Efficient Generation of Statistically Good Pseudonoise by Linearly Interconnected Shift Registers

W. J. Hurd
Communications Systems Research Section

Some new algorithms are presented for generating pseudorandom noise utilizing binary maximal length linear recursive sequences of high degree and with many nonzero terms. The ability to efficiently implement high degree recursions is important because the number of consecutive bits which can be guaranteed to be both linearly and statistically independent is equal to the degree of the recursion. The implementations are by interconnection of several short shift registers in a linear manner in such a way that different widely spaced phase shifts of the same pseudonoise sequence appear in the stages of the several registers. This is efficient both in hardware and in software. Several specific algorithms are subjected to extensive statistical evaluation, with no evidence found to distinguish the sequences from purely random binary sequences.

I. Introduction

Digitally generated pseudorandom noise and analog-generated random noise are extensively used in various research, development, simulation, testing and system evaluation and calibration activities. Digital pseudonoise has basic advantages over analog-generated noise, in its repeatability and inherent stability. Therefore, analog noise signals are frequently generated by converting digital noise to analog, and digital computer applications rarely use analog-generated noise.

This paper presents some efficient new algorithms for generating digital pseudonoise. The algorithms are efficient because a large number of new pseudorandom bits are generated at each iteration of a computer implementation, or at each clock pulse of a hardware implementation. The pseudonoise has good statistical properties because the several simultaneously generated bits are the corresponding bits from different phase shifts of the same maximal length linear recursive sequence. These sequences, also called maximal length shift register sequences, *pn*-sequences, and *m*-sequences, are well known to have good randomness properties (Refs. 1 and 2).

Some of the key features of the new algorithms are:

(1) The algorithms are efficient in both hardware and software implementations. Besides being independently useful in the two applications, this has the advantage that specific proposed hardware implementations can be efficiently simulated and evaluated before hardware is constructed.

(2) In software applications, the new algorithms are more efficient than existing algorithms for *pn*-

sequences with comparable randomness properties. Some efficient algorithms are known for particularly simple recursions, but the resulting sequences have poor statistical properties which the new algorithms avoid.

(3) Wideband pseudo-gaussian analog signals can be easily generated from hardware implementations. This is accomplished by the analog summation of voltage waveforms corresponding to the several new bits generated at each clock pulse. A noise generator has been constructed utilizing this principle, in similar manner to an earlier digital gaussian noise generator described elsewhere (Ref. 3).

(4) The statistical properties of the sequences have been evaluated extensively, with no evidence found to distinguish the sequences from purely random sequences of independent, equally likely, binary numbers.

This paper is divided into three main sections. First, some of the properties of *pn*-sequences are reviewed, and standard implementations are discussed. This serves as a motivation for the development of the new algorithms. Second, the new algorithms are described in general, and specific implementations are discussed. Finally, the results of extensive statistical evaluation of the pseudo-noise are presented, and the sequences are shown to have favorable randomness characteristics.

II. Motivation for Use of *pn*-Sequences

A binary linear recursive sequence is a sequence $\{X_k\}$ of *zeroes* and *ones* satisfying a linear recursion of the form

$$X_k = \sum_{i=1}^{n} a_i D^i X_k \qquad (1)$$

where D is the delay operator, i.e., $D^i X_k = X_{k-i}$, the a_i are *zero* or *one*, and addition is modulo-2. The degree of the recursion is the largest value of i for which $a_i = 1$, and the maximum possible period of a sequence from a linear recursion of degree n is $2^n - 1$. These maximal length linear recursive sequences, called *pn*-sequences, occur when the polynomial

$$P(D) = 1 + \sum_{i=1}^{n} a_i D^i$$

is primitive over $GF(2)$.

A. Randomness Properties

PN-sequences are known to have many favorable randomness properties (Refs. 1 and 2). Some important properties common to all *pn*-sequences are:

(1) For degree n, all of the $2^n - 1$ possible nonzero n-tuples, or sets of n consecutive bits, occur equally often. This means that binary numbers formed from disjoint subsets of the same n-tuple are independent and jointly uniformly distributed.

(2) All phase shifts are essentially uncorrelated, when correlation is defined as the number of places in which the phase shifts agree, less the number in which they disagree.

(3) Under suitable conditions, sets of n-tuples from different phase shifts of a sequence are uncorrelated, when considered as binary numbers (Ref. 2).

These properties indicate not only that *pn*-sequences are good sources of random numbers, but that it may be possible to utilize different phase shifts of the same sequence as essentially independent noise sequences.

The properties we have discussed so far apply equally to all *pn*-sequences, but other properties cause some sequences to appear more random than others. In particular, consider m-tuples for m greater than the degree n. Since some of the bits of the m-tuples are linearly related, they are not statistically independent. Fortunately the statistical dependence is usually not observed unless the recursion is a particularly simple one. Lindholm (Ref. 4), however, has investigated the weights of m-tuples, i.e., the number of *ones*. In a purely random sequence, the weights would be binomially distributed symmetrically about $m/2$, but there is significant deviation from this when the recursion is a trinomial or divides some trinomials of low degree.

The above properties of *pn*-sequences indicate that the randomness properties tend to improve with the degree and complexity of the recursion. They also indicate that bits from several phase shifts of the same sequence might be as useful as the same total number of bits from one phase shift. As we shall see, this has implementation advantages.

B. Well-Known Implementation

PN-sequences are easily generated one bit at a time in binary shift registers, as shown in Fig. 1. The input to the first stage is labeled X_k, and the outputs of the

n-stages are X_{k-1}, $X_{k-2}, \ldots, X_{k-n}$, the values of X_k at the n previous instants of time. As the time index advances from k to $k+1$, the state of each stage of the register assumes the value of its input. The register shown satisfies the trinomial recursion

$$X_k = X_{k-1} + X_{k-n} \pmod 2 \tag{2}$$

so that its characteristic polynomial in the delay operator is

$$P(D) = 1 + D + D^n \pmod 2 \tag{3}$$

The resulting sequence $\{X_k\}$ is a *pn*-sequence if $P(D)$ is primitive.

One can generate several phase shifts of the same sequence by several straightforward methods, using the cycle-and-add property. This property is that the mod-2 sum of any two phase shifts is another phase shift. Thus one method is to sum the outputs of any two stages to yield a new phase shift. Another method can be used if the input X_k depends on several, say N, of the previous n states. Then X_k is implemented in a series of $N-1$ two-input modulo-2 adders, and the output of each adder is a different phase shift. Both of these methods suffer the implementation deficiencies that the complexity increases with the number of terms in the recursion and the number of phase shifts generated, and that they are not amenable to software systems. They have the statistical deficiency that the various phase shifts are simply related, and therefore cannot be considered statistically independent, even though they must be uncorrelated.

In software, it is fairly easy to generate n successive bits of a degree n recursion in a few machine instructions, provided that the recursion is a trinomial, as was done by Kendall (Ref. 5) for $n = 47$. Unfortunately, the algorithm complexity increases with the number of terms in the recursion, and trinomial recursions result in statistical dependencies which sometimes lead to erroneous simulation results, as observed by Heller (Ref. 6). Furthermore, the method is efficient only when n is less than the number of bits in one or two computer words, because of the shifting operations which are required.

In summary, the considerations above indicate that there is a need both in hardware and software for algorithms for efficiently generating long period *pn*-sequences with complex recursions. The efficiency is to be gained by generating several new bits simultaneously, either consecutive bits from a sequence, or bits from several different phase shifts of a sequence. It is the latter approach which we develop here.

III. Sequence Generation by Interconnection of Shift Registers

The algorithms presented here utilize the linear interconnection of several shift registers to simultaneously generate several phase shifts of the same *pn*-sequence. For a degree n recursion, the n bits in memory at any one time are not consecutive bits from one *pn*-sequence, but are bits from several phase shifts. Nevertheless, they are linearly independent, and they retain the important statistical property that all disjoint subsets, considered as binary numbers, are independent and jointly uniformly distributed. This follows because all of the $2^n - 1$ possible nonzero states of the n bits occur equally often.

The algorithms are efficient both for hardware and software. The theoretical minimum of n stages of shift register and still fewer gates are required in hardware for a polynomial of degree n, and one new word is generated at each clock pulse. In software, approximately 12 machine instructions are required to generate one new word of pseudorandom bits, even though the degree of the polynomial may be much higher than the word length. Since the degree determines the number of consecutive bits which are guaranteed to be statistically independent, a number of successive independent computer words can be generated without sacrificing efficiency.

Figure 2 shows a simple linear interconnection of three shift registers. The registers are labeled 0, 1, and 2, with inputs at time k of $X_k^{(0)}$, $X_k^{(1)}$, and $X_k^{(2)}$. At time k, the first (leftmost) stages of the registers store the input values at time $k-1$, i.e., $X_{k-1}^{(0)}$, $X_{k-1}^{(1)}$, and $X_{k-1}^{(2)}$. The shifting is from left to right, so, in general, stage j of register i stores the value $X_{k-j}^{(i)}$ at time k.

In the particular example of Fig. 2, register 0 has three stages, and registers 1 and 2 have four stages. The input to each register is the modulo-2 sum of the output of the last stage of the same register, and the output of one stage from the previous register. Thus we can define the recursion by

$$X_k^{(i)} = X_{k-q_i}^{(i)} + X_{k-d_{i-1}}^{(i-1)}, \qquad i = 0,1,2 \pmod 3 \tag{4}$$

where q_i is the number of stages in register i, and d_{i-1} is the stage of register $i-1$ which forms an input to register i. We can also write

$$X_k^{(i)} = D^{q_i} X_k^{(i)} + D^{d_{i-1}} X^{(i-1)}, \qquad i = 0,1,2 \pmod 3 \tag{5}$$

and this set of equations is easily solved in general for the characteristic polynomial in the delay operator as

$$P(D) = \prod_{i=0}^{N-1} (1 + D^{q_i}) + D^{\sum_{i=0}^{N-1} d_i} \tag{6}$$

where we have now generalized to N registers, labeled $0,1,2,\ldots,N-1$.

For the particular case of Fig. 2,

$$P(D) = 1 + D^3 + D^5 + D^8 + D^{11} \tag{7}$$

which is a primitive polynomial, as can be verified by calculation or from tables (Ref. 7). Thus the sequence of states of the stages of each register is the *pn*-sequence defined by $P(D)$, or by the recursion

$$X_k = X_{k-3} + X_{k-5} + X_{k-8} + X_{k-11} \tag{8}$$

C. Final Configuration

To achieve simple implementations, it is necessary to restrict the register interconnections to some regular form. However, forms which have each input depend on only two register stages, as in Fig. 2, are probably not satisfactory, because they tend to suffer some of the statistical deficiencies of trinomial recursions, even though trinomial characteristic polynomials do not typically result.

The next alternative could be to have each input depend on three register stages. Most configurations in which each input depends on at least three register stages would probably be satisfactory statistically. However, for implementation considerations, we have chosen, instead, to have the first stage of each register depend on only two inputs and to modify the connections to the last stage of each register so that its input is the sum of its own output state and the state of the preceding stage. This operation is known as toggling, because the stage toggles, i.e., changes state, whenever its input is 1. This is the natural operation of a T flip-flop, or a J-K flip-flop with the two inputs the same. In delay operator notation, this stage performs the operation

$$\frac{D}{1+D}$$

instead of the operation D.

The final configuration consists in general of N registers of the form shown in Fig. 3. Register i has q_i stages, the first $q_i - 1$ of which shift, and the last stage of which toggles. Thus the stage outputs of register i at times k are

$$DX_k^{(i)}, D^2X_k^{(i)}, \ldots, D^{q_i-1}X_k^{(i)} \text{ and } \frac{D^{q_i}}{1+D}X_k^{(i)}$$

The input $X_k^{(i)}$ to register i is the modulo-2 sum of the last stage of register $i-1 \pmod N$ and stage d_{i-2} of register $i-2 \pmod N$. Thus the system is defined by the equations

$$\left.\begin{aligned} X_k^{(i)} &= \frac{D^{q_{i-1}}}{1+D} X_k^{(i-1)} + D^{d_{i-2}} X_k^{(i-2)} \\ i &= 0,1,\ldots,N-1 \pmod N \end{aligned}\right\} \tag{9}$$

D. Specific Realizations

In order to find specific systems corresponding to primitive polynomials, it is necessary to calculate the polynomials for various values of the system parameters and to test for primitivity. This is best done with a computer, and programs have been written for this purpose. To test for primitivity, one computes $D^r \pmod{P(D)}$ for all integers r which divide $2^n - 1$. The polynomial $P(D)$ is primitive if $r = 2^n - 1$ is the smallest value of r such that $D^r \equiv 1 \pmod{P(D)}$. This test cannot be performed for all degrees n, because the factors of $2^n - 1$ are not known in general. Furthermore, the average number of computations required to find a primitive polynomial increases as n^4. The highest degree for which a primitive polynomial system of the special form was found is 310.

Table 1 summarizes some of the realizations found which have primitive characteristic polynomials. This table is restricted to equal length shift registers of length q, with degree $n = Nq$. The d_i are also restricted. They are allowed to assume only two values, $d_i = d_0$ for $i = 0,1,\ldots,N_0 - 1$, and $d_i = d_{N-1}$ for $i = N_0, N_0 + 1, \ldots, N-1$. In other words, the first N_0 of the d_i are equal to d_0, and the rest are equal to d_{N-1}. Column T in Table 1 gives the number of nonzero coefficients in the resulting polynomial, i.e., it is a T-nomial.

Many more primitive configurations can be found by lifting the restrictions on the q_i and d_i.

E. Software Implementation

For computer implementation, realizations using equal length registers are most useful, especially when the number of registers is equal to the computer word length. If all registers are of length q, then a q word array of memory is used, as shown in Fig. 4. The last word of the array stores the first bit of each of the N shift registers, the next to last word stores the second bit of each register, etc. Thus word $q - j + 1$ stores

$$D^j X_k^{(0)}, D^j X_k^{(1)}, \ldots, D^j X_k^{(N-1)}$$

for $j = 1,2,\ldots, q-1$, and word 1 stores

$$\frac{D^q}{1+D} X_k^{(0)}, \frac{D^q}{1+D} X_k^{(1)}, \ldots, \frac{D^q}{1+D} X_k^{(N-1)}$$

We can envision shifting words upwards through the array, with toggling of the first word.

The computer program must accomplish three things, in principle simultaneously. First, it must selectively merge the various words to form a word

$$D^{d_0} X_k^{(0)}, D^{d_0} X_k^{(1)}, \ldots, D^{d_{N-1}} X_k^{(N-1)}$$

and modulo-2 add the proper shift of this word to the proper shift of word 1 to form the new word q. Second, it must form the new word 1 by modulo-2 adding words 1 and 2. Third, it must shift the old words $q, q-1, \ldots, 3$ into $q-1, q-2, \ldots, 2$. In practice, it is more efficient to generate a new array of q words at one time, i.e., the N registers are each shifted q places, yielding n new pseudo-random bits.

Several specific algorithms have been programmed for the XDS Sigma 5 computer. The statistical properties of these sequences are evaluated in Section IV, and a FORTRAN program for degree 288 with $N = 32$, $q = 9$ is described in the Appendix. In this particular program, the registers have been permuted so that the input to register i depends on registers $i-5$ and $i-10$, instead of on registers $i-1$ and $i-2$. The program execution time is 37 μs per 32-bit number.

F. Hardware Applications

The realizations for small n and N are useful in hardware applications. A pseudo-gaussian noise generator has been constructed for $N = 12$, $n = 60$ using only 20 integrated circuits. For this implementation, the d_i for $i = 0,1,\ldots, N-1$ are 3,2,1,4,2,4,3,2,1,4,4,1, and the polynomial has 25 nonzero coefficients. To transform from a binary to a pseudo-gaussian sequence, voltage waveforms corresponding to the 12 new bits generated at each clock pulse are summed in a resistor network and then filtered, in a similar manner to the noise generator of Ref. 1. The advantage to the new method over the earlier noise generator is that less hardware is required so that the cost is substantially less. Furthermore, the pseudonoise properties of this noise generator are very good, as will be shown next.

IV. Evaluation of Statistical Properties

Extensive chi-squared tests were performed on the sequences generated by computer programs for algorithms of degrees 310, 288, 160, and 60. The program for degree 60 implemented the same system as the hardware noise generator described above. The other three cases are the polynomials with the largest numbers of nonzero coefficients given in Table 1. These were evaluated to determine their suitability for use as standard computer noise generator algorithms. The sequences were tested to see whether computer words treated as binary numbers were jointly uniformly distributed in several dimensions, and the weight distributions of m-tuples were tested for being binomial for various m. The results of these tests showed no evidence to distinguish any of these pseudonoise sequences from truly random binary sequences.

A. Tests for Jointly Uniform Distributions

The results of the tests for uniformity are summarized in Table 2. To illustrate the use of this table, we explain the entries in each column of the second row. The first column identifies the case as corresponding to the degree 60 polynomial, or the one implemented in hardware. The second and third columns indicate that this test was for joint uniformity in three dimensions, with 999 degrees of freedom. This means that the data were sorted into 1000 bins, and since all bins were of equal size, the bin dimensions are $0.1 \times 0.1 \times 0.1$ if the numbers are considered to be between 0 and 1. The next column indicates that 25,000 data samples were used to compute each value of χ^2. In this case, 25,000 sets of three numbers were required. The last five columns indicate that 100 tests were run; the values of χ^2 which should be exceeded 99% and 95% of the time were exceeded in all but 0 and 10 tests, respectively; the values which should be exceeded 1% and 5% of the time were exceeded 0 and 8 times, respectively.

In order to search for any possible anomalies in the joint distributions of nonadjacent words on the sequences, the data samples used for the tests of Table 2 were not all adjacent words in the sequences. For a k-dimensional test, the first five sets of k words were adjacent on the sequence, the k words of the next five sets were spaced two words apart, the k words of the next five sets were spaced three words apart, etc., until the spacing between the words of last five sets of k-words was equal to the number of tests divided by five.

For all cases, the number of observances of large and small values of χ^2 are very close to the expected values of these events for independent and uniformly distributed random numbers. In particular, for the 2965 tests performed on the degree 288 polynomial, a total of 20 values of χ^2 were less than the 99% value, and 25 exceeded the 1% value. This is, in both cases, less than two standard deviations from the mean value of 29.65, since the standard deviation is 5.42. Considering the extensiveness of the tests performed, sequences whose statistics differed significantly from the statistics tested for would almost certainly result in many more large values of χ^2 than expected. Thus there is no evidence to distinguish the sequences from purely random sequences.

B. Test of Weight Distribution

For the same four algorithms, the distributions of the weights of m-tuples of consecutive bits were tested for being binomially distributed, as would be the case for a purely random sequence. This test was performed because pn-sequences generated by trinomials will fail this test.

Each m-tuple considered was made up of the bits of an integer number of words, where word length corresponds to the number of shift registers in the corresponding hardware implementation. Thus m is always a multiple of 12 for the degree 60 case, 31 for the degree 310 case, and 32 for the other two cases.

Table 3 summarizes the results of these tests. The columns in Table 3 are the same as in Table 2, except that the second column indicates the lengths of the m-tuples. Various m-tuple lengths up to more than 1200 were tested for each of the four algorithms. In all cases, the number of times χ^2 exceeded the 1% and 5% values was approximately as expected. Also, the numbers of values less than the 95% values were near the means.

The only unusual result was that only one value of χ^2 was less than the 99% value. We attribute this to the fact that the 99% threshold used was not the correct value for the actual observables, because the observables are only asymptotically chi-squared distributed. The deviation of the actual distribution from chi-squared is significant for small values. For example, a truly chi-squared variate can take on the value zero, whereas our observables had a minimum value greater than zero. This was because the observation space was divided into bins with non-integer expected numbers of occurrences. Since the numbers of observed values of χ^2 less than the 95% values were close to the mean, we do not consider the sparsity of extremely small values to be significant. The overall conclusion drawn from the weight distribution tests is that there is no evidence to distinguish the bit sequences from truly random sequences of independent, equally likely bits.

References

1. Golomb, S. W., *Shift Register Sequences.* Holden Day, San Francisco, 1967.
2. Tausworthe, R. C., "Random Numbers Generated by Linear Recurrence Modulo Two," *Math. Comp.*, Vol. 19, No. 90, April 1965, pp. 201–209. See also *Supporting Research and Advanced Development*, JPL Space Programs Summary 37-27, Vol. IV, pp. 185–189, Jet Propulsion Laboratory, Pasadena, Calif., June 30, 1964.
3. Hurd, W. J., "A Wideband Gaussian Noise Generator Utilizing Simultaneously Generated PN-Sequences," in *Proceedings of the Fifth Hawaii International Conference on System Sciences*, January 1972, pp. 168–170. See also *The Deep Space Network Progress Report*, Technical Report 32-1526, Vol. III, pp. 111–115, Jet Propulsion Laboratory, Pasadena, Calif., June 15, 1971.
4. Lindholm, J. H., "An Analysis of the Pseudo-Randomness Properties of Subsequences of Long m-Sequences," *IEEE Trans. Info. Theory*, Vol. IT-14, No. 4, July 1968, pp. 569–576.
5. Kendall, W. B., "A Generator of Uncorrelated Pseudo Random Numbers for Scientific Data Systems (SDS) Computers," *Supporting Research and Advanced Development*, JPL Space Programs Summary 37-34, Vol. IV, pp. 296–298, Jet Propulsion Laboratory, Pasadena, Calif., Aug. 31, 1965.
6. Heller, J. A., "Improved Performance of Short Constraint Length Convolutional Codes," *JPL Space Programs Summary 37-56*, Vol. II, p. 83, Jet Propulsion Laboratory, Pasadena, Calif., Apr. 30, 1969.
7. Peterson, W. W., *Error-Correcting Codes.* MIT Press and John Wiley, New York, 1961.

Table 1. Some primitive polynomial configurations

n	N	q	T	d_0	d_{N-1}	N_0
20	4	5	7	1	2	1
20	4	5	7	1	3	1
20	4	5	11	1	4	1
60	10	6	29	2	3	3
60	15	4	31	2	1	2
80	16	5	25	4	3	7
155	31	5	79	2	3	2
160	32	5	29	2	1	3
160	32	5	57	2	1	5
288	32	9	81	6	7	13
310	31	10	95	5	8	2
310	31	10	119	5	6	3
310	31	10	93	3	8	4
310	31	10	117	6	9	4
310	31	10	97	2	7	5
310	31	10	97	5	2	5

Table 2. Chi-squared tests for uniformity

Polynomial degree	Number of dimensions	Degrees of freedom	Samples per test	Number of tests	Values less than		Values exceeding	
					χ^2_{99}	χ^2_{95}	χ^2_{01}	χ^2_{05}
60	1	1023	25,000	100	1	7	1	6
60	3	999	25,000	160	0	10	0	8
60	Totals			260	1	17	1	14
160	3	999	25,000	225	2	9	1	11
288	1	999	25,000	250	2	20	3	7
288	2	960	25,000	750	8	46	11	37
288	3	999	25,000	750	5	30	4	24
288	4	1295	25,000	750	4	37	5	30
288	5	1023	25,000	465	1	19	2	19
288	Totals			2965	20	152	25	117
310	3	999	25,000	280	2	20	2	10

Table 3. Chi-squared tests on weight distributions

Polynomial degree	m-tuple length	Degrees of freedom	Samples per test	Number of tests	Values less than		Values exceeding	
					χ^2_{99}	χ^2_{95}	χ^2_{01}	χ^2_{05}
60	60	26	25,000	25	1	2	1	2
60	120	36	25,000	25	0	0	1	2
60	180	44	25,000	25	0	2	1	1
60	240	50	25,000	25	0	1	0	0
60	300	56	25,000	25	0	2	0	2
60	600	76	25,000	25	0	0	0	1
60	1200	106	25,000	25	0	0	1	2
60	1800	126	25,000	70	0	2	1	4
60		Totals		245	1	9	5	14
160	32	18	25,000	25	0	2	0	0
160	96	32	25,000	25	0	1	1	2
160	160	42	25,000	25	0	0	0	0
160	320	58	25,000	25	0	0	0	0
160	640	78	25,000	37	0	5	1	2
160	1280	108	25,000	25	0	1	2	3
160		Totals		162	0	9	4	7
288	32	18	25,000	25	0	1	0	0
288	96	32	25,000	25	0	0	1	1
288	160	42	25,000	25	0	1	0	4
288	320	58	25,000	25	0	1	0	1
288	640	78	25,000	25	0	0	0	1
288	640	88	100,000	25	0	2	0	2
288	1280	108	25,000	25	0	2	1	1
288		Totals		175	0	7	2	10
310	31	17	10,000	25	0	1	0	0
310	155	37	10,000	25	0	1	0	3
310	310	52	10,000	25	0	0	0	3
310	620	70	10,000	25	0	1	1	2
310	1240	96	10,000	10	0	0	0	0
310	1240	116	50,000	10	0	1	0	0
310		Totals		120	0	4	1	8

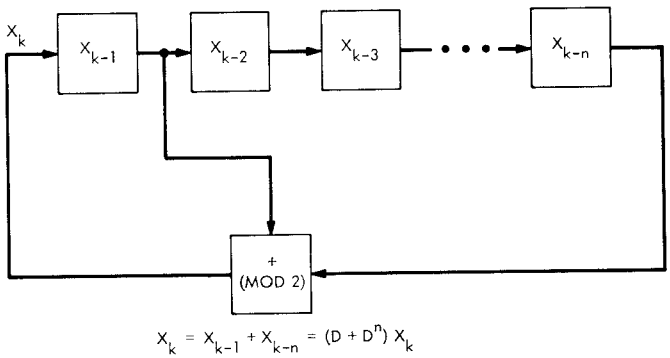

Fig. 1. Shift register of degree *n* with linear feedback

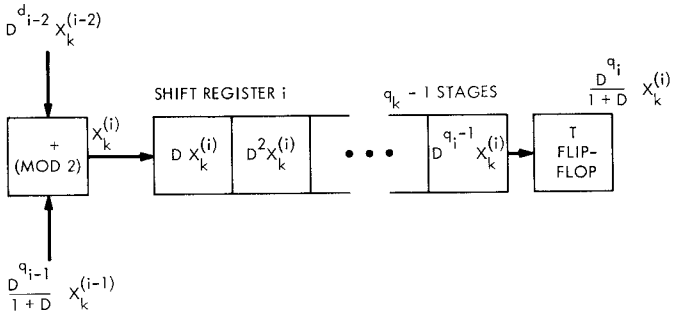

Fig. 3. General register of final configuration

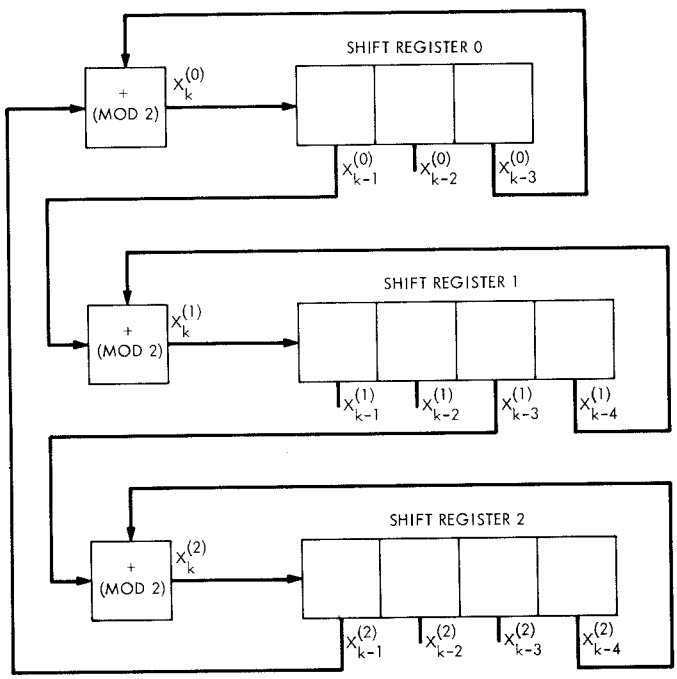

Fig. 2. A simple linear interconnection of three shift registers

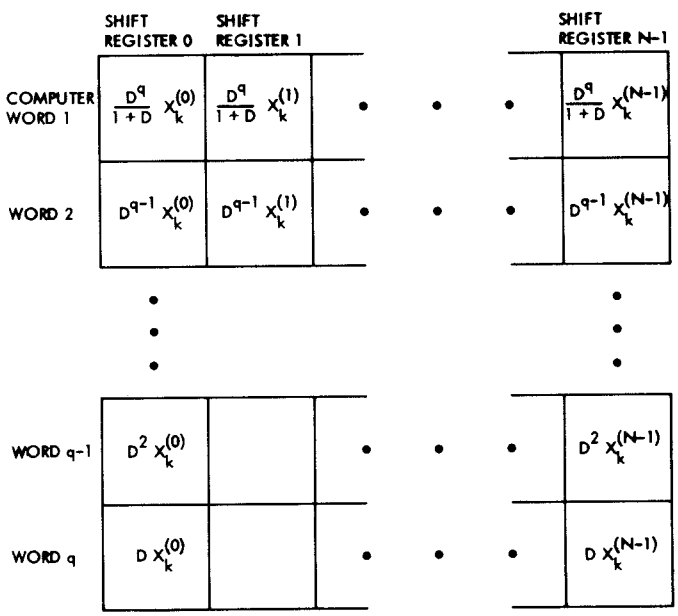

Fig. 4. Computer realization of shift registers

Appendix

XDS Sigma 5 FORTRAN Subroutine

The XDS Sigma 5 FORTRAN subroutine for the polynomial of degree 288 is listed in Fig. A-1. The program utilizes the machine dependent in-line code features of XDS Extended FORTRAN IV and is therefore not compatible with other machines. Usage is as follows:

(1) CALL PN288(N): Nine pseudorandom integers are returned in array N. Their distribution is uniform from -2^{31} to $2^{31}-1$.

(2) CALL PN1(M): One new number is returned in M.

(3) CALL PIX(N): The state of the nine words of memory determining the next output are returned in array N. These are not the same as returned by a CALL to PN-288.

(4) CALL PAX(N): Packs the nine words of memory from array N.

(5) CALL PIXJ(M): Returns the value of a pointer used by PN1.

(6) CALL PAXJ(N): Sets the pointer.

A general random number package utilizing this basic algorithm has been written by R. Winkelstein and will be reported on in a future article.

```
 1:         SUBROUTINE PN288(NOUT)
 2:         DIMENSION N(18),NOLT(0:1)
 3:         DATA J,NW/19,9/
 4:         DATA (N(I),I=1,9)/2036521439,-1235874521,1985321452,1478598765,
 5:        1-888521432,963598521,-1235741258,-2100352140,1865423518/
 6: S        BAL,2    2S
 7: S        LW,3     N+9
 8: S        LCI      9
 9: S        STM,3    *NOUT
10:         J=19
11:         RETURN
12: S2      LI,3     -9
13: S1      LW,4     N+11,3                 D7  INTO R4
14: S       LW,5     =X'94A5294A'
15: S       LS,4     N+12,3                 D6 INTO R4 ON MASK
16: S       SCS,4    -5
17: S       EOR,4    N                      ANS IN R4, NOT PROPERLY SHIFTED
18: S       SCS,4    -5                     ANS IN R4, PROPERLY SHIFTED
19: S       STW,4    N+18,3                 ANS IN N-ARRAY, LOWER PART
20: S       LW,5     N
21: S       EOR,5    N+10,3
22: S       STW,5    N                      NEW D9 INTO N
23: S       BIR,3    1S
24: S       LCI      8
25: S       LM,4     N+10
26: S       STM,4    N+1
27: S       B        *2
28:         ENTRY PN1(N1)
29:         IF(J.LE.18)GOTO 20
30:         J=10
31: S        BAL,2   2S
32:    20   N1=N(J)
33:         J=J+1
34:         RETURN
35:         ENTRY PAX(NOUT)
36:         DO 30 I=NW,1,-1
37:    30   N(I)=NOUT(I-1)
38:         J=19
39:         RETURN
40:         ENTRY PIX(NOUT)
41:         DO 40 I=NW,1,-1
42:    40   NOUT(I+1)=N(I)
43:         RETURN
44:         ENTRY PAXJ(JOUT)
45:         J=MAXO(JOUT,NW+1)
46:         RETURN
47:         ENTRY PIXJ(JOUT)
48:         JOUT=J
49:         RETURN
50:         END
```

Fig. A-1. XDS Sigma 5 FORTRAN subroutine for the polynomial of degree 288