about as a result of conversations between Stanislaw Ulam [21] and John von Neumann in the late 1940s. Von Neumann used the concept of cellular automata in his work on self-reproducing automata [22]. In the 1980s, Stephen Wolfram wrote extensively on cellular automata [23]. His work continues to the present [24].

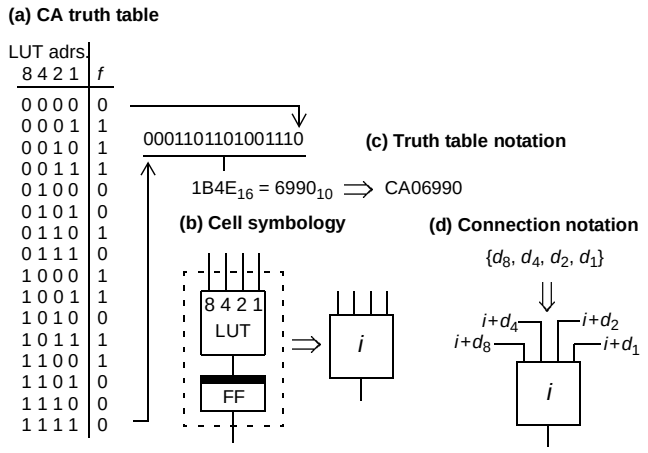

Figure 1: (a) The functionality of a cell derives directly from LUT truth table. (b) Cell symbology: each cell contains a 4-input lookup table which defines the cell functionality and a 1-bit register which holds the cell's state. (c) The notation derives directly from decimal value of the function's 16-bit truth table. Leading 0s are included to prevent confusion with 3-neighborhood rules. (d) Connections to cell *i* are expressed as a set of displacements from *i*'s ordinal value.

The function of a CA cell can be described by a truth table as shown in Fig. 1(a). As shown in Fig. 1(b), there is an implicit one-bit register associated with each cell. Since we are using four-input lookup tables (implying a neighborhood size N of four) as the basis for our experiments, there are 16 possible conditions to which a cell can respond. The number of unique responses can be viewed as a 16-bit binary number which yields 2^{16} unique machines. Wolfram introduced a notation embodying this number which uniquely names a cell with a name describing its functionality as shown in Fig.1(c). However, since we are extending the range of connectivity beyond the local neighborhood, this functional name is no longer adequate to uniquely describe a CA. As shown in Fig. 1(d), we have introduced a relative displacement notation to indicate, relative to a given cell i, how far away the connecting cells are.

We will use Fig. 2 to illustrate the effect of periodic boundary conditions on networks of one, two and three dimensions. All of the networks in our study contained 64 cells which allowed for the creation of 64-cell linear, 8×8 planer and $4 \times 4 \times 4$ cubic networks.

Fig. 2(a) shows a one-dimensional ring network with a relative connectivity of {-1, 0, 1, 2} from the perspective of cell 0. Due to the periodic boundary conditions, cells 0 and 63 are adjacent, so a unit displacement in the negative direction from cell 0 lands on cell 63.

Fig. 2(b) illustrates a two-dimensional network with periodic boundary conditions which transform a plane into the surface of a toroid where each row forms a ring and each column forms a ring. We use the compass directions *n*, *s*, *e*, *w* as shorthand to indicate relative unit displacement along columns and rows relative to a center cell *c*. Multiple steps in a given direction are indicated by a number preceding the direction. The example illustrates the connectivity implied by {2*n*2*w*, *c*, *ne*, *s*} with respect to center cell 7.

Cell 7 is both at the end of a row and a column which makes it an interesting test case. Considering the 2*n*2*w* displacement from cell 7: going north two steps takes us to cell 55, then going west two steps from 55 takes us to 53. The *c* in the relative displacement list implies a connection to itself, hence the 7. The *ne* displacement ends up at cell 56 by first going north and wrapping around to cell 63, then going east and wrapping around to cell 56. Directly south from cell 7 is cell 15.

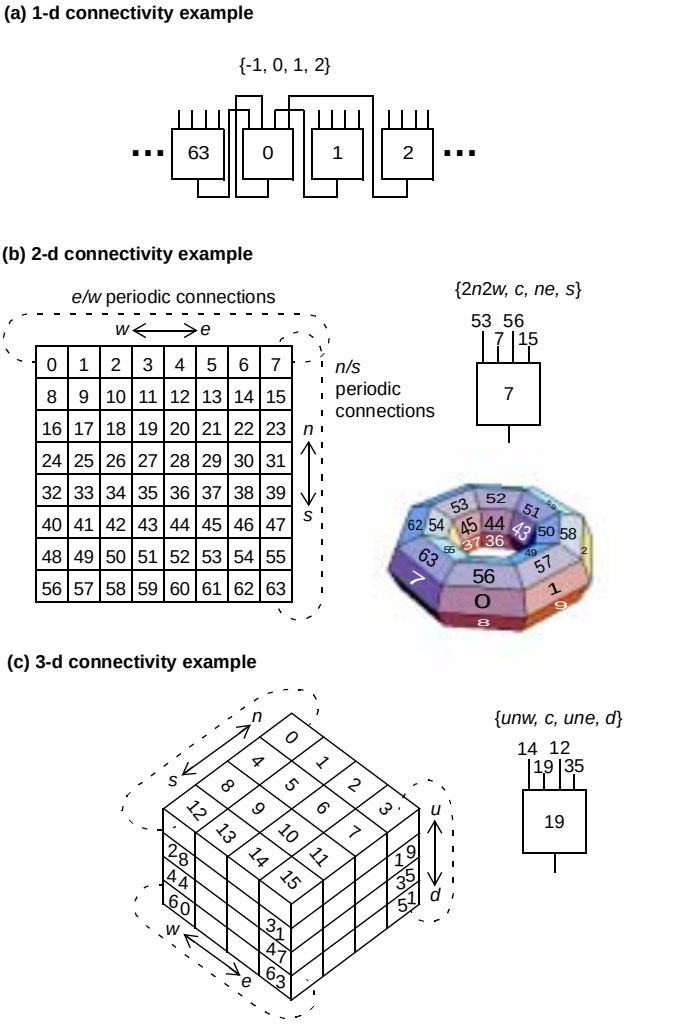

Figure 2: (a) Periodic connections on the 1-d network form a ring. In the example, the interconnection network is defined by {-1, 0, 1, 2} and is illustrated for cell 0. (b) Periodic connections on the 2-d networks form the surface of a toroid. For the 2-d network, relative displacements are indicated by *n, s, e, w,* with *c* being the target cell. (c) Periodic connections for the 3-d network. Unit displacements in the *z* axis are noted by *u* and *d* as shown in the example.

By applying periodic boundary conditions to a cubic lattice as shown in Fig. 2(c) we create a network that is composed of 12 intersecting toroids. This follows from the fact that each plane becomes a toroid as in the previous example and since there are now three axes, each with four planes, the total number of toroids is 12. Relative up and down displacements along the *z* axis are indicated by *u* and *d* as shown in the example.

4. METHODOLOGY

For a given connection topology, rather than test all possible CA candidates with DIEHARD, we first performed an entropy screening test. The best candidates from the entropy test were then subjected to the DIEHARD suite. In the next two subsections we will describe the search for high-entropy CA and then briefly outline the test results generated by the DIEHARD test suite.

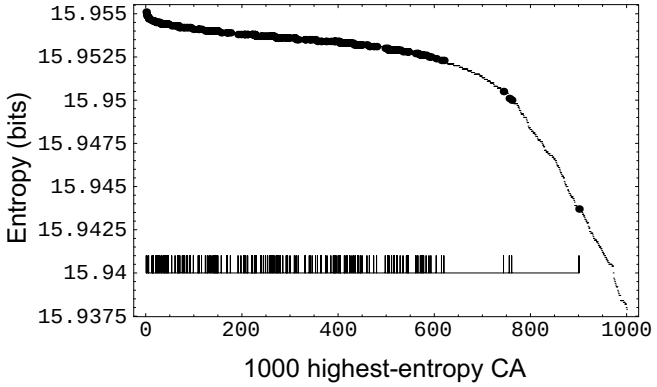

Figure 3: Entropies of the 1000 highest-entropy CA with connectivity {-7, 0, 11, 17}. The 167 CA that passed the DIEHARD suite are plotted with a '•'. A frequency distribution is shown in the lower portion of the plot.

4.1 Finding CA candidates to test as RNGs

Since we are dealing with a neighborhood of four (due to the fanin of four imposed by the lookup table size of popular FPGAs), there are only 2^{16} possible CA for a given connection topology. These can be exhaustively searched for high-entropy candidates. We prescreened candidate CA by calculating their 16-bit entropy S_{16} with the following summation [20]:

$$S_{16} = -\sum_{i=0}^{2^{16}-1} p_i \log_2 p_i$$

where the 16-bit binary value of i represents the sub-sequence being tested and p_i is the observed probability of its occurrence in the output of the RNG. For bit-entropy calculations, $\log_2$ of 0 is defined to be 0. S_{16} has a maximum value of 16 (indicating maximum entropy) which is obtained when all 16-bit sequences from all 0s to all 1s occur with equal probability.

For a given interconnection topology, the highest entropy candidate CA were searched for by starting with CA00000, supplying it with an initial state consisting of a single 1 in cell 0 with all other cells set to 0. The CA is initalized to a high-entropy state by advancing 80 cycles (i.e., its length plus 16 cycles to fill the bit-serial registers used as part of the entropy calculation).

For a given CA state, 16-bit sub-sequences are checked from the perspective of all 64 bits in the CA. The CA is viewed as a ring and the first test looks at bits 0–15, the second test looks at bits 1–16, and the 64th test looks at bits 63–14. Then the 16-bit shift registers connected to each of the 64 bits of the CA are checked. This yields 128 tests per CA state.

The CA state is then advanced and checked for a total of 2^{13} cycles before proceeding to the next CA truth table. For these parameters, the expected value (with a good RNG) for any of the 2^{16} subsequence counts is 16. The search is speeded considerably by rejecting any candidate whose sub-sequence count exceeds 4x the expected value.

After all CA truth tables have been tested, the 1000 highest entropy CA are tested with the DIEHARD random number test suite. Figure 3 shows the distribution of 16-bit entropy for the best 1000 CA with connectivity {-7, 0, 11, 17}. Of these best 1000, 167 passed the entire DIEHARD battery of tests (plotted with heavy dots). The list is printed in the Appendix.

4.2 RNG testing

The DIEHARD [5-7] random number test suite provides an extensive set of tests to evaluate a random number stream and can be downloaded from Marsaglia's site on the Internet [7].

Data for each DIEHARD run were collected in the following manner: The CA under test was first initalized to a high-entropy state by clocking for 64 cycles from an initial state of a single 1 in cell 0. Then, data collection commenced on the 65th cycle and continued for 3 million cycles, collecting a total of 3 million 32-bit words composed of the even-numbered bits and 3 million 32-bit words composed of the odd-numbered bits. Both sets of data were required to pass all DIEHARD tests for a CA to be considered as having "passed DIEHARD."

It is beyond the scope of this paper to explain in detail each of the DIEHARD tests. There is a summary report issued with each test that explains its statistical nature [7]. A number of the tests are also explained in [4, 5, 6]. To provide some sense of what the numbers mean that are plotted in the histogram (Figure 4), we will quote Marsaglia directly from the message that appears first on every DIEHARD test report [7]:

> NOTE: Most of the tests in DIEHARD return a *p*-value, which should be uniform on [0,1) if the input file contains truly independent random bits. Those *p*-values are obtained by $p = F(X)$, where F is the assumed distribution of the sample random variable X—often normal. But that assumed F is just an asymptotic approximation, for which the fit will be worst in the tails. Thus you should not be surprised with occasional *p*-values near 0 or 1, such as 0.0012 or 0.9983. When a bit stream really FAILS BIG, you will get *p*'s of 0 or 1 to six or more places. By all means, do not, as a Statistician might, think that a $p < 0.025$ or $p > 0.975$ means that the RNG has "failed the test at the 0.05 level." Such *p*'s happen among the hundreds that DIEHARD produces, even with good RNGs. So keep in mind that "*p* happens."

We will briefly list the DIEHARD tests below, numbering them as they appear in digest histogram (Figure 4) and in the order that they are printed in the DIEHARD test report. Some of the tests generate multiple *p*-values and then provide a summary Kolmogorov-Smirnof (KS) test value. In those cases, we plot the *p*-values as tic marks above the baseline and the KS value is plotted as a tic mark below the baseline. Otherwise, all test results are plotted as tic marks above the baseline.

1. Birthday Spacings Test: Nine *p*-values and a KS test on those values are reported.

2. Overlapping 5-Permutation Test (OPERM5): Two *p*-values are generated by the test.

3. Binary Rank Test: One *p*-value is reported for tests on 31x31 matrices and one *p*-value is reported for tests on 32x32 matrices.

4. Binary Rank Test: Twenty-five *p*-values for tests on 6x8 matrices and a KS *p*-value are reported.

5. Bitstream Test: Twenty *p*-values are reported.

6a. Overlapping Pairs Sparse Occupancy Test (OPSO): Twenty-three *p*-values are reported.

6b. Overlapping Quadruples Sparse Occupancy Test (OQSO): Twenty-eight *p*-values are reported.

6c. DNA Test: Thirty-one *p*-values are reported.

7. Count the 1s Test: The test is performed on a *stream* of bytes. Two *p*-values are reported.

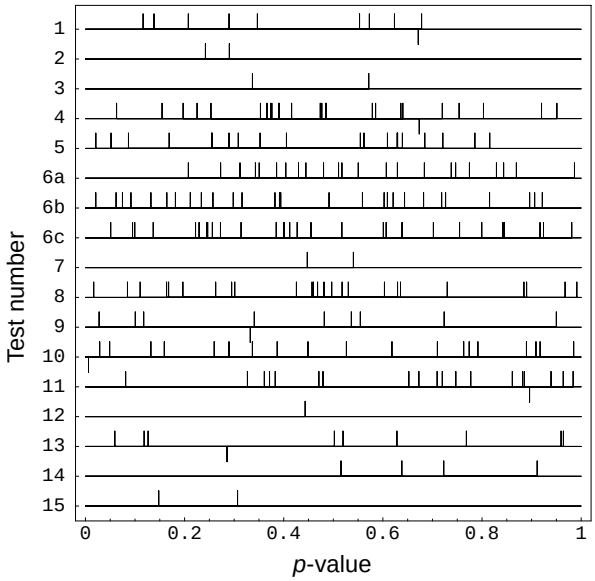

Figure 4: Sample histogram digest of the test results from the DIEHARD battery of random number tests. The tests are numbered 1-15 in the order that they appear in the DIEHARD test report and reported *p*-values are indicated as tic marks on each test's baseline. This example shows all tests passing.

8. Count the 1s Test: The test is performed for *specific* bytes. Twenty-five *p*-values are reported.

9. Parking Lot Test: Ten *p*-values and a KS test on those values are reported.

10. Minimum Distance Test: Twenty *p*-values and a KS test on those values are reported.

11. 3-D Spheres Test: Twenty *p*-values are along with a KS test on those values are reported.

12. Squeeze Test: One *p*-value is reported.

13. Overlapping Sums Test: Ten *p*-values generated from KS tests along with a KS test on those values are reported.

14. Runs Test: Ten *p*-values are generated for each test in a pair of runs-up/runs-down tests and four KS test results are reported.

15. Craps Test: The results of 200,000 simulated games of craps are analyzed, generating a single *p*-value each for the number of wins and for the throws/game.

5. RESULTS

5.1 Comparison of CA-based RNGs

Figure 5 shows space-time diagrams and DIEHARD test result digests for the well-known $N = 3$ CA30 and four of the $N = 4$ CA that were discovered in this investigation. Each will be discussed below:

CA30 represents the best of the one-dimensional, locally connected, neighborhood-of-three CA-based RNGs. As with all of the other CA shown, it was initalized with a single 1 in cell 0 (top line, far left). The space-time diagram shows 64 bits horizontally (0 to 63) and 200 time steps, down the page. CA30 fails DIEHARD tests 2, 4, 6b, 6c, 8, 12, 14, and 15.

CA38490, a 1-d ring network, is interesting in that it is considerably slower than the other neighborhood-of-four CA in advancing to a high-entropy state. It passes all of the DIEHARD suite and has a relative connectivity of {-3, 0, 4, 9}.

CA50745 is a 1-d ring network with a connectivity of {-7, 0, 11,

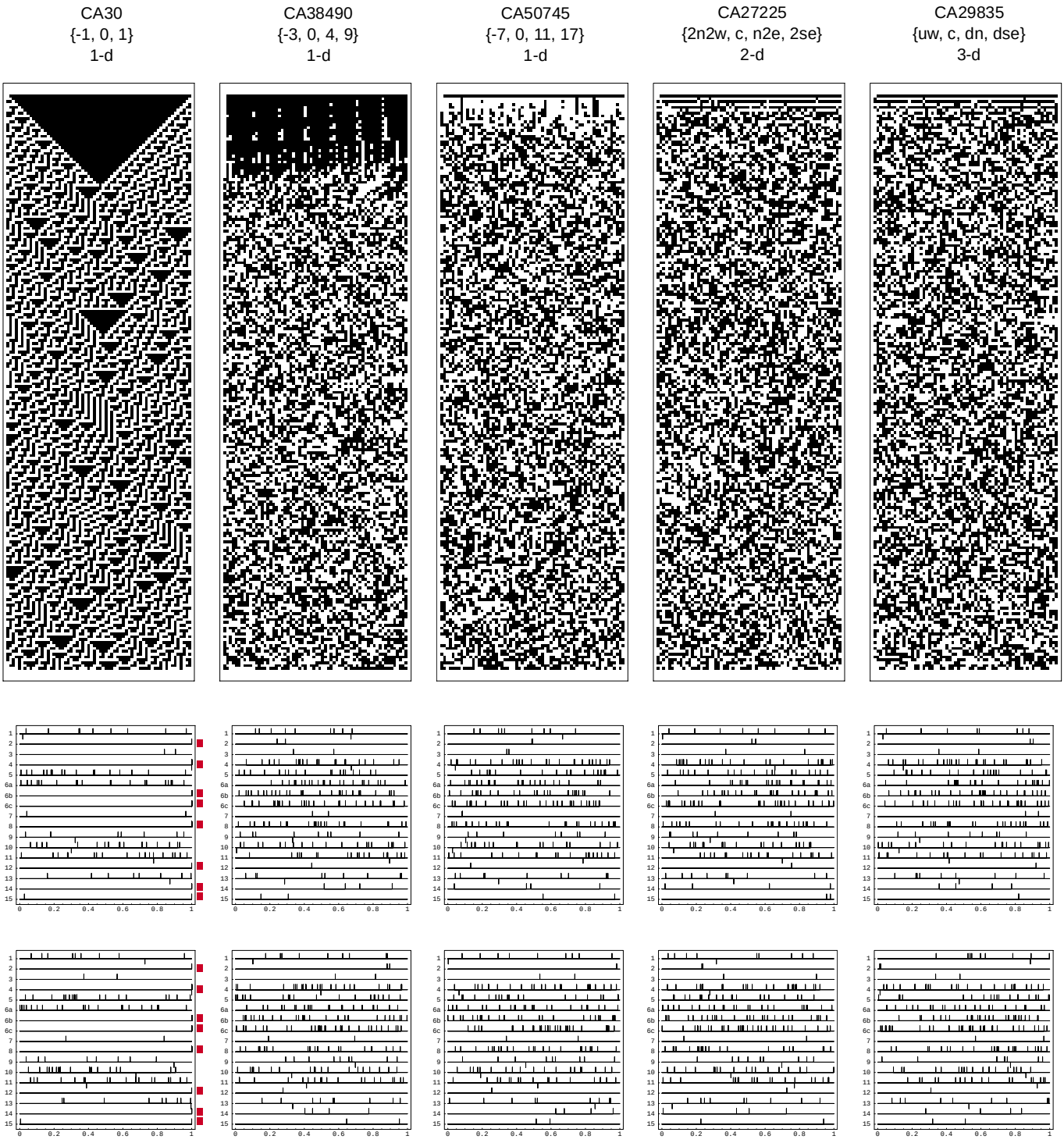

Figure 5 The top row shows space-time diagrams of five random number generators—the state of the CA (bits 0–63) is displayed horizontally (white = 1, black = 0) and subsequent states are displayed down from the top (200 time steps shown). All are shown starting with the same initial state of a single 1 in cell 0 (the left-most bit). The second and third rows are digests (as in Figure 4) of DIEHARD test results for the even and odd bits respectively of the above random number generators. CA30, shown for contrast, fails eight of the DIEHARD tests (indicated by solid square to the right of the failing test's histogram).

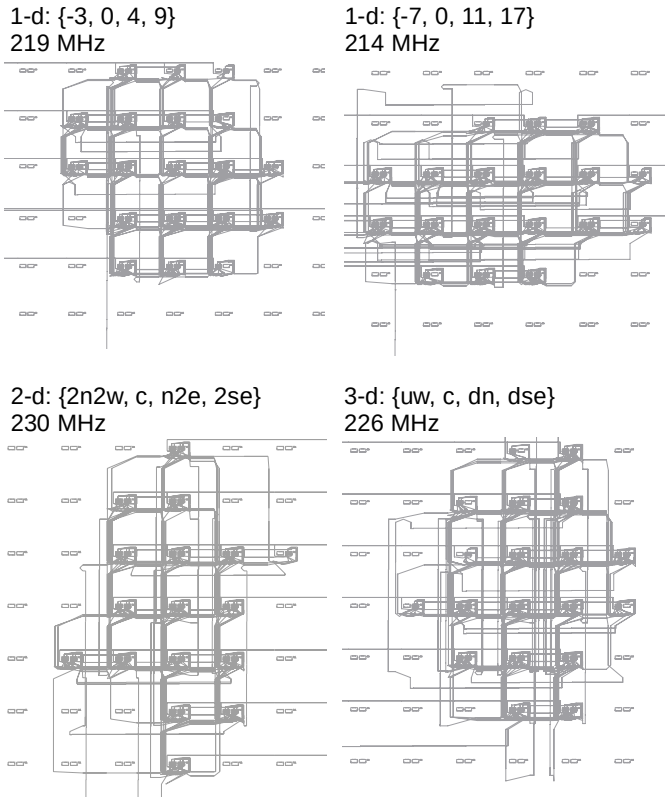

Figure 6: FPGA physical layouts and maximum predicted operating frequency for the neighborhood-of-four cellular automata random number generators shown in Figure 5 (based upon Xilinx XCV1000-6).

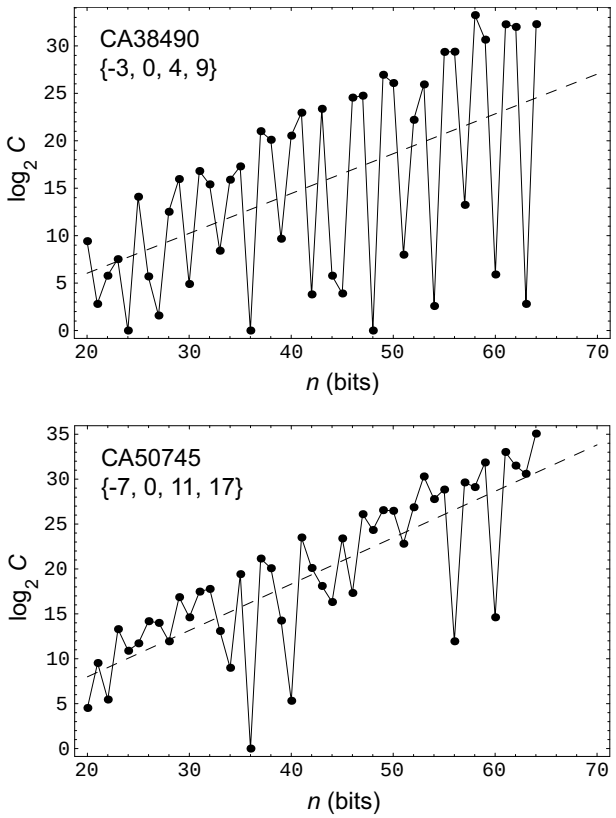

Figure 7: Plots of cycle length *C* vs. CA length *n* for CA38490 and CA50745 from Figure 5. The initial condition for all trials consisted of a CA state with a single 1. The resulting cycle lengths are not necessarily maximal.

17}. It was chosen as a contrast to the more locally connected network of CA38490. It passes all of the DIEHARD suite.

CA27225 is a 2-d toroidal network (Figure 2b) with a relative connectivity of {2n2w, c, n2e, 2se}. It passes all of the DIEHARD suite and has a very rapid advancement to a high-entropy state.

CA29835 is a 3-d network (Figure 2c) with a relative connectivity of {uw, c, dn, dse}. The CA passes all of the DIEHARD suite and has a very rapid advancement to a high-entropy state.

5.2 Physical design experiments

Given the context of a conventional FPGA (Xilinx XCV1000-6) and its physical design system (Xilinx Foundation Series ISI 3.3i), we conducted four implementation experiments to determine if there were any significant relationship between a CA's network topology and its maximum predicted operating clock frequency.

As shown in Figure 6, the place and route system responded differently to each of the 4-bit CA networks. The maximum predicted clock rate ranged from 214 MHz for the 1-d {-7, 0, 11, 17} topology to 230 MHz for the 2-d {2n2w, c, n2e, 2se} topology.

5.3 Cycle length

Any finite state machine that is clocked through a sequence of states must eventually encounter a previous state. The number of states in this repeating sequence we term the cycle length *C* which is an important measure for any CA being considered as a random number generator.

The CA-based RNGs presented here were found to have multiple cycles. The particular cycle that is finally entered is a function of the initial state. In addition, the non-linear function implemented within each CA cell allows multiple unique states to advance to the same successor state. For every instance where *k* unique states advance to the same single successor state, there must be $k-1$ states that have no predecessor. As a consequence, the cycles tend to have many merging non-cyclic state sequences that feed into them, which we term *feeder* states. For example, an exhaustive analysis of all the state sequences of the length $n = 31$ CA50745 {-7,0,11,17} is shown in Table 1 in terms of cycles, feeder states and no-predecessor *proto-states*.

Table 1: Properties of the *n* = 31 CA50745 {-7, 0, 11, 17}.

Cycle length C	Feeder states	Proto-states
182466	2147090938	756834341
1705	109368	39339
279	98859	35371
31	0	0
1	1	1

Additionally, we performed non-exhaustive cycle length experiments for the two 1-d CA shown in Figure 5. For each experiment, the CA length *n* was varied from 20 to 64 bits and the eventual cycle length *C* was determined for an initial state consisting of a single 1. The results are shown in Figure 7 where $\log_2 C$ is plotted against *n*.

The location of the single 1 in the initial state is immaterial due to

rotational symmetry that results from the periodic boundary conditions. It should be noted that these cycle lengths are not necessarily the maximum cycle lengths for a given *n*. Also, the number of feeder states leading from the initial state to a state in the final ring cycle can be a substantial fraction of *C* or in cases where *C* is small, many times *C*.

6. CONCLUSION

We have described a class of cellular automata-based 64-bit random number generators with a neighborhood of four which allows them to be efficiently implemented with four-input lookup tables found in widely used field programmable gate arrays. The random number generators pass all of the tests in Marsaglia's DIEHARD random number test suite.

Networks with periodic boundary conditions of one, two, and three dimensions were explored. Physical design experiments for these networks predicted a maximum clock frequency of from 214 MHz to 230 MHz according to the network. All of the CA-based random number generators required one four-input LUT and a single one-bit register per bit.

REFERENCES

[1] N. Metropolis and S. Ulam, "The Monte Carlo method," *Journal of American Statistical Association*, vol. 44, pp. 335–341, 1949.

[2] M. A. Kalos and P. A. Whitlock, *Monte Carlo Methods, Volume I: Basics*, Wiley Interscience, New York, 1986.

[3] S. K. Park and K. W. Miller, "Random number generators: good ones are hard to find," *Communications of the ACM*, vol. 31, no. 10, pp. 1192–1201, October 1988.

[4] D. E. Knuth, *The Art of Computer Programming: Volume 2, Seminumerical Algorithms*, 3rd ed., Addison-Wesley, ch. 3, 1998.

[5] G. Marsaglia, "Remarks on choosing and implementing random number generators," *Communications of the ACM*, vol. 36, no. 7, pp. 105–110, July 1993.

[6] G. Marsaglia, "A current view of random numbers," *Computer Science and Statistics: The Interface*, L. Billard, ed., Elsevier Science Publishers B. V., pp. 3–10, 1985.

[7] G. Marsaglia, *DIEHARD*, http://stat.fsu.edu/~geo/diehard.html, 1996.

[8] VR. C. Tausworthe, "Random numbers generated by linear recurrence modulo two," *Mathematical Computing*, vol. 19, pp. 201–209, 1965.

[9] S. W. Golomb, *Shift Register Sequences*, Holden-Day, San Francisco, 1967.

[10] S. Wolfram, "Random sequence generation by cellular automata," *Advances in Applied Mathematics*, vol. 7, pp. 123–169, June 1986. (Also available in S. Wolfram, *Cellular Automata and Complexity*, Addison-Wesley, 1994.)

[11] P. D. Hortensius, R. D. McLeod, and H. C. Card, "Parallel number generation for VLSI systems using cellular automata," *IEEE Transactions on Computers*, vol. 38, no. 10, pp. 1466–1473, October 1989.

[12] B. Shackleford, G. Snider, R. J. Carter, E. Okushi, M. Yasuda, K. Seo, and H. Yasuura, "A high-performance, pipelined, FPGA-based genetic algorithm machine," *Genetic Programming and Evolvable Machines*, vol. 2, no. 1, pp. 33–60, March 2001.

[13] P. D. Hortensius, H. C. Card, R. D. McLeod, and W. Pries, "Importance sampling for Ising Computers using one-dimensional cellular automata," *IEEE Transactions on Computers*, vol. 38, no. 6, pp. 769–774, June 1989.

[14] Ph. Tsalides, T. A. York, and A. Thanailakis, "Pseudorandom number generators for VLSI systems based on linear cellular automata," *IEE Proceedings-E*, vol. 138, no. 4, pp. 241–249, July 1991.

[15] S. Nandi, B. Vamsi, S. Chakraborty, and P. P. Chaudhuri, "Cellular automata as a BIST structure for testing CMOS circuits," *IEE Proceedings-Computer Digital Technology*, vol. 141, no. 1, pp. 41–47, January 1994.

[16] D. R. Chowdhury, I. Sengupta, and P. P. Chaudhuri, "A class of two-dimensional cellular automata and their applications in random pattern testing," *Journal of Electronic Testing*, vol. 5, pp. 67–82, 1994.

[17] M. Sipper and M. Tomassini, "Generating parallel random number generators by cellular programming," *International Journal of Modern Physics C*, vol. 7, no. 2, pp. 181–190, 1996.

[18] M. Tomassini, M. Sipper, M. Zolla, and M. Perrenoud, "Generating high-quality random numbers in parallel by cellular automata," *Future Generation Computer Systems*, vol. 16, pp. 291–305, 1999.

[19] M. Tomassini, M. Sipper, and M. Perrenoud, "On the generation of high-quality random numbers by two-dimensional cellular automata," *IEEE Transactions on Computers*, vol. 49, pp. 1146–1151, October 2000.

[20] S. Wolfram, "Statistical mechanics of cellular automata," *Reviews of Modern Physics*, vol. 55, pp. 601–644, July 1983. (Also available in S. Wolfram, *Cellular Automata and Complexity*, Addison-Wesley, 1994.)

[21] S. Ulam, "Random processes and transformations," *Proceedings of the International Congress of Mathematics (1950)*, vol. 2, pp. 264–275, 1952.

[22] J. von Neumann, *Theory of Self-Reproducing Automata*, ed. Authur Burks, University of Illinois Press, 1966.

[23] S. Wolfram (ed.), *Theory and Applications of Cellular Automata*, World Scientific, 1986.

[24] S. Wolfram, *A New kind of Science*, Wolfram Media, January 2002.

APPENDIX

Listed below are the 167 CA with connectivity {-7, 0, 11, 17} that passed our initial entropy screening and the DIEHARD battery of tests:

25946, 34425, 50745, 38553, 42075, 22117, 13929, 27029, 5865, 50739, 26211, 15045, 27795, 58395, 4845, 49977, 50742, 27027, 38454, 11985, 18870, 51510, 22121, 13005, 38501, 38595, 18615, 38565, 25398, 15513, 27798, 27705, 22933, 26307, 25449, 36210, 14742, 13257, 37692, 26262, 7905, 19125, 26775, 27225, 26313, 15507, 40086, 26006, 38556, 54315, 11220, 30855, 40545, 51555, 39267, 39363, 14025, 27237, 37788, 38547, 50019, 14019, 29835, 49974, 34170, 27702, 37833, 26937, 39318, 11475, 27801, 14649, 25500, 15561, 26969, 49470, 9690, 37737, 25961, 37782, 14652, 39990, 26166, 37731, 22953, 55335, 38457, 13974, 25494, 45645, 19890, 38249, 14745, 21865, 40035, 37485, 7395, 26965, 39273, 26979, 46155, 50793, 14793, 15459, 42329, 27033, 40131, 14646, 26966, 17340, 33660, 14739, 13515, 42345, 26025, 22695, 15411, 21165, 14787, 5355, 14697, 50838, 24990, 31875, 13923, 37689, 23715, 50787, 14748, 27081, 23189, 38601, 13980, 37785, 27843, 41310, 9180, 37995, 39015, 33405, 3315, 38451, 10200, 10455, 53295, 39574, 25443, 27747, 39258, 13161, 16830, 26931, 22181, 38489, 49725, 25545, 26934, 38549, 50235, 35190, 27753, 40041, 3060, 41820, 39510, 50490, 42840.