Parallel Random Number Generators for Sequences Uniformly Distributed Over Any Range of Integers

Richard Kuehnel
Headquarters
US European Command
Unit 30400
APO, AE 09131

James Theiler
Space and Remote Sensing Sciences Group
Los Alamos National Laboratory
Los Alamos, NM 87545

Yuke Wang
Department of Computer Science
Erik Jonsson School of Engineering and Computer Science
Box 830688, MS EC 31
University of Texas at Dallas
Richardson, TX 75083-0688

ABSTRACT

A VLSI design methodology is proposed for the efficient generation of multiple pseudo-random number sequences based on a simplification of Cauwenberghs' counter-propagation technique. We demonstrate that the counter-propagation of two sequences can be replaced by one propagating and one non-propagating sequence, requiring as few as half the number of flipflops, while still allowing new circuits to be added to the system without additional calculations – there is no need to keep track of random starting values, tap combinations, or time shifts. Moreover we extend our method from multiple bit sequences to multiple random number sequences that are uniformly distributed over any range of integers. In particular we address the more general problem of generating sequences over the range $[0, K]$, where $K+1$ is any desired integer, including a power of two or a prime number. To this end we demonstrate that the simple concatenation of random bits to form random bytes is a special case of a more general concept whereby random integers distributed over prime number ranges are concatenated to form random integers distributed over any range. We find that the proposed design compares favorably with design strategies based on cellular automata, both in terms of statistical properties and implementation efficiencies.

Index: random number generator, PRNG, uniform sequence, VLSI circuit, stochastic computing, stochastic processing, cellular automata

1 INTRODUCTION

Inserting small random number generators into hardware has a wide variety of potential applications, such as the computational elements of artificial neural networks [1][2]. The methodology presented in this paper was originally developed for stochastic computing, a technique that provides very low computation hardware area, fault tolerance, and efficient hardware implementations for high clock rates. Consider a generalized digital-to-stochastic converter consisting of a register containing the input X_n, a pseudo-random number generator producing R_n, and a comparator, as described in [3][4][5] and shown in Fig. 1.

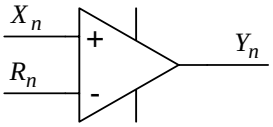

Fig. 1. In a digital-to-stochastic converter the digital input is compared to a uniformly distributed sequence to generate a Bernoulli sequence.

If X_n is restricted to $0 \le X_n \le K$ and R_n is uniformly distributed over the same range, then Y_n is a Bernoulli random variable with $P\{Y_n = 1\} = X_n / K$. The Bernoulli sequence $\{Y_n\}$ thus represents the value of X_n / K as a stochastic code, where the constant K is an important designer-specified parameter. In IIR filters, for example, the range of the random number distribution $[0, K]$ controls the width of the passband [6].

Stochastic computing has been successfully applied to artificial neural networks by using large numbers of these relatively simple computing elements [1]. Multiple circuits, each with an independent source of random numbers, are used to create a massively parallel system. This often requires separate pseudo-random number generators (PRNGs) for each circuit, so that a large amount of silicon area is consumed by random number production. Moreover, to ensure that the numbers are statistically uncorrelated, each PRNG must be designed using a different algorithm or a different starting value. This adds complexity to the design and increases the size of hardware implementations.

Where the numbers being produced are uniformly distributed over a range $[0, K]$, where $K+1$ is a power of two, Hortensius et al. have shown that a cellular automata-based approach has far better randomness characteristics than using the bits of a single linear feedback shift register (LFSR)

in parallel [7]. According to this method, numbers are represented as collections of single-bit cells. A cell's value is determined by its previous value and the previous values of nearby cells.

Alspector et al. propose an alternative, LFSR-based approach [1]. Also called Tausworthe generators, linear feedback shift registers have been shown to have useful theoretical properties [8] and combined LFSR generators have additional advantages [9]. In particular they have very long periods. Alspector et al. create multiple random bit sequences from a single LFSR by tapping into the shift register. Three taps are added together modulo-2 to produce identical sequences shifted in time by an amount that can be calculated, with different tap combinations resulting in different time shifts. Although the resulting bit patterns are the same, the time shifts between them are designed to be long enough so that they are effectively uncorrelated. In accordance with their methodology, the designer solves a set of equations to determine the time shift produced by each possible combination of 3 taps. A set of tap combinations is then selected to ensure that the time shifts between bit sequences are sufficient.

Alspector's method enables the system to use only one random bit generator, but numerous taps must be routed throughout the system, resulting in circuits that are difficult to interconnect efficiently in VLSI. Moreover, the requirement to compute and select the time shift for each individual circuit is administratively tedious - the designer must choose a new time shift whenever a new circuit is added to the system. Saarinen et al. review other methods of generating time-shifted sequences, but note that an optimum method of dealing with the complexity of the problem has not yet been developed [10].

Cauwenberghs developed a simplified scheme using counter-propagating linear feedback shift registers. The generated bit sequences are obtained from the XOR of the parallel outputs of two counter-propagating shift registers driven by different primitive polynomials. This technique drastically reduces the implementation complexity and routing requirements of Alspector's approach and has been used successfully in analog neural networks [2].

In this paper we propose a methodology based on a simplification of Cauwenberghs' technique. We demonstrate that the counter-propagation of two sequences can be replaced by one propagating and one non-propagating sequence, requiring as few as half the number of flipflops, depending on fanout constraints, while still allowing new circuits to be added to the system without additional calculations – there is no need to keep track of random starting values, tap combinations, or time shifts. Moreover we extend our method from multiple bit sequences to multiple random number sequences that are uniformly distributed over any range of integers. In particular we address the more general problem of generating sequences over the range $[0, K]$, where $K+1$ is any desired

integer, including a power of two or a prime number. To this end we demonstrate that the simple concatenation of random bits to form random bytes is a special case of a more general concept whereby random integers distributed over prime number ranges are concatenated to form random integers distributed over any range.

The paper is organized as follows. In Section 2 we describe the mathematical basis for our methodology. In Section 3 the generation of parallel bit sequences is presented. Section 4 extends these techniques to the generation of parallel sequences uniformly distributed over any integer range. Section 5 computes the number of logic gates needed by the system as a function of the desired distribution range as compared to other methods. Experimental results are presented in Section 6. Section 7 concludes this work.

2 MATHEMATICAL PRELIMINARIES

In this section we prove two theorems that will be used in later sections to propose a memory-efficient alternative to counter-propagation. These theorems, which are an extension of the concepts introduced by Fillmore and Marx [11], apply to the generation of random numbers over a range $[0, K]$, where $K+1$ is a prime number, including random bit generation where $K+1=2$. In this section we also introduce a third theorem that will be used to extend the design methodology to random numbers uniformly distributed over any range of integers.

DEFINITION 1. A sequence v, composed of elements $v_0, v_1, v_2, ...$ is called a *linear recursive sequence* over the field F if there exist $c_1, c_2, ..., c_m$ in F such that

$$v_{n+m} = \sum_{i=1}^{m} c_i v_{n+m-i} \qquad (2.1)$$

for every integer $n \geq 0$.

DEFINITION 2. The *shift operator* σ is defined for the sequence v by $\sigma v = w$, where $w_n = v_{n+1}$ for $n \geq 0$.

We will use σ^k to denote k applications of the shift operator so that if $\sigma^k v = w$ then $w_n = v_{n+k}$. We will use $\sigma^{-k} v = w$ as a notation for $\sigma^k w = v$.

DEFINITION 3. A *characteristic polynomial* in x over F is defined as

$$f(x) \equiv x^m - \sum_{i=1}^{m} c_i x^{m-i} \tag{2.2}$$

In this context m is called the *degree* of $f(x)$. The sequence v is said to *satisfy* the recursion associated with $f(x)$.

PROPOSITION 1. A linear recursive sequence v satisfies $f(x)$ if and only if $f(\sigma)v = 0$.

Proof: If the linear recursive sequence v satisfies the characteristic polynomial $f(x)$ corresponding to

$$v_{n+m} = \sum_{i=1}^{m} c_i v_{n+m-i} \tag{2.3}$$

then

$$v_{n+m} - \sum_{i=1}^{m} c_i v_{n+m-i} = 0 \tag{2.4}$$

and thus

$$\begin{aligned} 0 &= \sigma^m v - \sum_{i=1}^{m} c_i \sigma^{m-i} v \\ &= \left(\sigma^m - \sum_{i=1}^{m} c_i \sigma^{m-i} \right) v \\ &= f(\sigma) v \end{aligned} \tag{2.5}$$

The reverse of these steps is also true, so we conclude that the linear recursive sequence v satisfies the characteristic polynomial $f(x)$ if and only if $f(\sigma)v = 0$.

PROPOSITION 2. If $u = \sigma^m v + \sigma^k w$, where v and w satisfy the polynomials $f(x)$ and $g(x)$, respectively, then u satisfies $f(x)g(x)$.

Proof: Let $u = \sigma^m v + \sigma^k w$, then since operations are always commutative in a field we can write

$$
\begin{aligned}
f(\sigma)g(\sigma)u &= f(\sigma)g(\sigma)\sigma^m v + f(\sigma)g(\sigma)\sigma^k w \\
&= g(\sigma)\sigma^m f(\sigma)v + f(\sigma)\sigma^k g(\sigma)w \qquad (2.6) \\
&= 0
\end{aligned}
$$

Thus $f(\sigma)g(\sigma)u = 0$ and u satisfies $f(x)g(x)$ by Proposition 1.

DEFINITION 4. A polynomial $f(x)$ is called *irreducible* if its only divisor is itself and the unit polynomial $g(x) = 1$.

DEFINITION 5. An irreducible polynomial $f(x)$ of degree m over the finite field F of q elements is called *primitive* if the period of the sequence it generates is equal to $q^m - 1$. The resulting sequence is known as an *m-sequence*.

PROPOSITION 3. Suppose v and w are non-zero linear recursive sequences with periods M and N, respectively. If the polynomials of the recursions of these sequences are relatively prime, then $u \equiv v + w$ has a period equal to the least common multiple of M and N. (A proof of this proposition is found in [11].)

LEMMA 1: Let v and w be non-zero linear recursive sequences and let the polynomials of the recursions of these sequences be different and primitive. Then $u \equiv v + w$ has a period equal to the least common multiple of $q^m - 1$ and $q^n - 1$, where m and n are the degrees of the polynomials of v and w, respectively.

Proof: This is a direct result of Definition 5 and Proposition 3.

THEOREM 1. For the finite field F of q elements let v and w be linear recursive sequences satisfying the different primitive polynomials $f(x)$ and $g(x)$ of degrees m and n. Then, for any time shift k, the sequence $u \equiv v + \sigma^k w$ is periodic with a period equal to the least common multiple of $q^m - 1$ and $q^n - 1$.

Proof: If w is a non-zero linear recursive sequence that satisfies a polynomial that is primitive, then the sequence $\sigma^k w$ satisfies the same polynomial. Therefore, as a direct result of Lemma 1, $u \equiv v + \sigma^k w$ has a period equal to the least common multiple of $q^m - 1$ and $q^n - 1$ for any integer k.

THEOREM 2. For the finite field F of q elements let v and w be non-zero linear recursive sequences satisfying the different primitive polynomials $f(x)$ and $g(x)$ of degrees m_1 and m_2 where $m_1 > 1$ and $m_2 > 1$. Let M be the period of v and N be the period of w and let M and N be relatively prime. Consider the two sequences $u(1) \equiv v + \sigma^{k_1} w$ and $u(2) \equiv v + \sigma^{k_2} w$, where $1 \leq | k_1 - k_2 | < M < N$. Then $u(1) = \sigma^p u(2)$, where $|p| \geq M$.

Proof: According to Proposition 2, the sequences $u(1)$ and $u(2)$ both satisfy $f(x)g(x)$, which is of degree $m = m_1 + m_2$. Let

$$f(x)g(x) = x^m - \sum_{i=1}^{m} c_i x^{m-i} \tag{2.7}$$

Then

$$u(1)_{n+m} = \sum_{i=1}^{m} c_i u(1)_{n+m-i} \tag{2.8}$$

and

$$u(2)_{n+m} = \sum_{i=1}^{m} c_i u(2)_{n+m-i} \tag{2.9}$$

It is clear from the summations that all of the subsequent elements of a sequence can be determined by any m consecutive elements the sequence. If there exists even one value of n such that $u(2)_{n+m-i} = u(1)_{n+m-i+p}$ for all i, $1 \leq i \leq m$ then

$$u(2)_{n+m} = \sum_{i=1}^{m} c_i u(1)_{n+m-i+p} \tag{2.10}$$

i.e. $u(2)$ is the sequence $u(1)$ shifted by p. Both $u(1)$ and $u(2)$ are of length $\left(q^{m_1} - 1\right)\left(q^{m_2} - 1\right)$. If none of the m-length subsequences in $u(1)$ and $u(2)$ are the same then there exist at least

$2\left(q^{m_1}-1\right)\left(q^{m_2}-1\right)$ unique *m*-length subsequences. But only q^m-1 unique *m*-length subsequences exist. Since $2\left(q^{m_1}-1\right)\left(q^{m_2}-1\right) > q^m-1$ then at least one of the *m*-length subsequences of $u(1)$ is identical to a subsequence in $u(2)$ so that a value of p exists such that $u(1)=\sigma^p u(2)$. This means $v+\sigma^{k_1}w=\sigma^p v+\sigma^{p+k_2}w$. From this we infer that $v=\sigma^p v$ and $\sigma^{k_1}w=\sigma^{p+k_2}w$ and therefore conclude

$$p \bmod M = 0 \qquad (2.11)$$

(We use $a \bmod b$ to denote "a modulo b", i.e. the remainder resulting from the integer division of a by b.) It also means $k_1 \bmod N = (p+k_2) \bmod N$. Thus since $k_1 \neq k_2$ then

$$p \neq 0 \qquad (2.12)$$

From (2.11) and (2.12) we conclude that $|p| \geq M$. Therefore $u(1)$ and $u(2)$ will be the same sequence shifted by an amount greater than or equal to M.

Another way to view these concepts is to consider $v+w$ to be a sequence with period MN. This represents a concatenation of N different subsequences of length M. Then all the different sequences of the form $u=v+\sigma^k w$ correspond to these different subsequences.

The next theorem will later be used to demonstrate that the concatenation of random bits to form numbers distributed over ranges equal to a power of two represents a special case of a more generalized approach in which the range can be any positive integer.

THEOREM 3. For every integer $K>1$ there exists a set of prime numbers $q_1,...,q_M$, unique except for order, where $q_1 q_2 ... q_M = K+1$, such that if r_i is uniformly distributed over the set of integers $\{0,...,q_i-1\}$ for all i, $1 \leq i \leq M$, then the sum

$$\rho = r_1 + r_2 q_1 + r_3 q_1 q_2 + ... + r_M q_1 q_2 ... q_{M-1} \qquad (2.13)$$

is uniformly distributed over the set of integers $\{0,...K\}$. (A proof of this theorem is found in [12].)

3 GENERATING SEQUENCES DISTRIBUTED OVER A PRIME RANGE

Using the methodology of the proof to Theorem 2 we can show that one of the propagating sequences of the counter-propagation method, as shown in Fig. 2, can be replaced by a non-propagating sequence with negligible effect on the guaranteed time shift between output sequences. This cuts in half the number of flipflops required.

When both sequences propagate, moving from one output sequence to another results in a positive time shift of one input sequence and a negative time shift of the other. Let $u(1) \equiv v + w$ and $u(2) \equiv \sigma^{-k} v + \sigma^{k} w$ represent two output sequences separated by k. The signs of the shift operator reflect that the input sequences are counter-propagating. Without loss of generality, let $0 < 2k < M < N$, where M and N are the periods of v and w, respectively. Since $\sigma^{k} u(2) = v + \sigma^{2k} w$ then by Theorem 2, $\sigma^{k} u(2) = \sigma^{p} u(1)$ where $|p| \geq M$. This means $u(2) = \sigma^{p-k} u(1)$ where $|p - k| \geq M - k$. Therefore $u(1)$ and $u(2)$ will be the same sequence shifted by an amount greater than or equal to $M - k$.

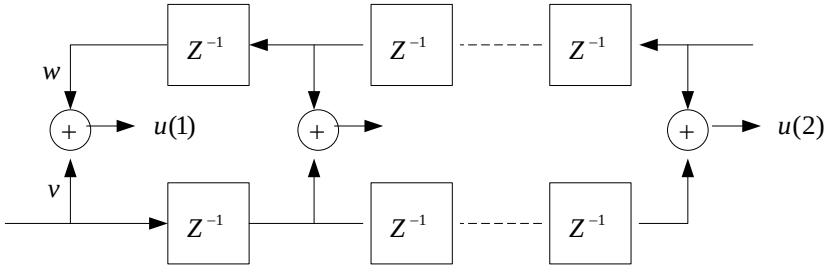

Fig. 2. The counter-propagation of two input sequences creates output sequences that are identical to each other but shifted substantially in time.

Our system for generating random sequences uniformly distributed over a prime-number range using only half the number of flipflops (excluding fanout considerations) is depicted in Fig. 3. This is the second of L circuits needing random sequences, as shown in Fig. 4. A non-propagating sequence R is generated with characteristic polynomial $f_R(x)$ and period M. A second, propagating sequence S is generated with characteristic polynomial $f_S(x)$ and period N. The polynomials are different and primitive. The designer assumes an upper bound $N_{\max}$ on the total number of random sequences that the system may need, taking into account any additional circuits that may be added in the future. Then the characteristic polynomials are selected so that $N > M > N_{\max}$. Good randomness properties dictate that, even for very large systems, M and N will generally be orders of magnitude larger than $N_{\max}$ and, as will be shown, the circuit area per random number generator increases at a rate of only $O(\log M) + O(\log N)$.

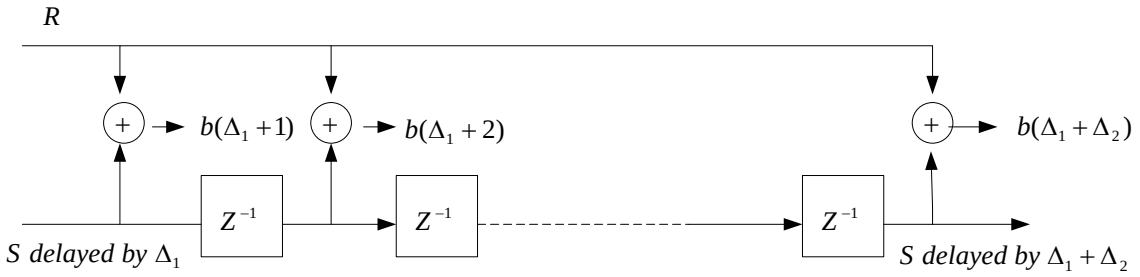

Fig. 3. Each client circuit generates as many random sequences as it needs from the propagating and non-propagating input sequences and passes the propagating input sequence to the next circuit.

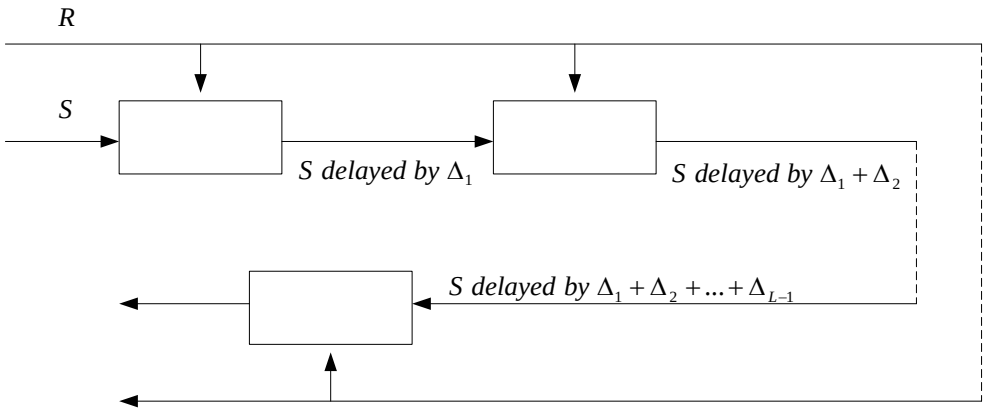

Fig. 4. Sequences generated by two LFSRs are used to drive a daisy chain of client circuits, each internally generating its own random sequences from the two inputs.

Each circuit delays the propagating sequence S by one clock cycle for every random sequence $\{b(m)_n\}$ that it needs, where m identifies the particular sequence. It generates these sequences by the modulo-q addition (an XOR gate for $q = 2$) of R_n and S_{n-m}, where the latter is delayed by one clock cycle from where it was used to generate the previous sequence.

The system is therefore characterized by two pseudo-random sequence generators that supply a daisy chain of client circuits. Circuits can be added to the system by inserting them anywhere in the daisy chain. Moreover, an individual circuit can be modified in a way that demands a greater number of random sequences without having to modify the system architecture.

4 GENERATING SEQUENCES OVER AN ARBITRARY RANGE

Linear feedback shift registers used to generate random bits are commonly known. A more general LFSR for any prime range is shown in Fig. 5.

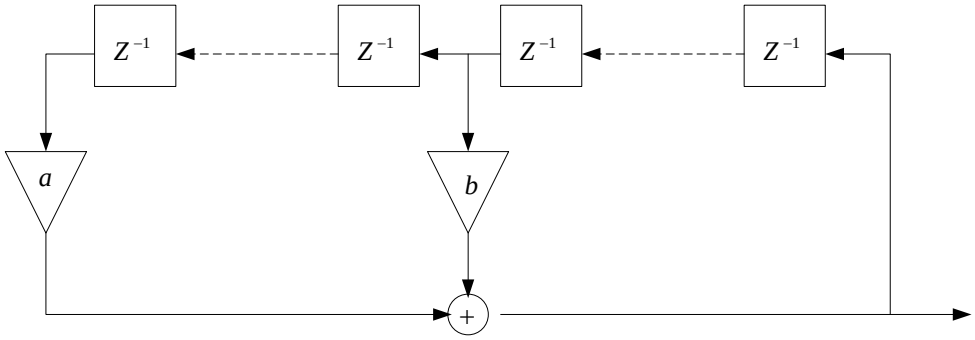

Fig. 5. A linear feedback shift register (LFSR) with two taps can be used to generate pseudo-random sequences uniformly distributed over a range *[0,q-1]*, where *q* is any prime number.

Each signal denotes a sufficient number of bits to represent the sequence's maximum value $q-1$. Both the output and the input to the shift register result from the mod-q addition

$$r = a\sigma^{-L}r \oplus b\sigma^{-\lambda}r \qquad (4.1)$$

where L is the length of the shift register and λ is a fixed tap such that $1 \le \lambda < L$. The coefficients a and b are determined by a primitive polynomial. More than two taps can also be used. As was the case for random bit generation, a list of appropriate polynomials is widely available and found, for example, in [13]and [14]. For $q = 2$ the tap coefficients are $a = b = 1$. Otherwise $1 \le a, b < q$ according to the selected polynomial. Moreover, a and b are not necessarily the same as the polynomial coefficients. A procedure for computing the tap coefficients from the characteristic polynomial when $q > 2$ is described in [12].

Relying on Theorem 3, we create a random sequence ρ that is uniformly distributed over $[0, K]$, where K is any integer greater than one, by factoring $K+1$ into its unique primes $K+1 = q_1 ... q_M$. We then generate M independent, random sequences r_i uniformly distributed over $0 \le r_i < q_i$, $1 \le i \le M$, and concatenate them by relation (2.13). Two LFSR circuits for each unique value of q_i are needed for the entire system, each having a different random sequence length. They generate the two input sequences shown in Fig. 4.

The input sequences are not used directly. Instead a new random sequence is generated by their modulo-q_i addition, where one input sequence is delayed by 1 clock cycle from where it was last used. The sequence r_i thus refers to an independent sequence uniformly distributed over $[0, q_i - 1]$ that is used at only one location in the circuit.

The desired sequence ρ is generated by (2.13). One can observe that multiplication by q_1 occurs in $M-1$ terms, whereas multiplication by q_M never occurs. Since a multiplication by a power of two can be efficiently implemented in VLSI, we choose to order the prime factors such that $q_1 \le q_2 \le ... \le q_M$. If $K+1$ is equal to a power of two, then $q_1 = ... = q_M = 2$ and equation (2.13) reduces to

$$\rho = \sum_{i=1}^{M} r_i (2)^{i-1} \qquad (4.2)$$

where each r_i is an independent random bit sequence. Note that if the periods of the bit sequences r_i are identical then ρ has this same period. The summation is achieved without logic by simply ordering the random bits from the least significant to the most significant. Thus the well-known technique of concatenating random bits to form random numbers is a special case of the more generalized method used here. The summation can also be performed without additional logic if $K+1 = \left(2^{M-1}\right) q_M$, where $q_M > 2$. In this case equation (4.2) remains unchanged. If, on the other hand, $K+1 = \left(2^{M-2}\right) q_{M-1} q_M$, where $2 < q_{M-1} \le q_M$, then

$$\rho = \sum_{i=1}^{M-1} r_i (2)^{i-1} + r_M (2)^{M-2} q_{M-1} \qquad (4.3)$$

The multiplication by q_{M-1}, fortunately a constant value, requires additional logic. Obviously if many of the prime number factors are not equal to 2, then significant additional circuitry may be required.

As an illustrative example, Fig. 6 shows a gaming system comprised of multiple circuits, each selecting playing cards at random from a deck of 52 cards. Every clock cycle each circuit selects a new card (pseudo-) independently of the cards it selected previously and independently of its neighboring circuits. Since $52 = (2)(2)(13)$, there are only two unique primes: 2 and 13.

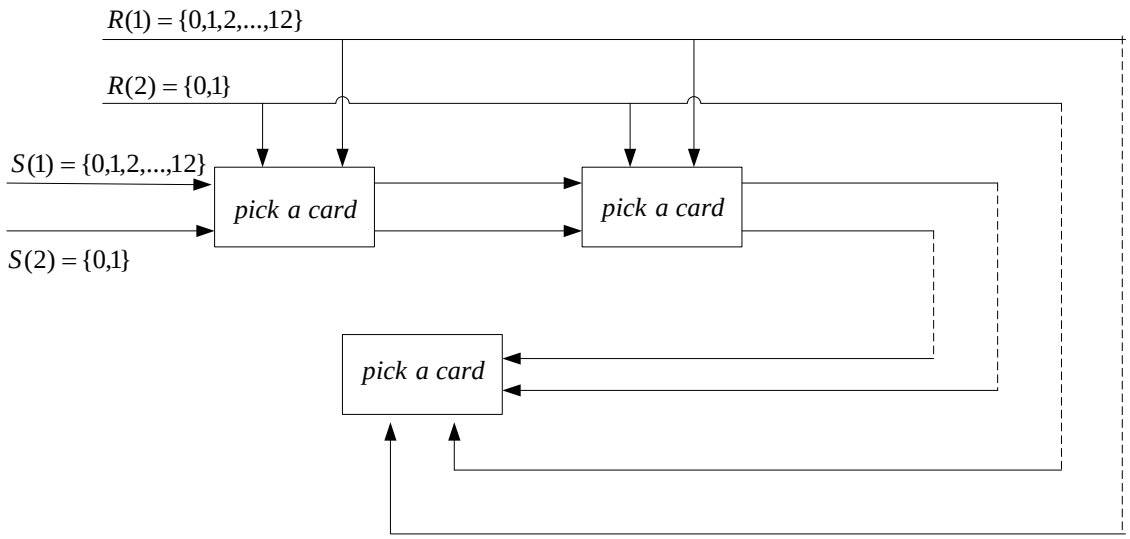

Fig. 6. A system of pseudo-independent circuits, each randomly selecting one of 52 cards, is constructed using only 4 LFSRs.

Thus only 4 LFSRs are required for the entire system, two that generate the random number sequences $R(1)$ and $S(1)$, each uniformly distributed over the range $[0,12]$, and two that generate the random bit sequences $R(2)$ and $S(2)$. Fig. 7 shows how an individual circuit creates the desired output sequence uniformly distributed over $[0,51]$.

From the input sequences it generates two shifted bit sequences and one shifted sequence distributed over the range $[0,12]$. These sequences are concatenated according to equation (2.13), where $q_1 = q_2 = 2$. Because only the largest prime $q_3 = 13$ is not equal to two, the concatenation for this case is trivial and requires no circuit logic. It merely assigns the random bits to the least significant bits of the result and assigns the bits of the sequence distributed over $[0,12]$ to the most significant bits of the result.

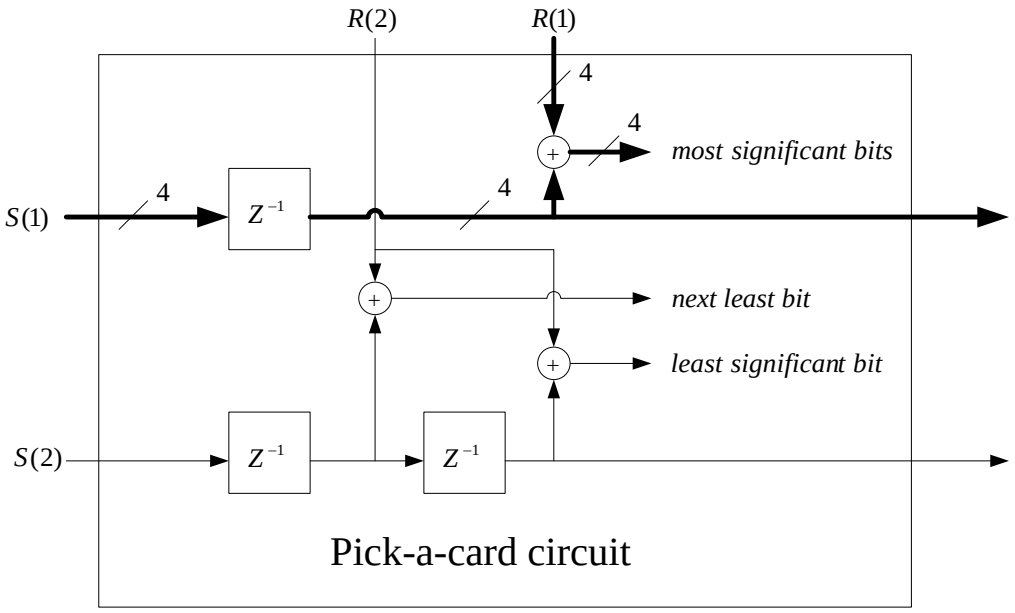

Fig. 7. A number uniformly distributed over the range *[0,51]* is constructed from the concatenation of a random number distributed over *[0,12]* and two random bits. The "next least bit" refers to the bit next to the least significant bit.

5 CIRCUIT SIZE

In stochastic computing applications the number of logic gates consumed by random number generation can easily exceed the number of gates used in arithmetic operations. It is important, therefore, to be able to quantify the size of random number generation at the circuit level.

The number of logic gates needed to create a random sequence varies depending on the range of the sequence. The least number of logic gates are required when as many of the prime factors as possible are equal to 2. To generate a sequence over $[0,255]$, for example, requires only eight XOR gates and eight flipflops. This compares to eight XOR gates and 16 flipflops for Cauwenberghs' technique and 24 XOR gates but zero flipflops for Alspector's. (While we include Alspector's technique here for size comparison, it has significant additional routing and implementation constraints, as noted in Section 1.) More generally, to create an independent pseudo-random sequence that is uniformly distributed over $[0, K]$, where $K+1 = q_1 ... q_M$, requires a modulo-q_i adder for each i, $1 \le i \le M$. The total number of flipflops needed is

$$\sum_{i=1}^{M} \lceil \log_2 (q_i) \rceil \qquad (5.1)$$

This compares to the same number of adders but twice the number of flipflops for Cauwenberghs' method [2], triple the number of adders but zero flipflops for Alspector's [1].

For sequences uniformly distributed over ranges that are not a power of two, additional logic elements for multiplication are required if $q_i > 2$, for any $i < M$. For example, if $q_{M-1} = 3$ and $q_M = 5$ then we need to compute the product $3r_M$ where $r_M = \{0,1,...,4\}$. Since the coefficient 3 is a fixed value, this multiplication would require only a lookup table with a 3-bit input to represent r_M and a 4-bit output to represent the product. These additional logic elements would also be needed to extend the Cauwenberghs and Alspector techniques to the generation of random integers over ranges that are not powers of two.

In addition to the circuit elements required for each independent random sequence, a pair of linear feedback shift registers is needed for each prime factor that is unique to the overall system. In contrast, Alspector's technique requires only one LFSR per prime factor. For large systems, however, the number of logic gates used for these shift registers becomes insignificant compared to the total used to generate each independent random sequence.

An *n*-bit modulo *m* addition for $n = \lceil \log_2 m \rceil$ can be viewed as a modulo *m* operation performed after the addition is done. Thus the modulo *m* addition follows:

$$y = (x_1 + x_2) \bmod m = \begin{cases} x_1 + x_2, & if\ x_1 + x_2 < m \\ x_1 + x_2 - m, & if\ x_1 + x_2 \geq m \end{cases} \qquad (5.2)$$

Generally, two methods can be used to complete the above computation: 1) Compute the results of both $x_1 + x_2$ and $x_1 + x_2 - m$, then select the correct result of modulo *m* addition from them. 2) Use a correction table to correct the addition $x_1 + x_2$ to the result $(x_1 + x_2) \bmod m$.

In the first method, two *n*-bit adders are used; the first adder computes $x_1 + x_2 - m$, while the second adder computes $x_1 + x_2$. The carry bit generated from the second adder indicates whether or not $x_1 + x_2$ is greater than *m* (Fig. 8). A multiplexer, controlled by the carry, selects the correct output.

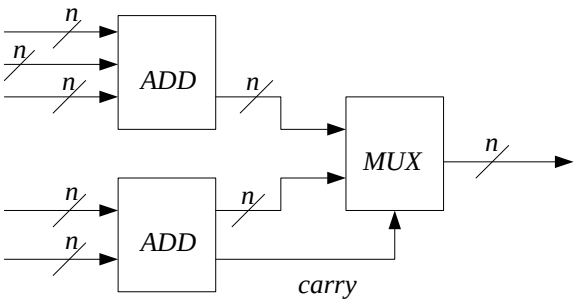

Fig. 8. A modulo adder can be constructed from two adders and a multiplexer.

In the second method, a lookup table is used to replace the first *n*-bit adder and the multiplexer in the first method (Fig. 9). When the lookup table in the ROM is small, i.e. when the modulus *m* is small, the second method can have better performance than the first method for fast table lookup.

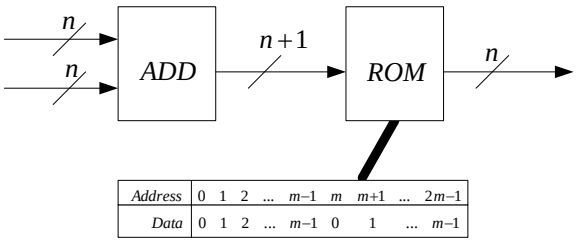

Fig. 9. A lookup table can be used when the modulus is small.

However, for an *n*-bit modulo addition the second method requires a 2^n-entry ROM with *n* bits for each entry. The hardware consumption for the lookup table is thus much greater than the first method when the size of the modulus is large. When this is the case the first method is the preferred implementation.

The ultimate circuit size achieved in practice will also depend on fanout considerations that affect the non-propagating sequence. As with any circuit affected by fanout, the traditional solution is to divide the circuit into smaller subcircuits and connect them together. If, for example, a non-propagating bit sequence needs to be divided into two subsequences because of fanout limitations, then two additional flipflops would be connected to the output of the non-propagating sequence's LFSR. Their outputs would separately drive the two subcircuits.

In a cellular automaton (CA) the next value of a cell is determined by its current value and the current values of its neighbors. If the output of every cell is used then the number of flipflops needed is the same as for our propagating LFSR technique. Often, however, CA sites are skipped to improve statistical performance, increasing the number of flipflops required. For a CA rule 90 cell the next value is the modulo-2 sum of two of its neighbors, which requires the same number of adders as for the propagating LFSR. A system made solely of rule 90 cells has poor statistical properties, however [7]. A rule 30 cell requires an additional OR gate and a rule 150 cell requires an additional modulo-2 adder. (Rules 30, 90, and 150 are explained in Section 6.) By their nature circuits based on cellular automata do not suffer from fanout constraints.

6 ASSESSING THE RANDOMNESS OF THE BITS

For an ideal ensemble of random bit generators in a circuit, every bit should be independent, both of the bits that came before it and of the bits produced by the other generators. While it is impossible to exhaustively identify whether sequences are truly random (indeed, we know that they are not), statistical tests have been developed to assess the relative irregularity of sequences of numbers or bits [15][16][17]. We have performed statistical tests to show that the bit generators proposed here are approximately random and to provide some comparisons with alternative random bit generators, including cellular automata [7] and counter-propagating LFSR sequences [2]. Our aim is not so much to tease out the most subtle correlations which are inevitably present with small generators, as it is to demonstrate that it is practical to obtain many simultaneous bitstreams, each of which is (approximately) random and all of which are (approximately) independent each other.

6.1 Statistical tests of randomness

In what follows, let $b(m)$ denote the bit sequence from the *m*th generator, with $b(m)_n$ the *n*th bit in that sequence. Many of the tests involve blocks of $M \times N$ bits. The block at time *t* is the set of bits $b(m)_n$ where $0 \le m \le M-1$ and $t \le n \le t+N-1$. In general we do not use overlapping blocks in these tests. If the blocks are independent, the statistics are more straightforward.

Equidistribution: The simplest requirement for a random bit generator is that the same number of 0's and 1's are generated, on average. The ***Eq-M×N*** test computes a histogram of the number of 1's observed in a sequence of $M \times N$ blocks, and performs a chi-squared test comparing that histogram to what is expected [17].

Correlation: A basic requirement for most applications is that random bit sequences *p* and *q* be uncorrelated: that is, the correlation $C\{p,q\} = \frac{1}{T}\sum_{n=1}^{T} 4[p_n - \frac{1}{2}][q_n - \frac{1}{2}]$ should be statistically indistinguishable from zero. For truly uncorrelated sequences, C is distributed with mean zero and variance $1/T$. We will test for three kinds of correlations: ***Corr-1×N*** is based on the autocorrelation from a single bit generator; that is, $b(0)$ and $\sigma^t b(0)$ for $1 \le t \le N-1$. ***Corr-M×1*** corresponds to "spatial correlation" between the sequences $b(0)$ and $b(m)$ for $1 \le m \le M-1$, and ***Corr-M×N*** is the "spatio-temporal" correlation between sequences $b(0)$ and $\sigma^t b(m)$ for $0 \le t \le N-1$ and $0 \le m \le M-1$ but not $t = m = 0$.

Approximate Entropy: An important measure of randomness is the predictability of a bit, given the bits that preceded it. The ***ApEn-M×N*** statistic provides a measure of that predictability by

estimating the entropy difference between a block of $M \times N$ bits and an augmented block of $M \times N+1$ bits. The extra bit is spatially in the center of the block and temporally just after it. If that extra bit is truly random, then the entropy of the augmented block should be log(2) larger than the entropy of the m bits. For a block of $M \times N$ bits, there are 2^{MN} patterns. If π_i indicates the frequency of occurrence for the ith pattern, then $\Phi(m) = \sum_{i=0}^{2^{MN}-1} \pi_i \log \pi_i$ is the negative entropy associated with that block, and $n\left[\Phi(m) - \Phi(m+1) - \log 2\right]$ is a statistic which indicates whether the last bit is random with respect to the previous m bits [17][18].

Runs: A "run" of bits is a subsequence in which all the bits have the same value. In the sequence 000110111, for example, there are four runs: 000, 11, 0, and 111. The runs statistic compares the number of runs in a long sequence with the expected number [17]. We apply the statistic to M simultaneous sequences (M odd) by combining the sequences to a single bit sequence. ***Runs-M*** refers to the same test applied to a bit sequence obtained by adding the M sequences *b(0), …, b(M-1)*, where M is odd, and taking a 0 if the sum is less than $M/2$ and 1 if the sum is greater than $M/2$.

Rank: The rank test is based on the rank of a square matrix produced from the bits of the generator. The rank is equal to the number of linearly independent columns in the matrix, where the linear operators are defined in the modulo 2 algebra. For example the 3x3 matrix

$$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

has rank two, because the first two columns are linearly independent, but the third column is the sum of the first two. ***Rank-M×N*** arranges the MN bits in the $M \times N$ block into a square $k \times k$ matrix where $k = \sqrt{MN}$. In each case the number of matrices with rank k, $k-1$, and less than $k-1$, are tabulated and used in a statistic [17].

6.2 Pseudorandom bit generators

We will use the above tests to investigate the randomness of a few variants of our proposed random number generation scheme, as well as several random bit generators that have been proposed previously. We will in particular compare our generation method to four of the most successful CA based generators [7] and to the counter-propagating LFSR [2]. We did not consider the generators given by Alspector *et al.* [1] since each generator has to be hand-designed and large arrays of

simultaneous bitstreams are thus impractical. In general, we used small generators – both the CA-based and the LFSR-based random generators improve if larger variants are used.

In our proposed scheme, each generated bitstream is an XOR combination of two different LFSR generators. For our ***LFSR-prop*** we used two base generators associated with the primitive polynomials $f(x) = x^{15} + x^1 + 1$ and $f(x) = x^{17} + x^3 + 1$, so that $R = \sigma^{-15} R \oplus \sigma^{-15+1} R$, $S = \sigma^{-17} S \oplus \sigma^{-17+3} S$, and the bitstream *b(m)* is given by $R + \sigma^m S$. Here R is the nonpropagating and S is the propagating generator. The counter-propagating ***LFSR-counterprop*** method that was introduced by Cawerberghs [2] differs from *LFSR-prop* in that the *m*th bitstream is given by $b(m) = \sigma^{-m} R + \sigma^m S$.

Cellular automata methods have been investigated by Hortensius et al. [7]. Here the next bit of a bitstream $b(m)_{t+1}$ is given by a function of the three current bits $b(m-1)_t$, $b(m)_t$, and $b(m+1)_t$. ***CA-30*** is a 30-bit wide CA based on CA rule 30, given by $b(m)_{t+1} = b(m-1)_t \oplus \left(b(m)_t \cup b(m+1)_t \right)$. To alleviate the local correlations, a "site spacing" scheme was introduced in Hortensius et al. [7]. Here, some number γ of sites are skipped for each output bitstream. For the $\gamma = 4$ variant of *CA-30*, which only has 30 sites to begin with, this leaves only six output bitstreams. In a hybrid CA, different bitstreams *m* employ different rules. The hybrid CA's generally produce better random bits than the pure CA's, but they require much more careful design. The ***CA-hybrid*** we investigate is the preferred hybrid in [7]. It uses rule 90, given by $b(m)_{t+1} = b(m-1)_t \oplus b(m+1)_t$, and rule 150, given by $b(m)_{t+1} = b(m-1)_t \oplus b(m)_t \oplus b(m+1)_t$. In a hybrid CA, the pattern of alternation between the two rules must be determined in advance. For a CA with 28 bits, it turns out that the optimal pattern is simple alternation. We also consider ***CA-hybrid-B29***, which is the same alternating rule but with 29 bits. Simple alternation is not the best pattern for a 29 bit wide CA, and we find that this bit generator fails many of the tests. We also consider a $\gamma = 1$ version of the hybrid CA.

In all cases, we generated sequences of two million time-steps, but only kept the last million bits for testing. This represents the performance of bit generators after they have had a chance to "warm up." We used three different random number seeds, and based on those seeds employed the minimal standard random number generator of Park and Miller [19] to initialize the various bit generators. Since our aim was to identify specific flaws in the generators, we considered a generator to fail a test only if failed two out of three runs. In practice we found that most of the time a generator would either pass all runs or fail all runs. Occasionally there would be one failure for a given test; which

could be due to a subtle weakness in the generator or because of an inevitable random fluctuation that will happen when so many different tests are applied.

Table 1 shows that none of the random bit generators passed all of the tests, and in particular the rank tests were most stringent. In short runs, without a "warm up" period, the *CA-30* generator actually did better, but after a long transient time a trajectory was reached in which the statistical properties of the bits was poor. Experiments with larger *CA-30* generators also showed better performance (results not shown).

We see that the CA generators, in general, have trouble with the *ApEn* tests and the *Runs* tests, although the $\gamma = 1$ variant of the hybrid CA did pass the *Runs* tests. The sensitivity of the hybrid CA to the details of its design is evident in the performance of the *CA-hybrid-B29* generators, which was terrible. Adding another cell to an existing hybrid CA generator generally requires the reconstruction of the rule sequence for the entire system [7]. The propagating LFSR and non-hybrid CA generators have no such restriction, making them more suitable for systems that are designed as independent modules or that are frequently modified.

The LFSR tests were more robust to the ApEn and Runs tests, though the counter-propagating LFSR failed the *ApEn-3x3* test.

	CA-30	*CA-30, γ=4*	*CA-hybrid*	*CA-hybrid, γ=1*	*CA-hybrid-B29*	*CA-hybrid-B29, γ=1*	*LFSR-counterprop*	*LFSR-prop*
Eq-1x1								
Eq-1x25	**F**	**F**				**F***		
Eq-25x1	**F**	**--**		**--**	**F***	**--**		
Eq-5x5		**F**			**F**	**F**		
Corr-1x100	**F**	**F**			**F***	**F**		
Corr-100x1	**--**	**--**	**--**	**--**	**--**	**--**		
Corr-10x10	**F**	**--**			**F**	**F**		
ApEn-1x5	**F**	**F**			**F**	**F**		
ApEn-5x1	**F**	**F**	**F**		**F**	**F**		
ApEn-3x3	**F**	**F**	**F**	**F**	**F**	**F**	**F**	
ApEn-5x2	**F**	**F**	**F**	**F**	**F**	**F**	**F**	
Rank-1x25	**F**	**F**			**F**	**F**		
Rank-25x1	**F**	**--**		**--**	**F**	**--**		**F***
Rank-3x3	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
Rank-5x5	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
Runs-1	**F**	**F**			**F**	**F**		
Runs-3	**F**	**F**	**F**		**F**	**F**		
Runs-7	**F**	**--**	**F**		**F**	**F**		

Table 1. Results of the application of a suite of statistical tests applied to a variety of random bit generators. The tests and the generators are described in the text. An 'F' indicates that the generator failed all three instances of the test, an 'F*' indicates failure in two out of three instances. The dash '—' indicates that for the particular generator there are an insufficient number of bitstreams to perform the test.

7 CONCLUSIONS

In this paper we described a methodology for the generation of multiple random bit sequences that simplifies and generalizes Cauwenberghs' counter-propagation algorithm [2]. We demonstrated

that the propagation of both sequences can be replaced by one propagating and one non-propagating sequence, thereby cutting the number of flipflops in half. Like counter-propagation, the proposed method reduces the routing requirements to only two 1-bit signals. It preserves the ability for new client circuits to be added to the system without additional calculations – there is no need to keep track of random starting values, tap combinations, or time shifts. The methodology was also extended to random number sequences uniformly distributed over the range of integers $[0, K]$, where $K+1$ need not be either prime or a power of two.

8 REFERENCES

[1] J. Alspector, J. Gannett, S. Haber, M. Parker, and R. Chu, "Generating Multiple Analog Noise Sources from a Single Linear Feedback Shift Register with Neural Network Applications," in *Proc. IEEE Int. Symp. on Circuits and Systems*, 1990, vol. 2, pp. 1058-1061.

[2] G. Cauwenberghs, "An Analog VLSI Recurrent Neural Network Learning a Continous-Time Trajectory," *IEEE Trans. on Neural Networks*, vol. 7, no. 2, pp. 346-361, Mar. 1996.

[3] J. Ortega, C. Janer, J. Quero, L. Franquelo, J. Pinilla, and J. Serrano, "Analog to Digital and Digital to Analog Conversion Based on Stochastic Logic," in *Proc. IEEE Int. Conf. on Industrial Electronics, Control, and Instrumentation*, 1995, vol. 2, pp. 995-999.

[4] J. Quero, S. Toral, J. Ortega, and L. Franquelo, "Continuous Time Filter Using Stochastic Logic," in *Proc. Midwest Symp. on Circuits and Systems*, 2000, vol. 1, pp. 113-116.

[5] B. Gaines, "Stochastic Computing Systems," *Advances in Information Systems Science*, J. Tou, ed., vol. 2, New York: Plenum Press, 1969, pp. 37-172.

[6] R. Kuehnel, "Binomial Logic: Extending Stochastic Computing to High-Bandwidth Signals," in *Proc. Asilomar Conf. on Signals, Systems, and Computers*, 2002, vol. 2, pp. 1089-1093.

[7] P. Hortensius, R. McLeod, and H. Card, "Parallel Random Number Generation for VLSI Systems Using Cellular Automata," *IEEE Trans. on Computers*, vol. 38, no. 10, pp. 1466-1473, Oct. 1989.

[8] J. P. R. Toothill, W. D. Robinson, and A. G. Adams, "The Runs Up-and-Down Performance of Tausworthe Pseudo-Random Number Generators," *Journal of the ACM*, 1971, vol. 18, pp. 381-399.

[9] S. Tezuka and P. L'Ecuyer, "Efficient and Portable Combined Tausworthe Random Number Generators," *ACM Trans. On Modeling and Computer Simulation*, 1991, vol. 1, pp. 99-112.

[10] J. Saarinen, J. Tomberg, L. Vehmanen, K. Kaski, "VLSI Implementation of Tausworthe Random Number Generator for Parallel Processing Environment," *IEE Proceedings-E*, 1991, vol. 138, no. 3, pp. 138-146.

[11] J. Fillmore and M. Marx, "Linear Recursive Sequences," *SIAM Review*, 1968, vol. 10, no. 3, pp. 342-353.

[12] R. Kuehnel and Y. Wang, "A Method of Generating Uniformly Distributed Sequences over [0,K], where K+1 is not a Power of Two," *Proc. IEEE Int. Conf. On Acoustics, Speech, and Signal Processing*, 2003, vol. 2, pp. 801-804.

[13] http://fchabaud.free.fr/English

[14] V. Yarmolik and S. Demidenko, *Generation and Application of Pseudorandom Sequences for Random Testing*, New York: John Wiley and Sons, 1988.

[15] G. Marsaglia, "Diehard Battery Tests of Randomness," http://stat.fsu.edu/pub/diehard.

[16] P. L'Ecuyer, "Testing Random Number Generators," *Proc. 1992 Winter Simulation Conference*, pp. 305-313.

[17] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, S. Vo, "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," *NIST Special Publication 800-22*. http://csrc.nist.gov/rng/.

[18] A. L. Rukhin, "Approximate Entropy for Testing Randomness," *Journal of Applied Probability*, 2000, vol. 37, pp. 88-100.

[19] S. Park and K. Miller, "Random Number Generators: Good Ones are Hard to Find," *Communications of the ACM*, 1988, vol. 31, pp. 1192-1201.

Reviewers' and Associate Editor's Comments
==========

Recommendation

Resubmit after Minor Revision for Review as a Regular Paper

Comments to the Author

This paper will be rejected if the authors do not revise their manuscript according to reviewer's comment carefully, especially compare their result with those of published papers.

Review Number 1.
****

Does the revision adequately address the concerns expressed in the original review?

Yes.

Comments to the Author

On page 12 of the revised paper, line 4 from above, states that if K+1 is equal to a power of two, then equation (4.3) reduces to equation (4.4). However, the authors have removed equation (4.3) and refer back to equation (2.13) in the revised paper. Therefore, "equation (4.3)" should be replaced by "equation (2.13)". Moreover, since (4.2) and (4.3) have been removed, (4.4) and (4.5) should be renumbered to (4.2) and (4.3).

Oops! We thank the reviewer for catching these errors. We have made these changes and carefully reviewed the equation numbering in all of the sections.

Review Number 2.
****

Does the revision adequately address the concerns expressed in the original review?

No.

Comments to the Author

Please make comparisons with other works, otherwise I cannot see the significance of this work.

We have completely rewritten Section 6 to include statistical comparisons of our approach to several CA-based generators, as well as to the counter-propagating LFSR. The new section includes a suite of statistical tests (previously we only considered pairwise correlation), and a table showing how each generator fared with each

test. The tests were sufficiently rigorous that no generator passed all of them.

Although our original aim was just to show that our generator did not have any glaring correlations, we think that the more careful comparative study does improve the paper, and we thank the reviewer for suggesting the comparisons.

Review Number 3.
****

Comments to the Author

This paper presents a design methodology for the efficient generation of multiple pseudo-random number sequences that are statistically uncorrelated. The contribution is to propose the simple methodology of generating a uniform random number between the range [0,K] for any integer number K. Theorem2 can be the generalization of the counter-propagation method illustrated in Fig.2. However, based on theorem2, a random number over a prime-number range is also proposed as Fig.3. It allows replacing one propagation-input by a non-propatation input in Cauwenberghs' algorithm. But according to the definition of the two sequences u(1) and u(2) in page 7, it is not intuitive to apply theorem 2 to Fig.2 directly. Therefore, this paper is little difficult to read.

The reviewer's point is well-taken. We were a little loose in our application of Theorem 2 (which, as stated, really only applies to the system in Fig 3) to the counterpropagating system in Fig. 2. We have reworked the explanation at the beginning of Section 3 so that Theorem 2 is applied more appropriately. The result is the same, but we hope this clarifies the argument and makes this part of the paper a little easier to read.

Correction:
a) There is no equation (4.3) in the description on page12. Besides, equations (4.2) and (4.3) seem to be missing in section 4.

Thanks – our numbering was incorrect and we have fixed it.

b) In Fig.7, does the term "next least bit" refer to the bit next to the least significant bit? (If the 6 bits correspond to the card selected)

The reviewer is correct and to make this clearer in the paper we have added this explanation to the text that describes the figure.