REPORT:

Author:
***DooWhan Choi** is a member of the technical staff in the Digital Transmission Facilities Department of AT&T Bell Laboratories in Ward Hill, Massachusetts. Mr. Choi joined AT&T in 1984 and is responsible for synchronous digital transmission systems and cross-connect systems for metropolitan areas. He has a B.S. and an M.S. in electronic engineering from Seoul National University, Korea, and a Ph.D. in electrical engineering from the University of Texas at Austin.*

PARALLEL SCRAMBLING TECHNIQUES FOR DIGITAL MULTIPLEXERS

Introduction

Scrambling of a digital bit stream is required in order to reduce interference (by randomizing a bit stream), to provide a high transition rate, and to suppress static pattern-dependent jitter.[1,2] Two fundamental scrambling techniques, self- and frame-synchronous scrambling, are widely used in transmission systems. Self-synchronous scrambling has an error multiplication effect and may not work properly for consecutive input of zeros and ones.[3] For a certain periodic input, the scrambled sequence has a periodicity the same as that of the input.[1]

For frame-synchronous scrambling, the insertion or deletion of a bit in a bit stream garbles a whole descrambled sequence until the next frame pulses. The scrambled sequence is static random. That is, when some portion of the input is fixed, the corresponding scrambled sequence is also fixed. This static randomness is a drawback. For example, when searching for frame synchronization, the scrambled bit stream of a fixed or a slowly varying pattern may resemble the framing pattern.

When employing a scrambler in a multiplexer that interleaves several tributaries, it is better to put the scrambler at the multiplexed bit stream in order to guarantee statistically reliable scrambling. For low-speed operation, there is no problem in scrambling at the multiplexed rate. For high-speed operation, however, the multiplexed rate is sometimes too high in frequency for an economical implementation. (Consider an optical transmission system operating at the rate of several hundred megabits per second.)

This article describes parallel scrambling techniques that operate at the tributary rate with minimal hardware overhead, but have the same effect as the serial scrambling at the multiplexed rate when the prescrambled tributaries are interleaved. The article uses a simple example to explain parallel frame-synchronous scrambling. Some properties of maximum-length shift register sequences are used to generalize this parallel frame-synchronous scrambling technique. The parallel scrambling technique is also applied to a self-synchronous scrambling.

Parallel Frame-Synchronous Scrambling—Method I

Consider a multiplexer that interleaves three tributaries bit by bit and scrambles the multiplexed bit stream frame synchronously with the sequence of a 5-stage pseudo-random noise (PN) sequence generator (Figure 1). (In general, a PN sequence should satisfy three randomness criteria: the equal-density criterion, the run-length criterion, and the two-valued autocorrelation criterion.[4,5]) Because the shift registers in Figure 1 satisfy the recurrence relation

$$a_n = a_{n-3} + a_{n-5} \qquad (1)$$

the characteristic polynomial of the PN sequence generator is given by[4]

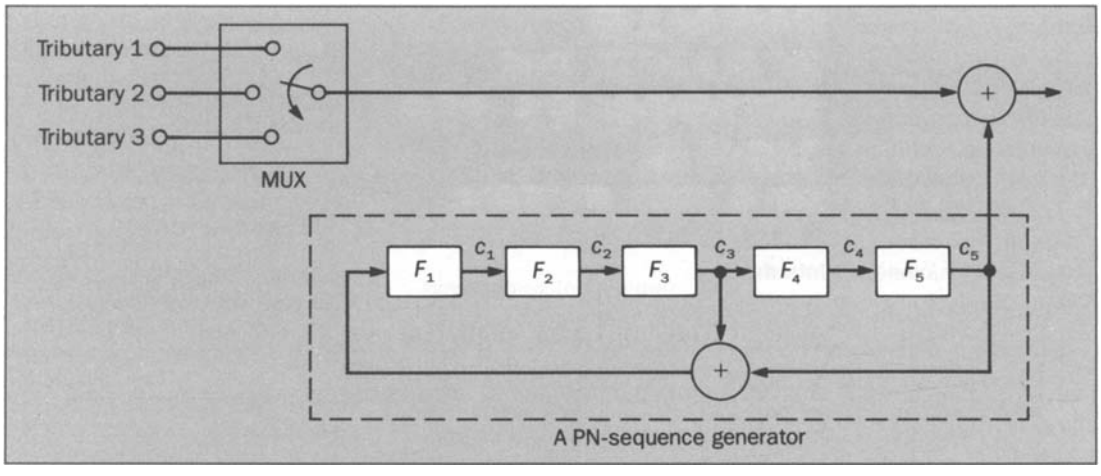

Figure 1. A serial frame-synchronous scrambler.

$$f(x) = x^5 + x^3 + 1 \qquad (2)$$

This characteristic polynomial is primitive,[4,5] and thus guarantees the maximum length of $31(=2^5 - 1)$.

The matrix representation of this PN sequence generator is[1]

$$S_{n+1} = TS_n \qquad (3)$$

The S_n here is a column vector and represents the nth state of the shift registers in the PN sequence generator

$$S_n^T = [c_1, c_2, c_3, c_4, c_5]$$

where c_i is the content of the shift register F_i in Figure 1. If the nth state is explicitly required, then c_i will be represented by $c_{i,n}$ where the second subscript n denotes the nth state. If not, just c_i is used for simplicity of notation. A state transition matrix T is given by

$$T = \begin{bmatrix} 00101 \\ 10000 \\ 01000 \\ 00100 \\ 00010 \end{bmatrix} \qquad (4)$$

From a current state and the state transition matrix T, we can predict the next state by Equation (3).

To have a parallel scrambler that operates on the tributaries in parallel, it is necessary to have three parallel sequences at the tributary rate from a PN sequence generator. To see how these would be derived, first note, in Figure 1, that the current output of the PN sequence generator is the content of the shift register F_5, the next is that of F_4, and the next is that of F_3. These bits from F_5, F_4, and F_3 would be the first bits of each of the three parallel sequences (see Figure 2). Now we

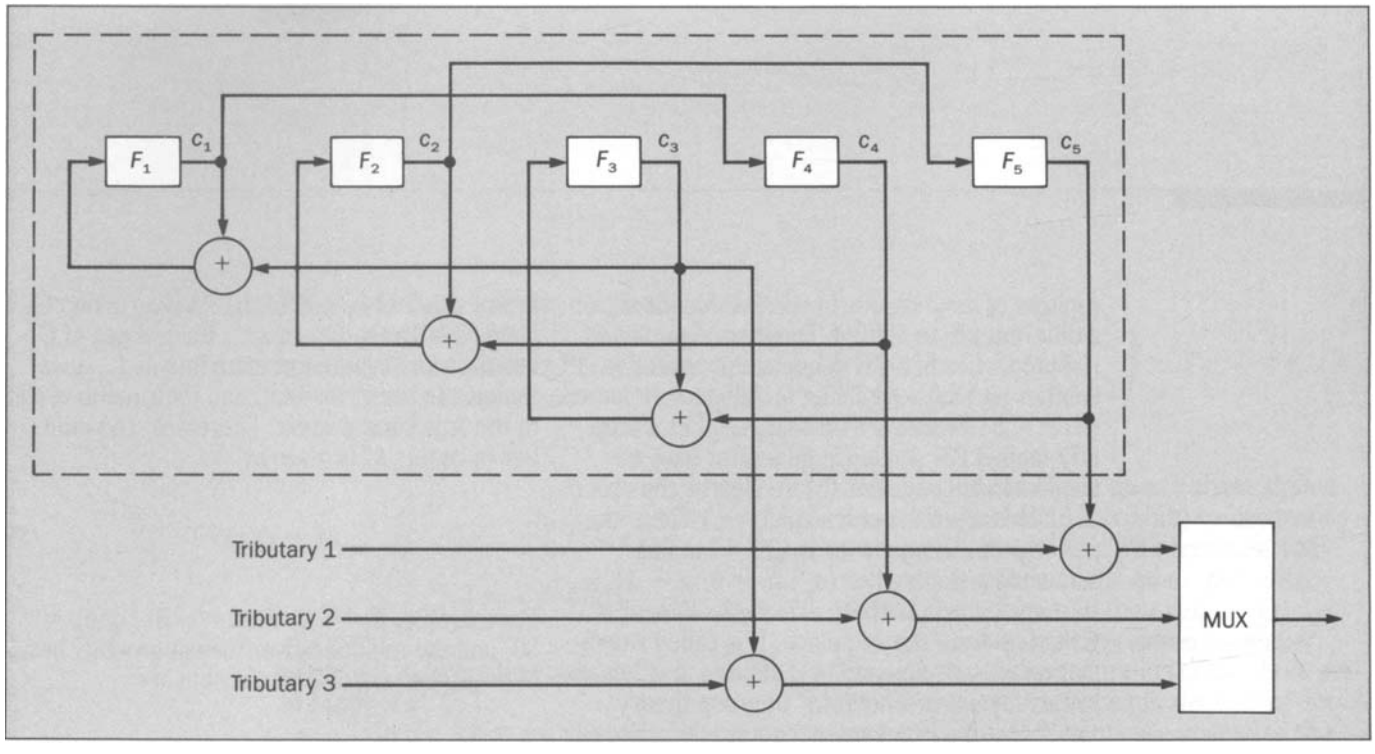

Figure 2. A parallel frame-synchronous scrambler—Method I.

need to consider how the 3-bit shift of the original sequence at the multiplexed rate can be equivalently performed by just one shift pulse at the tributary rate.[6]

At each shift pulse of the multiplexed rate, the original PN sequence generator moves to the next state. Three shift pulses later, the state becomes

$$S_{n+3} = T^3 S_n \tag{5}$$

Therefore, an intelligent interconnection of the five shift registers that satisfies the state transition matrix T^3 allows us to move from the S_n state to the S_{n+3} state by just one shift pulse. Calculating T^3 yields

$$T^3 = \begin{bmatrix} 10100 \\ 01010 \\ 00101 \\ 10000 \\ 01000 \end{bmatrix} \tag{6}$$

This T^3 matrix shows that c_1 of the S_{n+3} state is the modulo-2 sum of c_1 and c_3 of the S_n state, etc. The interconnection of five shift registers having T^3 as a state transition matrix is shown in Figure 2. The parallel scrambling is realized by adding (modulo-2) the output sequences of F_5, F_4, and F_3 to the incoming tributaries 1, 2, and 3, respectively. When multiplexed, this scrambled (in parallel) bit stream is the same as the serially scrambled bit stream of Figure 1.

In general, with a given state transition matrix T of an r-stage PN sequence generator, the computation of T^m is sufficient to implement a parallel scrambling for m tributaries.

Parallel Frame-Synchronous Scrambling—Method II

A PN sequence has two interesting combinatorial properties. One is that a proper decimation of a PN sequence becomes also a PN sequence. The other is that a modulo-2 addition of a PN sequence with phase-shifted

replicas of itself results in another replica with a different phase shift.[4,5] These two combinatorial properties of a PN sequence provide another parallel scrambling technique.

Decimation of a PN Sequence. Let a properly tapped PN sequence generator have r stages of shift register. (More clearly, the order of its characteristic polynomial is r.) Then, the period p of this sequence is $(2^r - 1)$. The sequence is denoted as $\{a_n ; n = 0, p - 1\} = \{a_0, a_1, \ldots, a_{p-1}\}$. When every qth element is picked up from the sequence, it is called a decimation of the sequence by q. The "Rearrangement Theorem" of group theory states that if q is any integer relatively prime to p, then the numbers $q, 2q, 3q, \ldots, pq$ (modulo p) are the same as $1, 2, 3, \ldots, p$, except for the order in which they occur.[7] Therefore, the decimation of the original sequence by q, i.e., $\{a_0, a_q, a_{2q}, \ldots, a_{(p-1)q}\}$, is simply a rearrangement of the original sequence $\{a_0, a_1, a_2, \ldots, a_{p-1}\}$.

There are $\phi(p)$ numbers $q, 0<q<p$, which are relatively prime to p, where $\phi(p)$ is the Euler function.[4,5,7] The $\phi(p)$ numbers q form an Abelian group $G = \{q_1, q_2, \ldots, q_{\phi(p)}\}$ with respect to modulo-p multiplication. For a Mersenne prime $p = 2^r - 1$, $\phi(p) = 2^r - 2$, and its Abelian group G is given by[4,9]

$$G = \{q_1, q_2, \ldots, q_{\phi(p)}\} = \{1, 2, 3, \ldots, 2^r - 2\}$$

In the multiplicative Abelian group, there always exists a subgroup C_1 with r elements

$$C_1 = \{1, 2, 4, 8, \ldots, 2^{r-1}\} \tag{7}$$

which is called the multiplexer subgroup.[4,9] This multiplexer subgroup, when multiplexed by any other elements of the Abelian group G, yields cosets $C_2, C_3, \ldots, C_M$. Each coset of C_1 has the same number of elements as C_1 does; the cosets are all distinct, and their union is all of the Abelian group G.[8] Therefore, the number of cosets M is given by

$$M = \phi(p)/r \tag{8}$$

For example, if $r = 5$, then $p = 31$, $\phi(p) = 30$, and the multiplicative Abelian group G has $6(= \phi(p)/r)$ cosets. The cosets are

$$\begin{aligned} C_1 &= \{1, 2, 4, 8, 16\} \\ C_2 &= \{3, 6, 12, 24, 17\} \\ C_3 &= \{9, 18, 5, 10, 20\} \\ C_4 &= \{27, 23, 15, 30, 29\} \\ C_5 &= \{19, 7, 14, 28, 25\} \\ C_6 &= \{26, 21, 11, 22, 13\} \end{aligned} \tag{9}$$

It is well known[4] that if q's are elements of the Abelian group G (i.e., q is relatively prime to p), then the decimation of a PN sequence by q is again a PN sequence with the same period. If q_i and q_j belong to the same coset, then the two decimated sequences, one by q_i and the other by q_j, differ by no more than a phase shift. The number of distinct characteristic polynomials that generate the maximal-length PN sequences from r-stage shift registers are also given by $\phi(p)/r$. There is one-to-one correspondence between the characteristic polynomials and the cosets. From this relation, all $\phi(p)/r$ PN sequences, provided by distinct characteristic polynomials of r-stage shift registers, can be obtained from any one of these

PN sequences by suitable decimations. Also, all the $\phi(p)$/r characteristic polynomials corresponding to each distinct PN sequence are obtained from any given characteristic polynomial $f_1(x)$ by[4]

$$f_{k+1}(x) = \prod_{i=1}^{q} f_k(\omega^i x^{1/q}), \quad 1 \leq k \leq \phi(p)/r \tag{10}$$

where q is a decimation step and ω is a primitive root of $x^q = 1$.

In the above analysis, the decimation of a PN sequence by an element q of the multiplicative Abelian group G is studied. If $q \in G$ and $q \notin C_1$, the decimation yields another PN sequence of the same period. If $q \in C_1$, then the decimated sequence is just a phase-shifted replica of the original sequence. There is another way of obtaining a phase-shifted sequence, other than decimation, by using an Abelian group property of a PN sequence.

Abelian Group Property of a PN Sequence. Let $A_1 = \{a_0, a_1, \ldots, a_{p-1}\}$ be a PN sequence with a period p. Let $A_i = \{a_{i-1}, a_i, \ldots, a_{p-1}, a_0, a_1, \ldots, a_{i-2}\}$. That is, A_i is a phase-shifted (leading) sequence of A_1 by $(i - 1)$. Also, let $A_0 = \{0, 0, \ldots, 0\}$. Then, the sequences A_0, $A_1, \ldots, A_p$ form an Abelian group H with respect to termwise modulo-2 addition, where the identity element is A_0 and the inverse element of A_i is A_i itself.[4,7] The importance of this Abelian group property is that any termwise modulo-2 sum of A_i's is also the phase-shifted sequence of A_i. Therefore, we can obtain the total p phase-shifted sequences of A_1 (including itself) from r sequences $A_1, A_2, \ldots, A_r$ by termwise addition since

$$\binom{r}{1} + \binom{r}{2} + \cdots + \binom{r}{r} = 2^r - 1 = p \tag{11}$$

Note that we can obtain up to r phase-shifted sequences (including zero shift) by decimation with $q_i \in C_1$, $1 \leq i \leq r$. With termwise addition, we can obtain all the other $(p\text{-}r)$ phase-shifted sequences from the r sequences.

Parallel Scrambling Pattern Generation. Consider again the generation of the three parallel sequences that are equivalent to the serial sequence of the PN sequence generator shown in Figure 1. What we want are the three decimated-by-3 (i.e., decimated by the number of tributaries) parallel sequences from the shift registers F_5, F_4, and F_3, respectively. Let us call the three decimated sequences SEQ_5, SEQ_4, and SEQ_3. The relations among the original output sequences of F_5, F_4, and F_3, before decimation, are phase shifts. The output sequence of F_3 leads that of F_4 by one bit and leads that of F_5 by two bits. After decimation, they are still phase shifted relative to each other, but the amount of phase shift is different.

Two steps are required to generate the three decimated parallel sequences. The first step is to find out the characteristic polynomial that provides the decimated sequence SEQ_5. The second step is to measure the phase shifts among SEQ_3, SEQ_4, and SEQ_5 in the decimated sequence, and then generate SEQ_3 and SEQ_4. Generation of SEQ_3 and SEQ_4 is accomplished by termwise addition of SEQ_5 and up to four phase-leading (in general, up to r-1) sequences of SEQ_5, which are directly obtained from the shift registers of the decimated PN

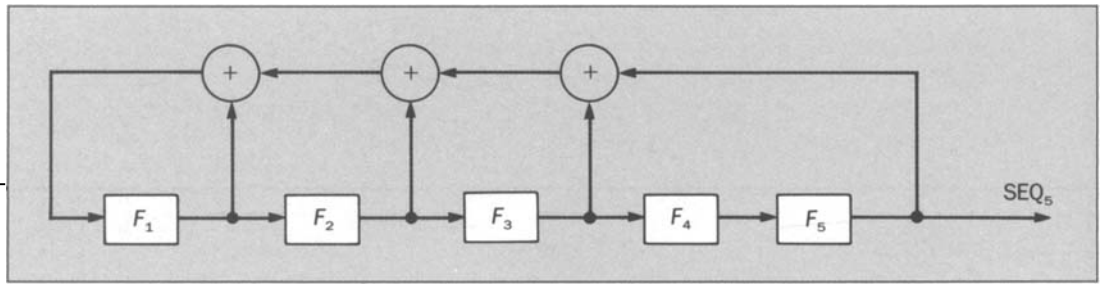

Figure 3. A PN sequence generator with $f_2(x) = x^5 + x^3 + x^2 + x + 1$

sequence generator.

We decimate the original sequence by 3. Because the number 3 is an element of coset C_2 [see Equation (9)], the decimated sequence is not just a phase-shifted sequence but another PN sequence. The new characteristic polynomial $f_2(x)$ corresponding to the coset C_2 is obtained from the original characteristic polynomial by Equation (10);

$$f_1(x) = x^5 + x^3 + 1$$
$$f_2(x) = f_1(\omega x^{1/3}) f_1(\omega^2 x^{1/3}) f_1(x^{1/3})$$

where ω is a primitive root of $x^3 = 1$. That is, $\omega^3 = 1$, $\omega^2 + \omega + 1 = 0$. This equation yields

$$f_2(x) = x^5 + x^3 + x^2 + x + 1$$

Therefore, the decimated sequence SEQ_5 is obtained by the shift register F_5 of Figure 3. The phase difference τ between SEQ_i and SEQ_{i+1} is calculated by

$$m \cdot \tau \ (modulo\ p) = 1 \qquad (12)$$

where m is the decimation step (i.e., the number of tributaries) and p is the period of the PN sequence. When $m = 3$ and $p = 31$, $\tau = 21$. Therefore, SEQ_4 and SEQ_3 lead SEQ_5 by 21 and 11 bits, respectively. Now, let us consider how to generate SEQ_3 and SEQ_4 by termwise addition of SEQ_5 and the sequences with phase leads of up to 4 bits.

The arrangement of shift registers in Figure 3 satisfies the recurrence relation

$$a_n = a_{n-1} + a_{n-2} + a_{n-3} + a_{n-5}$$

Using a unit delay operator D, we can rewrite this recurrence relation as

$$(D^5 + D^3 + D^2 + D + 1)\{a_n\} = 0$$

Therefore, the delay operator D always satisfies the characteristic polynomial[4]

$$f_2(D) = D^5 + D^3 + D^2 + D + 1 = 0 \qquad (13)$$

A unit lead operator L also satisfies

$$f_2(1/L) = 0 \qquad (14)$$

For the case of Figure 3, the lead operator L satisfies

$$L^5 + L^4 + L^3 + L^2 + 1 = 0 \qquad (15)$$

If we denote SEQ_5 as $\{b_n\}$, then SEQ_4 and SEQ_3 can be expressed as $L^{21}\{b_n\}$ and $L^{11}\{b_n\}$, respectively; since SEQ_4 and SEQ_3 lead SEQ_5 by 21 and 11 bits, respectively. SEQ_3 and SEQ_4 can also be reduced to

$$SEQ_4 = L^{21}\{b_n\} = (L^2 + L)\{b_n\} \qquad (16)$$
$$SEQ_3 = L^{11}\{b_n\} = (L^4 + L^2)\{b_n\}$$

due to Equation (15). Using these relations, the parallel scrambling circuitry can be implemented as shown in Figure 4.

Comparison of Methods I and II. When comparing Figure 4 of Method II with Figure 2 of Method I, both implementing the same function, Method I looks more desirable because

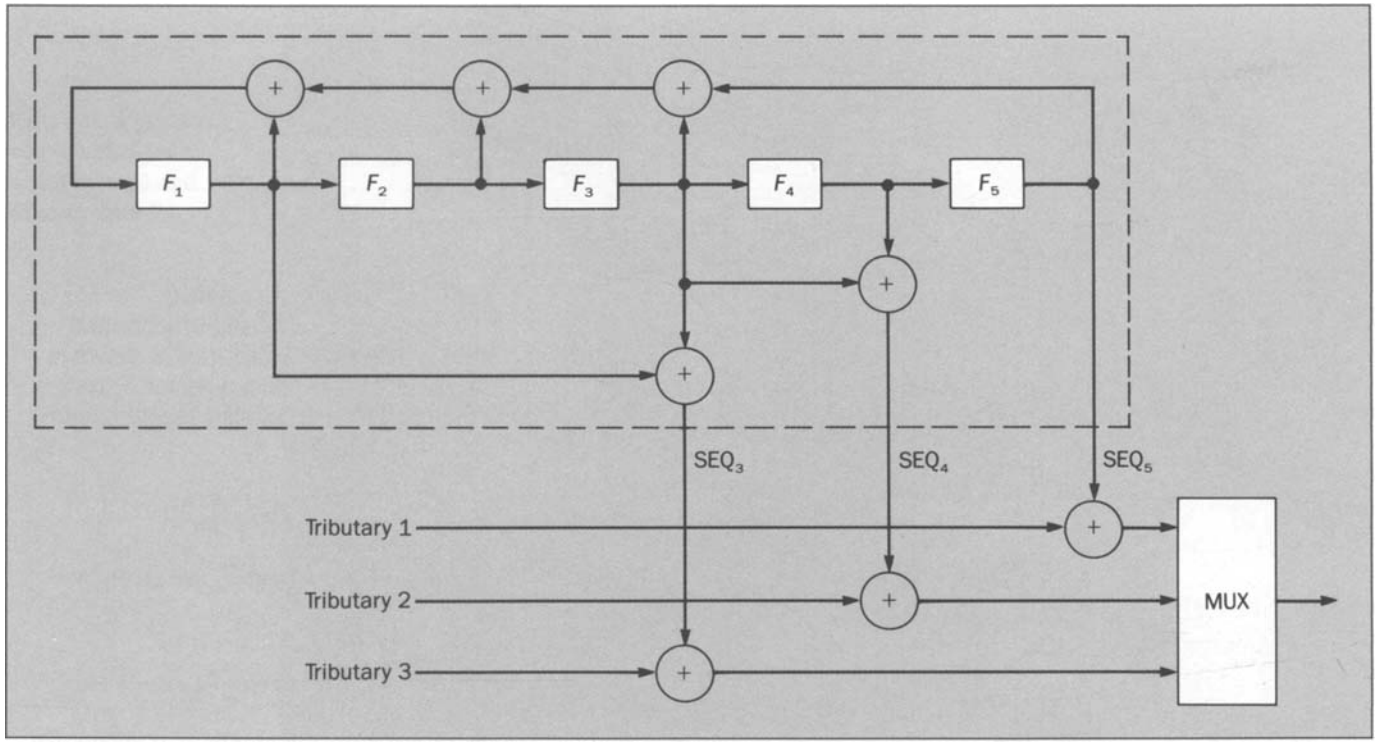

Figure 4. A parallel frame-synchronous scrambler—Method II.

Figure 2 requires a fewer number of exclusive-OR (XOR) gates than does Figure 4. However, this is not generally true because the number of XOR gates depends on the selection of an original characteristic polynomial that generates a serial PN sequence and the number of tributaries to be multiplexed. For example, if we are going to use the first technique for a multiplex with four tributaries, we need two XOR gates for a feedback loop because

$$T^4 = \begin{bmatrix} 01101 \\ 10100 \\ 01010 \\ 00101 \\ 10000 \end{bmatrix}$$

For the second technique, if we select the original characteristic polynomial as

$$f_1(x) = x^5 + x^4 + x^2 + x + 1$$

which also generates a PN sequence, then the characteristic polynomial for the decimated-by-3 sequence becomes very simple,

$$f_2(x) = x^5 + x^3 + 1$$

Therefore, the parallel sequence generator of the second technique needs only one XOR gate in the feedback loop compared to three in Figure 4. If we select the original characteristic polynomial intelligently, the second parallel scrambling technique becomes quite simple. Also, a modular sequence generating method[5] can sometimes be used to reduce XOR gates in a feedback loop.

Parallel Self-Synchronous Scrambling

For frame-synchronous scrambling, a PN sequence generator provides a code sequence to scramble an incoming bit stream. When the characteristic polynomial and the

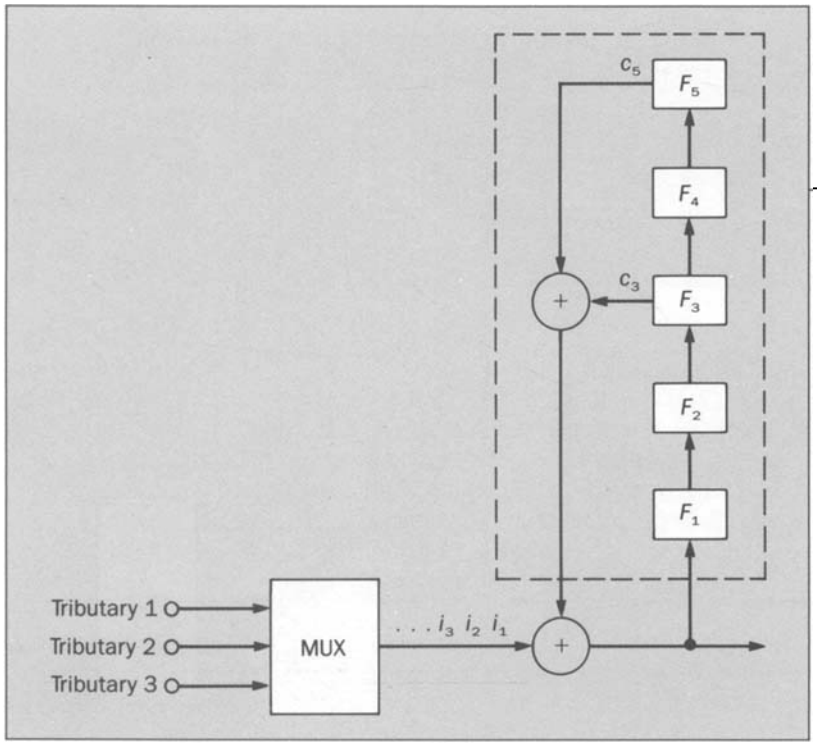

Figure 5. A serial self-synchronous scrambler.

present state of a PN sequence generator are given, the whole forthcoming PN sequence can be predicted independently of an incoming bit stream. This does not apply, however, to self-synchronous scrambling because the next state of a self-synchronous scrambler depends on the incoming bit stream itself. The number of predictable next states is limited to the number of forthcoming input bits whose state is already known.

Given a random input bit stream, for a serial self-synchronous scrambler, we know only one forthcoming bit. For a parallel situation, we know m forthcoming bits at a time where m is the number of tributaries to be multiplexed. Correspondingly, the next m states of the scrambler can be predicted. However, the nice mathematics of the second frame-synchronous scrambling technique does not apply for self-synchronous scrambling because the next states are input dependent and, hence, neither periodic nor predictable for more than the next m states. Therefore, we should restrict ourselves to the first technique for parallel self-synchronous scrambling.

A 5-stage serial self-synchronous scrambler with three tributaries is shown in Figure 5. We let the column vector S_n represent the current state of shift registers and incoming data

$$S_n^T = [I_n^T, C_n^T]$$

The column vector I_n and C_n are given by

$$I_n^T = [i_m, i_{m-1}, \ldots, i_1]$$
$$C_n^T = [c_1, c_2, \ldots, c_{max\,(m,r)}]$$

where m and r are the numbers of tributaries and shift register stages, respectively. For the case of Figure 5, S_n^T turns out to be

$$S_n^T = [i_3, i_2, i_1, c_1, c_2, c_3, c_4, c_5]$$

where i_1 is the current input, i_2 is the next input, and so forth. Because we know the forthcoming three bits, we can predict up to the third state by using the state transition matrix T_s:

$$S_{n+k} = T_s^k S_n, \quad 0 < k \leq 3$$

where T_s is given by

$$T_s = \left[\begin{array}{c|c} 000 & 00000 \\ 100 & 00000 \\ 010 & 00000 \\ \hline 001 & 00101 \\ 000 & 10000 \\ 000 & 01000 \\ 000 & 00100 \\ 000 & 00010 \end{array}\right]$$

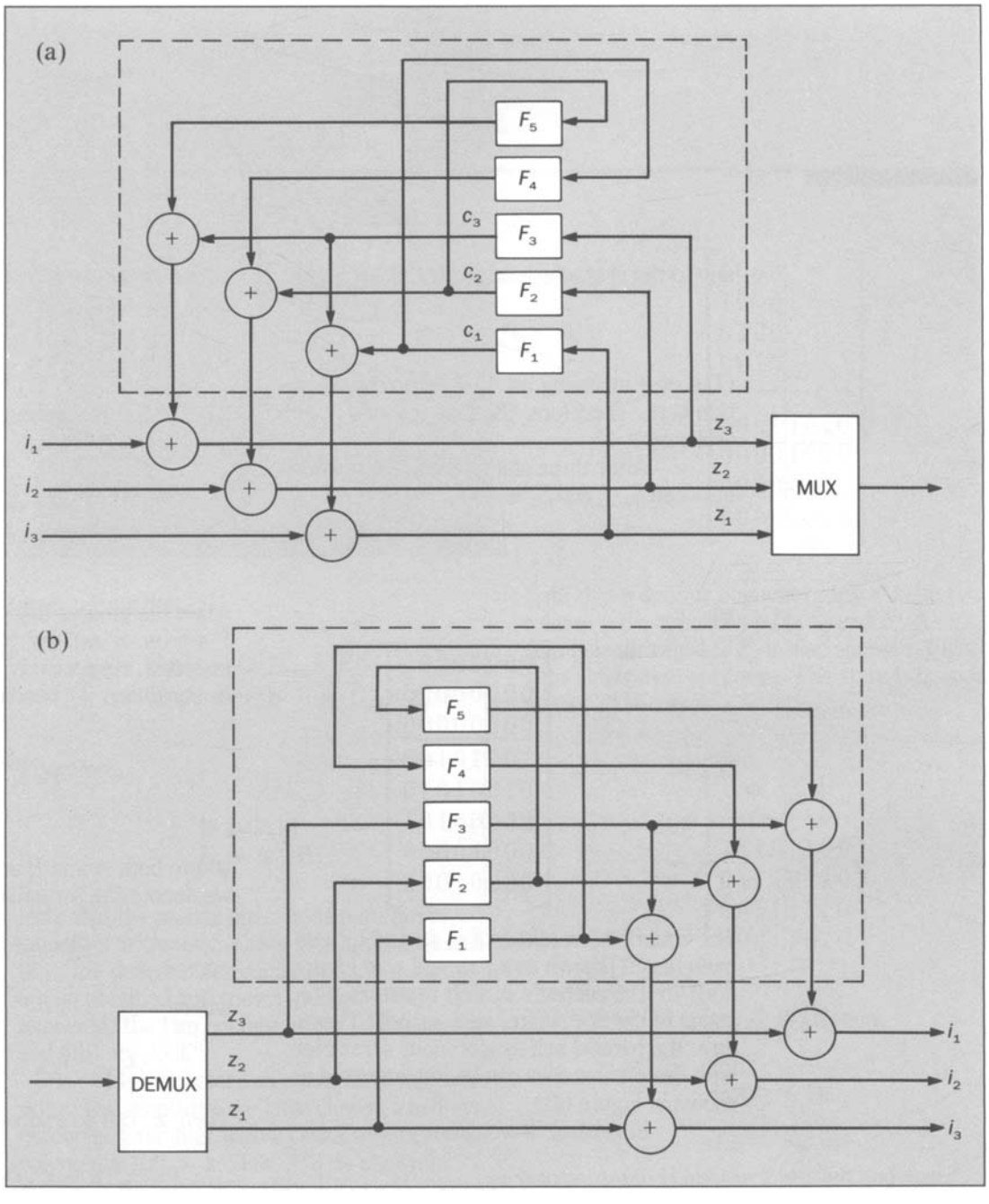

Figure 6. A parallel self-synchronous scrambler (a) and descrambler (b).

due to the characteristic polynomial of Figure 5

$$f(x) = x^5 + x^3 + 1$$

(The next incoming bit, i_4, is of no consideration here. Therefore, the first row of T_s can be arbitrary.)

After three shift pulses, the relation between S_{n+3} and S_n is

$$S_{n+3} = T_s^3 S_n$$

where

$$T_s^3 = \left[\begin{array}{ccc|ccccc} 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ \hline 1&0&0&1&0&1&0&0 \\ 0&1&0&0&1&0&1&0 \\ 0&0&1&0&0&1&0&1 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \end{array}\right]$$

The scrambled versions of i_1, i_2, and i_3 to be multiplexed [shown as z_3, z_2, and z_1 in Figure 6(a)] are the contents of shift registers c_3, c_2, and c_1 of the S_{n+3} state, respectively. Therefore, the parallel self-synchronous scrambler with three tributaries can be implemented as shown in Figure 6(a).

In general, if we define

$$\begin{aligned} S_n^T &= [I_n^T, C_n^T] \\ &= [i_m, i_{m-1}, \ldots, i_1, c_1, \\ &\quad c_2, \ldots, c_{max\,(m,r)}] \end{aligned} \tag{17}$$

where m and r are the number of tributaries and shift register stages, respectively. Then, with a given state transition matrix T_s, the next mth state is given by

$$S_{n+m} = T_s^m S_n \tag{18}$$

The T_s^m has the general form

$$T_s^m = \left[\begin{array}{c|c} \Phi & \Phi \\ \hline \mathrm{A} & \mathrm{B} \\ \hline \Phi & \Gamma \end{array}\right] \tag{19}$$

when r is greater than m, and where A, B, and Γ are $(m \times m)$, $(m \times r)$, and $[(r-m) \times r]$ matrixes, respectively. When m is greater than or equal to r, T_s^m becomes

$$T_s^m = \left[\begin{array}{c|c} \Phi & \Phi \\ \hline \mathrm{A} & \mathrm{B} \end{array}\right] \tag{20}$$

where both A and B are $(m \times m)$ matrixes. If we denote the scrambled parallel sequence as

$$Z_n^T = [z_1, z_2, \ldots, z_m]$$

where

$$z_{i,n} = c_{i,n+m} \qquad 1 \leq i \leq m$$

Then, Z_n can be expressed as

$$Z_n = \mathrm{A}I_n + \mathrm{B}C_n \tag{21}$$

Descrambling entails determining the two matrixes Θ and E such that the descrambled parallel sequence O_n satisfies

$$O_n = \Theta Z_n + \mathrm{E}C_n = I_n \tag{22}$$

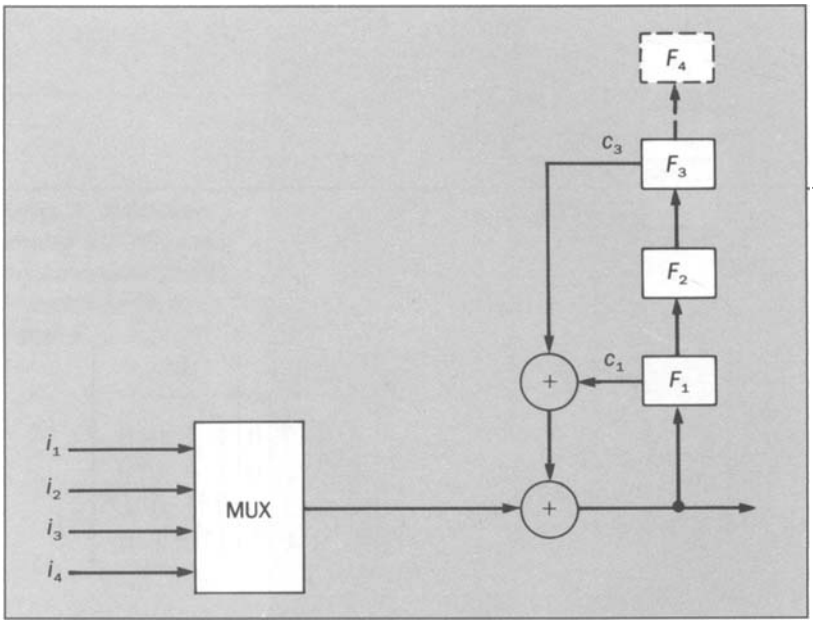

Figure 7. A serial self-synchronous scrambler with $m > r$.

Due to Equation (21)

$$O_n = \Theta A I_n + \Theta B C_n + E C_n$$
$$= \Theta A I_n + (\Theta B + E) C_n = I_n$$

Therefore,

$$\Theta = A^{-1} \qquad (23)$$
$$E = A^{-1}B$$

Note that the matrix operations here are in modulo-2 arithmetic. Using Equations (22) and (23), the descrambler configuration for Figure 6(a) is obtained and shown in Figure 6(b). As expected, the two configurations of Figure 6(a) and 6(b) are dual.

Let's consider an example of the case $m>r$. For simplicity, we have chosen a self-synchronous scrambler with a characteristic polynomial $f(x) = x^3 + x + 1$ as shown in Figure 7. Because $m>r$, the nth state vector S_n is

$$S_n^T = [i_4, i_3, i_2, i_1, c_1, c_2, c_3, c_4]$$

and the state transition matrix T_s becomes

$$T_s = \left[\begin{array}{cc} 0000 & \\ 1000 & \phi \\ 0100 & \\ 0010 & \\ \hline 0001 & 1010 \\ 0000 & 1000 \\ 0000 & 0100 \\ 0000 & 0010 \end{array}\right]$$

Note that a dummy element c_4 and a dummy column for c_4 are included in S_n and T_s. A dummy shift register F_4 is also shown in Figure 7 for illustration purposes. The state transition matrix T_s^4 becomes

$$T_s^4 = \left[\begin{array}{cc} \phi & \phi \\ \hline 1110 & 1100 \\ 0111 & 0110 \\ 0011 & 1110 \\ 0001 & 1010 \end{array}\right] \qquad (24)$$

The scrambled output Z_n is given by

$$Z_n = A I_n + B C_n \qquad (25)$$

where A and B are the lower left and lower right (4×4) submatrixes of T_s^4, respectively. The descrambled output O_n is, then, obtained by

$$O_n = A^{-1} Z_n + A^{-1} B C_n \qquad (26)$$

where

$$A^{-1} = \begin{bmatrix} 1101 \\ 0110 \\ 0011 \\ 0001 \end{bmatrix} \qquad (27)$$

$$A^{-1}B = \begin{bmatrix} 0000 \\ 1000 \\ 0100 \\ 1010 \end{bmatrix} \qquad (28)$$

The implementation of a scrambler given by Equations (24) and (25) is very complex. By carefully examining the matrix T_s^4 of Equation (24), we recognize that the scrambler implementation can be simplified. The state transition matrix T_s^4 of Equation (24) can be rewritten as

$$T_s^4 = \left[\begin{array}{cc|cc} \phi & & \phi & \\ \hline 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right] \left[\begin{array}{cc} & \\ \hline 1 & 0 \\ 1 & 0 \\ 0 & L^m \\ 1 & 0 \end{array}\right] \left[\begin{array}{cc} & \\ \hline 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{array}\right] \left[\begin{array}{cc} & \\ \hline 0 & 0 \\ 0 & L^m \\ 0 & L^m \\ 0 & 0 \end{array}\right]$$

where L is a unit lead operator. That is,

$$L^m c_{i,n} = c_{i,n+m} = z_{i,n}$$

Again

$$T_s^4 = \left[\begin{array}{cc|cc} \phi & & \phi & \\ \hline 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right] \left[\begin{array}{cc} & \\ \hline 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{array}\right] \left[\begin{array}{cc} & \\ \hline 0 & 0 \\ L^m & 0 \\ 0 & L^m \\ 1 & 0 \end{array}\right] \left[\begin{array}{cc} & \\ \hline 0 & 0 \\ 0 & L^m \\ 0 & L^m \\ 0 & 0 \end{array}\right]$$

and again

$$T_s^4 = \left[\begin{array}{cc|cc} \phi & & \phi & \\ \hline 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right] \left[\begin{array}{cc} & \\ \hline 0 & L^m \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{array}\right] \left[\begin{array}{cc} & \\ \hline 0 & L^m \\ L^m & 0 \\ 0 & L^m \\ 1 & 0 \end{array}\right] \left[\begin{array}{cc} & \\ \hline 0 & L^m \\ 0 & L^m \\ 0 & L^m \\ 0 & 0 \end{array}\right] \qquad (29)$$

Therefore, a new A becomes a unity matrix and a new B becomes

$$B = \begin{bmatrix} 0 & L^m & 0 & L^m \\ 1 & 0 & L^m & 0 \\ 0 & 1 & 0 & L^m \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Here, we notice that the lead operators in the new B match with the 1s above the diagonal in A^{-1} of Equation (27). The implementation of a simplified parallel scrambler/descrambler is shown in Figure 8. As seen in this implementation, the parallel scrambler/descrambler pair is again dual as expected.

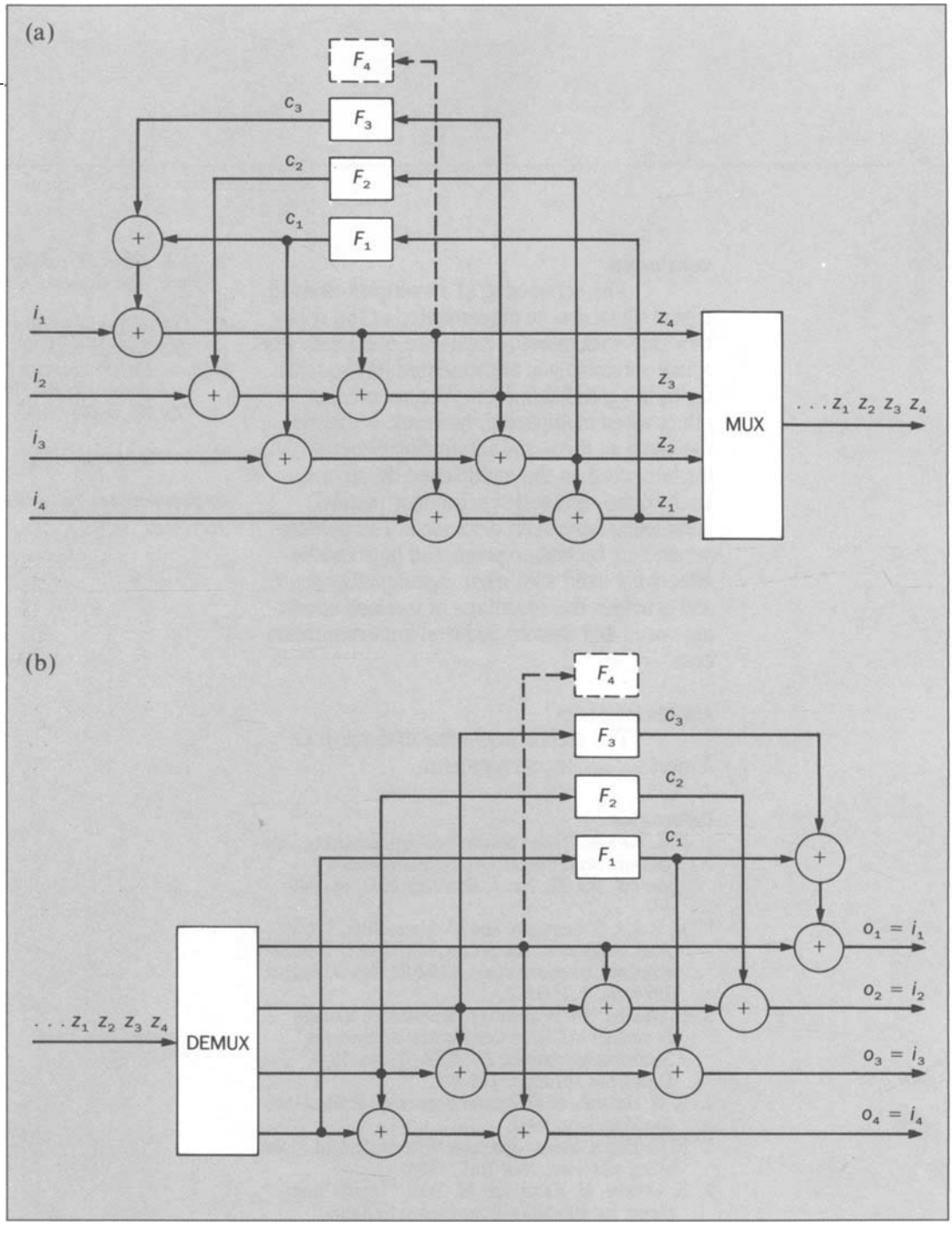

Figure 8. Simplified parallel self-synchronous scrambler (a) and descrambler (b) for Figure 7.

Conclusion

The scrambling of a multiplex-derived digital signal can be implemented at the tributary rate with minimal hardware overhead. The actual scrambling is implemented on the tributaries in a parallel fashion. The scrambling effect when multiplexed, however, is exactly the same as the serial scrambling directly implemented on the multiplexed bit stream. Both frame- and self-synchronous parallel scrambling have been discussed. The parallel scrambling technique presented here can be effectively used with most digital multiplexers, and provides the advantage of reduced operating speed and thereby reduced implementation cost.

Acknowledgment

The author would like to thank J. O. Azaret for his encouragement.

References

1. J. E. Savage, "Some Simple Self-Synchronizing Digital Data Scramblers," *Bell System Technical Journal,* Vol. 46, No. 1, February 1967, pp. 449-487.
2. H. Kasai, S. Senmoto, and M. Matsushita, "PCM Jitter Suppression by Scrambling," *IEEE Transactions on Communications,* COM-22, No. 8, August 1974, pp. 1114-1122.
3. H. Muller, "Bit Sequence Independence through Scrambler in Digital Communication Systems," *Nachrichtentechnische Z—NTZ 27,* No. 12, December 1974, pp. 475-479.
4. S. W. Golomb, *Shift Register Sequences,* Holden-Day, San Francisco, 1967.
5. R. C. Dixon, *Spread Spectrum Systems,* 2nd ed., John Wiley and Sons, New York, 1984.
6. K. Ohtake, H. Kasai, and M. Taka, "Digital Multiplexer for 400-Mb/s Transmission Systems," *Review of the Electrical Communication Laboratories,* NTT, Japan, Vol. 24, No. 9-10, September-October 1976, pp. 725-736.
7. A. A. Albert, *Fundamental Concepts of Higher Algebra,* The University of Chicago Press, Chicago, 1956.
8. V. Pless, *Introduction to the Theory of Error-Correcting Codes,* John Wiley and Sons, New York, 1982.
9. R. E. Blahut, "Algebraic Fields, Signal Processing, and Error Control," *Proceedings of the IEEE,* Vol. 73, No. 5, May 1985.

(Manuscript received August 25, 1986)