

Chapter 3: The Matrix Approach

Chapter Contents

3.1 Review of earlier Chapters

3.2 Matrix Solution of Figure 2.

The A matrix.

3.3 Matrix Solution of Figure 1.

The B matrix.

3.4 Shift Register Generators Revisited

Fact 1: The entire state vector of a primitive polynomial shift register generator satisfies the difference equation (2.3.2). Thus, each component of the state vector satisfies (2.3.2) as well, and each component is the state of a particular register.

Fact 2: The state of each individual register in a primitive polynomial shift register generator cycles through the characteristic sequence.

3.5 The Output of a Scrambler

Fact 2: In the presence of an input stream $\{i_n\}$, and with initial state vector $\mathbf{q}_0 = \mathbf{0}$, the state of any register of a primitive polynomial scrambler equals the "dot product" of a characteristic sequence multiplied by the (time reversed) input sequence.

A Lemma:

Review of Leeper 1973.

3.6 The Spectral Density of a Scrambler Output

" Barring anomolous input sequences, the spectral density of the output of an operating scrambler is essentially the same as the spectral density of a random white NRZ signal."

Interpretation of the Spectrum;

Power Distribution by Hump

3.7 The NRZI Mini-Scrambler.

Fact 1: The output of the above mini-scrambler changes state each time a one is encountered in the input data stream, and it holds its state each time a 0 is encountered in the input stream.

Definition: An *NRZI line code* is what emerges from the mini-scrambler when you input a NRZ linecode.

Scrambler Summary!

3.8. Matrix analysis of the polynomial multiplier (de-scrambler) of Figure 7 .

3.9 Matrix analysis of the standard polynomial multiplier of Figure 8 .

3.10. Proof that a Descrambler really descrambles the output of a Scrambler.

Figure 3.10.1: Proposed scrambler structure for serial digital communication .

Proof in the Z Domain;

Proof in the Time Domain;

Conclusions

(contents continued on next page)

3.11 The Kill Sequence Problem and Strings of Zeros

Fact 1: It is possible that a k -stage scrambler will output a string of $N+k$ zeros if a string of N zeros exists in the input stream; the probability is $1/P$ where $P = 2^k - 1$.

Example 1: Suppose in D1 video.

Example 2: Ancillary data

Fact 2: Assume a scrambler is in some state **a**. Consider the k -symbol pattern that next shifts in, call this pattern **i**. After this pattern shifts in, the scrambler is in state **b**. We claim that, given any patterns **a** and **b**, there exists a unique pattern **i** which causes **a** to go to **b**.

Corollary 2: If pattern **b** is 0, there exists a unique pattern **i** which takes **a** to 0.

Definition: Such a pattern is called a *kill vector*.

Explicit formula for vector **i**.

Example: $k=9$

Discussion of the zeros problem.

Fact 3: If a k -stage scrambler has an even number of feedback taps, then if the input stream is set to all 1's, the scrambler behaves exactly as if the input stream were all 0's and there were an inverter on the final output.

Some final notes:

Appendix 3.1: Proof of Eq. (3.10.15)

Chapter 3: The Matrix Approach

This Chapter will lean heavily on Galois Chapter 8. There, it was shown that it is possible to construct an explicit matrix representation of any Galois Field. Here, as we attempt to solve directly for the output sequence (and state vector sequence) of a scrambler, we will find ourselves eyeball-to-eyeball with exactly the matters discussed in Galois Chapter 8.

3.1 Review of earlier Chapters

Because this document is so lengthy, we feel it is always worthwhile to review our current status.

In Chapter 1 we discussed the polynomial divider circuits of Figure 1 and Figure 2. Figure 1 is what we called "the scrambler style" circuit, whereas Figure 2 is the "standard circuit". We showed how these circuits both perform the division of polynomials $O(z) = I(z)/H(z)$. An attempt was made to show how each circuit carries out this task in comparison to the way a human being would carry out the long division of polynomials. The Figure 2 circuit was in some ways easier to understand since its registers always contain the current remainder, or current dividend. However, the Figure 1 circuit is intrinsically more interesting to use since we are really studying Scramblers.

Polynomial multipliers were also treated in Chapter 1, and again two circuit forms were considered, Figure 7 and Figure 8. We spent less time worrying about the multipliers because they have no feedback, only "feedforward" and are easier to understand. It is a mathematical fact that multiplication is simpler than division.

In Section 1.8 of Chapter 1, we gave a summary box which outlines the behavior of dividers and multipliers in both the Z-domain and the time domain. In the Z-domain, the equations are just the statements of polynomial division and multiplication. In the time domain, we get difference equations which relate the output stream symbols to the input stream symbols.

In Chapter 2 we attempted to characterize the behavior in time of the polynomial divider circuits with their input streams set to zero. Such circuits are called shift register generators. Of particular interest is the periodicity of the register "state vector", and the periodicity of the output stream. We showed that the use of a Galois primitive polynomial for the polynomial $h(x)$ [same as $H(z)$] resulted in things having the maximum possible periods. We made no comments as to whether this is good or bad, desirable or undesirable.

In Section 2.3 we did a rather thorough analysis of the difference equation which describes the output sequence o_n of a shift register generator, namely,

$$\sum_{j=0}^k h_j o_{n+j} = 0 \quad n = \text{any integer} \quad (2.3.2) \text{ repeated}$$

Here we were dealing with numbers in $GF(p)$, and with a shift register generator having k registers, and h_i were the feedback coefficients, also the coefficients of $h(x)$.

We showed that there are p^k solutions of the difference equation, one of which is all zeros. In the interesting case that $h(x)$ is a primitive polynomial of $GF(p^k)$, the remaining $p^k - 1$ solutions turned out to be starting time variations of *one* characteristic solution of the form $\{c, c, c, \dots\}$. Here, the sequence c has period $p^k - 1$, and c could be considered any code word of the cyclic code $(n=p^k - 1, k)$. The generator of this code, $g(x)$, is a polynomial of degree $n-k = p^k - 1 - k$ which is obtained by dividing $x^n - 1$ by $h(x)$. It seemed simplest to consider the sequence c to be the code word consisting of the $(p^k - 1 - k) + 1$ coefficients of $g(x)$, padded with $(k-1)$ zeros.

We went on to state various properties of the characteristic sequence, also known as a maximal length sequence. Along the way, various Facts from the Galois notes were quoted and used.

In Section 2.2 we used the special polynomial residue class ring representation of $GF(p^m)$ to show that the behavior of the Figure 2 divider circuit could be described by a Galois Iterator equation, which we duplicate here:

$$\beta_s = \alpha^s \beta_0 + [\alpha^{s-1} i_0 + \alpha^{s-2} i_1 + \dots + \alpha i_{s-2} + i_{s-1}] 1 \quad (2.2.10) \text{ repeated}$$

The symbol β_0 represented the starting state vector of the registers in Figure 2. Then β_s gave the state vector after s clockings of the circuit. We identified α with a certain entity $\{x\}$. As can be seen in the above, the input sequence i_n does of course play a role in the subsequent state vector β_s . We then turned off the input stream to get the simpler equation $\beta_s = \alpha^s \beta_0$ which describes the shift register generator.

Perhaps the reader noticed on the part of the writer a certain dissatisfaction with the above equation, and how it was not very much pursued, although it was declared to be important. There were several problems. First, we seemed to have the above equation apply to Figure 2, but not to Figure 1. This certainly is unsatisfying, since both circuits do the same thing, and since Figure 1 is the circuit used for a scrambler. Secondly, the equation above has a certain abstract feel to it. We have these abstract Galois elements α and β floating around, and we have a dim connection that $\alpha = \{x\}$ where $\{x\}$ is an element of some polynomial quotient ring business. But the connection to our circuits is just not very clear. On the other hand, we realize and appreciate how the abstract Galois theory has provided us with all of our most powerful claims made so far, such as those about the period of the characteristic output sequence of a shift register generator.

In the present Chapter, all these deficiencies will be remedied.

3.2 Matrix Solution of Figure 2.

We assume from here on out that $h(x)$ is monic, so that $h_k = 1$. From Section 1.4 we describe the operation of Figure 2 as

$$q_{j+1}(n+1) = d_{j+1}(n) = q_j(n) - h_j o(n) \quad j = 0, 1, 2, \dots, k-1 \quad (1.4.1) \text{ repeated}$$

As earlier, we write $f(n) = f$, and $f(n+1) = f'$. Since $h_k = 1$, $o = q_k$. And $i = q_0$. We now explicitly write out the above k equations:

$$\begin{aligned}
 q'_1 &= i - h_0 q_k \\
 q'_2 &= q_1 - h_1 q_k \\
 q'_3 &= q_2 - h_2 q_k \\
 q'_4 &= q_3 - h_3 q_k \\
 &\dots \\
 q'_{k-1} &= q_{k-2} - h_{k-2} q_k \\
 q'_k &= q_{k-1} - h_{k-1} q_k
 \end{aligned} \tag{3.2.1}$$

These equations can be trivially re-expressed in matrix form as follows. (we apologize for the vertical bars in place of proper large parentheses)

$$\begin{array}{c|c|c|c|c|c|c|c}
 q'_1 & & 0 & 0 & 0 & \dots & 0 & -h_0 \\
 q'_2 & & 1 & 0 & 0 & \dots & 0 & -h_1 \\
 q'_3 & & 0 & 1 & 0 & \dots & 0 & -h_2 \\
 q'_4 & = & 0 & 0 & 1 & \dots & 0 & -h_3 \\
 \dots & & \dots & \dots & \dots & \dots & \dots & \dots \\
 q'_{k-1} & & 0 & 0 & 0 & \dots & 0 & -h_{k-2} \\
 q'_k & & 0 & 0 & 0 & \dots & 1 & -h_{k-1}
 \end{array}
 +
 \begin{array}{c|c}
 q_1 & i \\
 q_2 & 0 \\
 q_3 & 0 \\
 q_4 & 0 \\
 \dots & \dots \\
 q_{k-1} & 0 \\
 q_k & 0
 \end{array} \tag{3.2.2}$$

Now we can write this in compact vector/matrix notation as:

$$\mathbf{q}' = \mathbf{A} \mathbf{q} + \mathbf{i} \mathbf{1} \tag{3.2.3}$$

Here $\mathbf{q}$ and $\mathbf{q}'$ are the obvious column vectors, $\mathbf{1}$ is a unit column vector, and $\mathbf{A}$ is the matrix. Now put back the time indices n which we took out above, and get rid of the primes. Put the time index as a subscript, since this notational slot is again available,

$$\mathbf{q}_{n+1} = \mathbf{A} \mathbf{q}_n + \mathbf{i}_n \mathbf{1} \tag{3.2.4}$$

This is a vector difference equation for the state vector $\mathbf{q}_n$ of Figure 2. It is easy to solve (3.2.4) by brute force, and here is the result:

$$\mathbf{q}_n = \mathbf{A}^n \mathbf{q}_0 + [\mathbf{i}_0 \mathbf{A}^{n-1} + \mathbf{i}_1 \mathbf{A}^{n-2} + \dots + \mathbf{i}_{n-2} \mathbf{A} + \mathbf{i}_{n-1} \mathbf{I}] \mathbf{1} \tag{3.2.5}$$

Observations:

(1) The matrix A has exactly the "companion" form C discussed in Galois Chapter 8, Section 8.5. This means that $h(x)$ is the characteristic polynomial of A .

(2) Therefore, from Big Fact 2 of Section 8.1, we know that $h(A) = 0$, which we can rewrite as,

$$A^k = -h_0 I - h_1 A - h_2 A^2 - \dots - h_{k-1} A^{k-1}$$

(3) Since A is a root of $h(x)$, we know that A is some element of a *matrix representation* of the Galois Field $GF(p^m)$.

(4) It follows from the Section 8.5 Big Corollary that, if $h(x)$ is a *primitive* polynomial of $GF(p^m)$, then the matrix A can be regarded as an explicit matrix representation of an abstract *primitive* element α of the Galois Field $GF(p^m)$. In this case, all non-zero elements of $GF(p^m)$ can be represented as powers of A .

(5) The above solution (3.2.5) is a concrete realization of the abstract Galois Iterator obtained in Chapter 2, Eq. (2.2.10),

$$\beta_n = \alpha^n \beta_0 + [i_0 \alpha^{n-1} + i_1 \alpha^{n-2} + \dots + i_{n-2} \alpha + i_{n-1} 1] 1 \quad (2.2.10) \text{ repeated}$$

In (2.2.10) the quantities $\beta_0, \beta_n, \alpha^k, 1$ are abstract field elements, and the i_k are numbers in $GF(p)$. In Eq. (3.2.5) the quantities β_0, β_n , and the outside 1 are replaced with vectors $\mathbf{q}_0, \mathbf{q}_n, \mathbf{1}$. The quantities α^k and the inside 1 are replaced with matrices A^k and I .

Equation (3.2.4) could be put into a computer to generate the state vector sequence, while (3.2.5) gives the result in closed form.

Before going on to draw more conclusions, we pause to derive a similar matrix equation for the scrambler circuit of Figure 1.

3.3 Matrix Solution of Figure 1.

In notation similar to that of the previous section, we now write the equations of motion for the divider circuit of Figure 1

$$\begin{aligned} q'_{k-1} &= i - h_0 q_0 - h_1 q_1 - h_2 q_2 - \dots - h_{k-1} q_{k-1} \\ q'_{k-2} &= q_{k-1} \\ q'_{k-3} &= q_{k-2} \\ q'_{k-4} &= q_{k-3} \\ &\dots \\ q'_1 &= q_2 \\ q'_0 &= q_1 \end{aligned} \quad (3.3.1)$$

These equations can be trivially re-expressed in matrix form as follows,

$$\begin{array}{c|c|cccccc|c|c|c}
q'_{k-1} & & -h_{k-1} & -h_{k-2} & -h_{k-3} & \dots & -h_1 & -h_0 & q_{k-1} & & i \\
q'_{k-2} & & 1 & 0 & 0 & \dots & 0 & 0 & q_{k-2} & & 0 \\
q'_{k-3} & & 0 & 1 & 0 & \dots & 0 & 0 & q_{k-3} & & 0 \\
q'_{k-4} & = & 0 & 0 & 1 & \dots & 0 & 0 & q_{k-4} & + & 0 \\
\vdots & & \dots & \dots & \dots & \dots & \dots & \dots & \vdots & & \vdots \\
q'_1 & & 0 & 0 & 0 & \dots & 0 & 0 & q_1 & & 0 \\
q'_0 & & 0 & 0 & 0 & \dots & 1 & 0 & q_0 & & 0
\end{array}$$

(3.3.2)

As before, we can write this in compact vector notation as,

$$\mathbf{q}' = \mathbf{B} \mathbf{q} + \mathbf{i} \mathbf{1} \quad (3.3.3)$$

Notice that the ordering of the registers in the vector $\mathbf{q}$ is different here than it was in the previous section. Making the same notational change as before, we get a difference equation and solution:

$$\mathbf{q}_{n+1} = \mathbf{B} \mathbf{q}_n + \mathbf{i}_n \mathbf{1} \quad (3.3.4)$$

$$\mathbf{q}_n = \mathbf{B}^n \mathbf{q}_0 + [\mathbf{i}_0 \mathbf{B}^{n-1} + \mathbf{i}_1 \mathbf{B}^{n-2} + \dots + \mathbf{i}_{n-2} \mathbf{B} + \mathbf{i}_{n-1} \mathbf{I}] \mathbf{1} \quad (3.3.5)$$

The matrix $\mathbf{B}$ above is in one of the standard forms for a companion polynomial (form C_1 of Galois Section 8.5). Thus, all observations made about matrix $\mathbf{A}$ in the previous section also apply to matrix $\mathbf{B}$ here.

This is a step forward from Chapter 2. We now see that the Galois Iterator equation applies to Figure 1 as well as Figure 2. These iterator equations are (3.2.5) and (3.3.5).

3.4 Shift Register Generators Revisited

We have learned that, whether implemented as Figure 1 or Figure 2, the "equation of motion" of the state vector of a scrambler has the basic form exemplified in (3.2.5) and (3.3.5).

We are now in a position to make more powerful statements about the shift register generator than we were able to make in the previous chapter. Consider such a generator. Setting the input sequence to zero in (3.3.5) we get,

$$\mathbf{q}_n = \mathbf{B}^n \mathbf{q}_0 \quad n = 0, 1, 2, \dots \quad (3.4.1)$$

In particular we can write

$$\mathbf{B}^j \mathbf{q}_n = \mathbf{B}^{j+n} \mathbf{q}_0 = \mathbf{q}_{n+j} \quad (3.4.2)$$

Meanwhile, we know that matrix $\mathbf{B}$ is a root of the primitive polynomial $h(x)$, $h(\mathbf{B}) = 0$, so we can write:

$$\sum_{j=0}^k h_j B^j = 0 \quad (3.4.3)$$

Now apply both sides of (3.4.3) onto the vector $\mathbf{q}_n$, and make use of (3.4.2) to get

$$\sum_{j=0}^k h_j \mathbf{q}_{n+j} = 0 \quad (3.4.4)$$

Each side of this equation is a k -component vector. We have now proved the following

Fact 1: The entire state vector of a primitive polynomial shift register generator satisfies the difference equation (2.3.2). Thus, each component of the state vector satisfies (2.3.2) as well, and each component is the state of a particular register.

In Chapter 2 we learned a lot about sequences which satisfy (2.3.2), for $h(x)$ being a primitive polynomial of $\text{GF}(2^k)$. The main results were stated in Facts 8 and 9 of Section 2.3. The upshot is that any solution of (2.3.2) is a characteristic sequence. Therefore we now know that:

Fact 2: The state of each individual register in a primitive polynomial shift register generator cycles through the characteristic sequence.

Notice how this is an improvement over Fact 11 of Section 2.2.

Corollary 2: The output of a shift register generator cycles through the characteristic sequence.

Proof: This is just because the output happens to be one of the registers.

3.5 The Output of a Scrambler

In this section, finally, we consider the output of our polynomial divider circuit of either Figure 1 or Figure 2 when there is an input stream present. Such a device is known as a data scrambler. As above, we assume there are k registers, and that the feedback polynomial $h(x)$ is a primitive polynomial of $\text{GF}(2^k)$.

First, we need one more technical matter dealt with. Take (3.4.3) and multiply both sides by B^n to get,

$$\sum_{j=0}^k h_j B^{j+n} = 0 \quad (3.5.1)$$

This says that difference equation (2.3.2) is also obeyed by the following sequence of *matrices*: $\{I, B, B^2, \dots\}$. If we examine any particular matrix element of this sequence of matrices, call it the *rs* element, this sequence of numbers also satisfies (2.3.2), that is,

$$\sum_{j=0}^k h_j [B^{j+n}]_{rs} = 0 \quad (3.5.2)$$

We now just proved:

Fact 1: The sequence of numbers $[B^n]_{rs}$ for any fixed rs , and for $n=0,1,2,\dots$ either cycles through the characteristic sequence, or is identically 0. Thus, $[B^n]_{rs} = g_n$, $n = 0,1,2,\dots$ where we indicate by $\{g_n\}$ the characteristic sequence.

Aside: I feel certain that this second option is not possible ($[B^n]_{rs}$ identically 0), but do not know how to prove it. It has little bearing on the following discussion.

Example: If you look at the example at the end of Galois Section 8.5, you see that each matrix element of the set of matrices $\{A^n\}$ for $GF(8)$ runs through the sequence 0010111. Don't forget to include the identity matrix, so there is a total set of 7 matrices.

Now consider (3.3.5) in the case that the starting vector $\mathbf{q}_0 = \mathbf{0}$. In other words, we assume that at some (possibly ancient) time the scrambler was in the zero state, and then some input stream began. We then have:

$$\mathbf{q}_n = [i_0 B^{n-1} + i_1 B^{n-2} + \dots + i_{n-2} B + i_{n-1} I] \mathbf{1} = \sum_{j=0}^{n-1} i_{n-1-j} [B^j] \mathbf{1} \quad (3.5.3)$$

We can express the s^{th} component of this vector equation (corresponding to the state of some register s) as follows,

$$[q_n]_s = i_0 [B^{n-1}]_{s,1} + i_1 [B^{n-2}]_{s,1} + \dots + i_{n-2} [B]_{s,1} + i_{n-1} [I]_{s,1} = \sum_{j=0}^{n-1} i_{n-1-j} [B^j]_{s,1} \quad (3.5.4)$$

According to Fact 1, the sequence of matrix elements in this sum *is* a characteristic sequence. For convenience, we shall label this sequence as follows:

$$g_{j+1} = [B^j]_{s,1}$$

Then we can rewrite (3.5.4) as ,

$$[q_n]_s = i_{n-1} g_1 + i_{n-2} g_2 + i_{n-3} g_3 + \dots + i_1 g_{n-1} + i_0 g_n = \sum_{j=1}^n g_j i_{n-j} \quad (3.5.5)$$

We have now proved the following:

Fact 2: In the presence of an input stream $\{i_n\}$, and with initial state vector $\mathbf{q}_0 = \mathbf{0}$, the state of any register of a primitive polynomial scrambler equals the "dot product" of a characteristic sequence multiplied by the (time reversed) input sequence.

We now wish to make some comment about the statistical nature of $[\mathbf{q}_n]_s$. Remember that one of the registers (some s) in either Figure 1 or Figure 2 represents the output stream of the scrambler, so we are really seeking a statement about the scrambler output statistics. To this end we need:

Lemma: If you add up n terms of a sequence $\{r_i\}$ in $GF(2)$, and if each term in the sequence is an independent random variable with probability ε of being 1, then the probability that the sum of the n terms equals 1 is given by

$$P_n = (1/2) [1 - (1-2\varepsilon)^n]$$

Notice that, if $0 < \varepsilon < 1/2$, P_n approaches $1/2$ as n keeps increasing. Also, if $\varepsilon = 1/2$, $P_n = 1/2$.

Proof: Let P_n = probability that the sum of n terms is a 1. Then add one more term to get

$$P_{n+1} = P_n(1-\varepsilon) + \varepsilon(1-P_n) = P_n(1-2\varepsilon) + \varepsilon = \alpha P_n + \varepsilon$$

where we define $\alpha = (1-2\varepsilon)$. Iterate this difference equation starting with $P_1 = \varepsilon$ to get

$$P_n = \varepsilon (1 + \alpha + \alpha^2 + \dots + \alpha^{n-1}) = \varepsilon (1 - \alpha^n)/(1-\alpha) = (1/2) (1 - \alpha^n). \quad \text{Q.E.D.}$$

Now consider the state of any scrambler register as being the (reversed) input sequence dotted with the characteristic sequence, as shown in (3.5.4) above. Consider one particular term in this sequence, for example,

$$g_3 i_{n-3}$$

Assume that an input bit (like i_{n-3}) has a probability p of being a 1. We know (see Section 2.5, Case 1 Analysis, $\beta = p'$) that the probability of a characteristic sequence bit being 1 is $p' = \beta = (1/2)(1 + 1/P)$, where $P = 2^k - 1$ = the Period of the characteristic sequence. In other words, for reasonably large k , p' is just slightly larger than $1/2$. In order for our term $g_3 i_{n-3}$ to be 1, both factors must be 1, so the probability that this term is 1 is

$$\varepsilon = pp' \quad p = \text{input sequence prob of 1} \quad p' = (1/2)(1 + 1/P) \approx 1/2, \quad P = 2^k - 1$$

We can now use the above Lemma to obtain the probability that the n term sum shown in (3.5.5) is a 1:

$$\{ \text{Prob that } [\mathbf{q}_n]_s = 1 \} = (1/2) [1 - (1-2pp')^n]$$

Since $p' \approx 1/2$, we have $\varepsilon = pp' < 1/2$ unless p is unusually close to 1. Thus, as the scrambler continues to run and n increases, the probability that any register of the scrambler is a 1 gets closer and closer to $1/2$, *regardless* of the value of the p of the input stream, as long as p is not too close to 1.

Admittedly, we have not been too rigorous in our statistical analysis. With this caveat, we claim to have proven the following fact

Fact 3: If we assume that the input stream is characterized by a p in some reasonable range, say $0.1 < p < 0.9$, then after some number M of clocks has passed, the p of any register of the scrambler is very close to $1/2$. It differs from $1/2$ in our example by amount $(1/2)(0.9)^M$. If we let $M = 1000$ clocks, this difference quantity is on the order of 10^{-46} .

Corollary 3: Barring highly anomolous input streams, and assuming the average value of the input stream (with symbols 0,1) lies in some reasonable range like $0.1 < p < 0.9$, and assuming there are at least say 6 registers in a primitive polynomial data scrambler, so that $p' \approx 1/2$, then after the scrambler runs for on the order of 1000 clocks, the average value of the output stream will be so close to $1/2$ that the variation from $1/2$ will be unmeasurable.

Review of Leeper 1973.

A Universal Digital Data Scrambler, by David. G. Leeper, BSTJ 52,10, Dec 1973, pp 1851-1865.

This paper considers the scrambler-descrambler combination just as we are used to seeing it. Earlier papers dealt with scramblers having periodic input streams, while this paper deals with an arbitrary input stream. Subject to a certain technical qualification, the paper concludes that, by choosing a sufficiently large number of stages in the scrambler, the output sequence can be made as "white" as one requires. By this, the author means that the "first and second order statistics" of the scrambler output can be made arbitrarily close to those of white noise.

The "first order statistics" is that the probability of an output bit being a 1 can be made as close to $1/2$ as required. We have shown in our own discussion above how this comes out, and we have stolen some of Leeper's arguments in our presentation.

The "second order statistics" are twofold. First, Leeper shows that the autocorrelation function of the output sequence can be made arbitrarily close to that shown in our Figure 2.5.2 of Chapter 2. Second, he shows that any pair of symbols in the scrambler output sequence can be made as "independent" as required. This means that the "joint second-order density" $p(o_m, o_n) = p(o_m)p(o_n) + \varepsilon$, where ε can be made as small as required, again by choosing a suitable number of registers k . Here $\{o_n\}$ is the output sequence. Since the first order statistic says that $p(o_m) \approx 1/2$, he gets $p(o_m, o_n) \approx 1/4$.

The "technical qualification" of the paper is a bit tricky, and we hope that some later paper has been able to bring it a little closer to the practical realm. It is my belief that with $k=9$ stages, a scrambler designed for use with signals which come from a "live" video source will easily meet this technical qualification. The technical qualification is that the input stream cannot be "too perfect" in some sense, and the author is able to make it less perfect by intentionally adding some "error" to the input stream by simulating a binary symmetric channel with some small probability of error. Leeper comments that the inherent

noise in a digital signal that derives from an analog source is likely to have a sufficient amount of this "imperfection" so that a scrambler with a small number of registers will meet the conditions of his Universal Scrambler Theorem, which results in the conclusions noted above -- namely, that the output stream has the same first and second order characteristics as white noise.

3.6 The Spectral Density of a Scrambler Output

The basic conclusion of the previous section can be summarized as follows:

" Barring anomolous input sequences, the spectral density of the output of an operating scrambler is essentially the same as the spectral density of a random white NRZ signal."

The interesting fact we learned is that, even if $p \neq 1/2$ for the input stream, we end up with $p = 1/2$ for the output stream. This is what "white" means in our current context.

We have already derived the spectral density of such a signal in Spectral Section 6.6(a), and we quote that result here, setting $p = 1/2$:

$$\frac{\langle |X(\omega)|^2 \rangle}{T} = V^2 T_1 \left[\text{sinc}(\omega T_1/2) \right]^2 \left\{ (1/4) + (1/4) 2\pi\delta(\omega T_1) \right\} \quad \text{Spectral (6.6.1)}$$

If we integrate the DC line, we get power = $(V/2)^2$. Our "power" in effect operates on a 1 ohm load, so this is the power going into a cable due to the average voltage being $(1/2) V$. In a real system this would be blocked by adding a DC offset to the signal, then there would be no DC line at all.

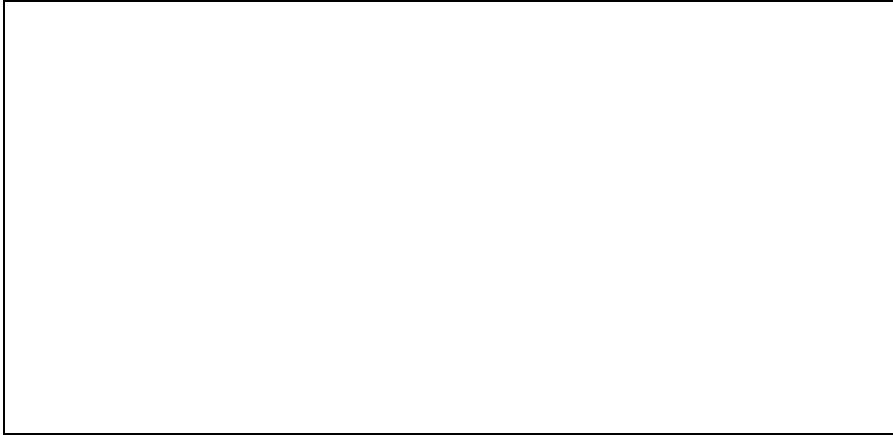
This leaves the interesting part of our scrambler output spectral density,

$$\frac{\langle |X(\omega)|^2 \rangle}{T} = (V/2)^2 T_1 \text{sinc}^2(\omega T_1/2) \quad (3.6.1)$$

Here, T_1 is the width of a NRZ "1" or "0", and V is the peak-to-peak amplitude of the NRZ signal.

Interpretation of the Spectrum

It might be helpful to look at the Spectral Figure 17.1 plot of $\text{sinc}(x)$, which we repeat here,



For a periodic square wave pulse train having half-pulses of width T_1 , the spectrum was a line spectrum with the first "fundamental" line occurring at $x = \pi/2$ of $\text{sinc}(x = \omega T_1/2)$. This line is at

$$\omega = 2\pi / 2T_1 \quad f = 1/(2T_1) = (1/2) f_1.$$

If such a signal represented the string 01010101... for a 140 Mbit/sec NRZ signal, one would have $f_1 = 140$ MHz, and $f = 70$ MHz. One would normally use a channel with slightly more than 70 MHz bandwidth, and get this line passed through, and ignore the higher lines.

Our scrambler output signal is different. The spectrum is entirely continuous, and it is the area under the above $\text{sinc}(x)$ curve squared. In the pure periodic square wave signal, each "line" was at the peak of one of the above humps. In the scrambled real signal, one can imagine that each such discrete line has been broadened out to encompass the entire hump. The right side of the hump is the "upper sideband", and the left side of the hump is the "lower sideband". There is no residual "carrier" line left at the center of each hump. However, there is amplitude at the hump centers, just as there is everywhere except at the zeros. It is just the amplitude of the continuum $\text{sinc}^2(x)$ envelope.

Fact: If you have a 140 Mbit/sec scrambled NRZ signal, and you want to pass the entire main hump, you need a system bandwidth of 140 MHz. To pass the left half of the main hump only, you are back to the 70 MHz bandwidth.

Power Distribution by Hump

It might be interesting to examine the amount of power in various portions of the spectrum. We obtain relative answers by integrating $\text{sinc}^2(x)$ over various ranges. This is not a handy closed form integral, the easiest thing is to do parts and cast the indefinite integral in terms of the $\text{Si}(x)$ (sine integral) function. We will then integrate from 0 to various points of interest:

$$\int_0^b dx \text{sinc}^2(x) = -\sin^2(b)/b + \text{Si}(2b)$$

Here is a table,

b	$2b$	$\text{Si}(2b) =$	$-\frac{\sin^2(b)}{b}$	<u>integral</u>
$\pi/2$	π	$\text{Si}(\pi) = 1.8519370$	$-2/\pi = -.6366198$	1.2153172
π	2π	$\text{Si}(2\pi) = 1.4181516$	0	1.4181516
2π	4π	$\text{Si}(4\pi) = 1.4921612$	0	1.4921612
3π	6π	$\text{Si}(6\pi) = 1.5180339$	0	1.5180339
4π	8π	$\text{Si}(8\pi) = 1.5311313$	0	1.5311313
5π	10π	$\text{Si}(10\pi) = 1.5390291$	0	1.5390291
∞	∞	$\text{Si}(\infty) = \pi/2 = 1.5707963$	0	1.5707963

And here are the contributions of the various humps, as a percent of the total power:

% Left Half of Main Hump	$= 1.2153172/1.5707963 = 0.7736950$	$= 77.4\%$
% Right Half of Main Hump	$= (1.4181516 - 1.2153172)/1.5707963 = 0.1291284$	$= 12.9\%$
% Main Hump	$= 1.4181516/1.5707963 = 0.9028234$	$= 90.2\%$
% Second Hump	$= (1.4921612 - 1.4181516)/1.5707963 = 0.0471160$	$= 4.7\%$
% Third Hump	$= (1.5180339 - 1.4921612)/1.5707963 = 0.0164711$	$= 1.6\%$
% Fourth Hump	$= (1.5311313 - 1.5180339)/1.5707963 = 0.0083381$	$= 0.8\%$

Fact: If, for a 140 Mbit/sec NRZ signal, you pass only the left half of the main hump by using a 70 MHz bandwidth system, you pass 77.4% of the power spectrum. This seems like a reasonable thing to do.

3.7 The NRZI Mini-Scrambler.

Typically, a scrambler circuit is followed by an output stage consisting of this circuit:

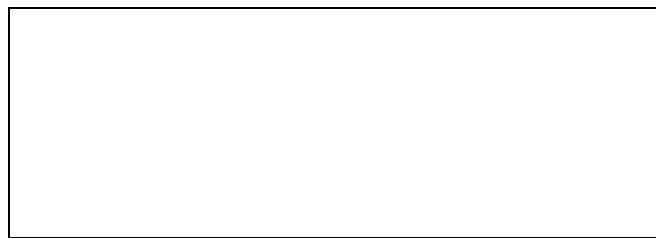


Figure 3.7.1: The NRZI mini-scrambler.

This is a $k=1$ scrambler (see Figure 1), and the only possible irreducible polynomial is $h(x) = 1 + x$, so $h_0 = h_1 = 1$. Thus, the multiplying x 's in the figure can be ignored ($-1 = +1$ in $\text{GF}(2)$).

Consider the generator matrix B for this scrambler that we would use in equation (3.3.5). The matrix represents a primitive element of $\text{GF}(2^k) = \text{GF}(2^1) = \text{GF}(2) = \{0,1\}$. Thus, the matrix is 1×1 and is simply the number "1".

$$B = 1 \quad h(B) = 0 = \text{the characteristic polynomial} = B + 1$$

The state vector for this scrambler is the state of its one register. Using this fact, and $B = 1$, we can write down equation (3.3.5) as follows:

$$o_n = o_0 + [i_0 + i_1 + i_2 + \dots + i_{n-1}] \quad (3.7.1)$$

Thus, the output of this "mini-scrambler" is simply the *sum* of the input stream plus the initial state of the register. Assume that $n-1$ bits have been input as shown, and that o_n has some value, either 0 or 1. Consider the effect of the next input bit i_n . If it is a 0, o_n does not change, and if it is a 1, o_n toggles. We have now proven that

Fact 1: The output of the above mini-scrambler changes state each time a 1 is encountered in the input data stream, and it holds its state each time a 0 is encountered in the input stream.

Comment: We have shown this result based on a trivial reduction of our matrix scrambler formalism. The result is of course obvious just looking at the above circuit, since the GF(2) adder is an XOR gate.

Definition: An *NRZI line code* is what emerges from the above circuit when you input a NRZ linecode.

NRZ means "non-return-to-zero", the idea being that on a 1 or mark, the signal stays at 1 for the entire state of the mark, which we have usually called T_1 . Various other line codes only hold the mark for half the period and are therefore called RZ or "return to zero" type codes. The I in NRZI = NRZ(I) means that one bit value (in our case a 1) is encoded as an Inversion of the level, ie, as an *edge*.

Corollary 1: In an NRZI signal as output by the above flipflop, a 1 is indicated by a state which begins with an edge, and a 0 is indicated by a state which begins with no edge. The location of the state relative to the edge is unimportant. The important thing is that a 1 is represented by an edge, and a 0 is represented by the *absence* of an edge.

Observation: Since there is no mention of the polarity of an "edge", we see that the NRZI signal is non-polar. If you have a differential NRZI signal carried on a twisted pair, it does not matter which way you hook up the wires at the receiver. Hooking them up "wrong" just changes the polarity of edges, which does not affect the interpretation of the NRZI signal in terms of 1's and 0's. The desire to have a non-polar signal is the only reason for the use of the mini-scrambler following the main scrambler.

Question: What is the effect of the mini-scrambler on the output of the main scrambler in terms of statistics and frequency spectrum?

On its own, the mini-scrambler cannot add any statistical whiteness to its incoming data stream. It is too weak to do this. We have seen above how the mini-scrambler produces an output stream that is just a sum of the input stream. This should be compared with Equation (3.5.5) of Section 3.5 which shows what a "real" scrambler does to its input stream. The stream is multiplied term by term with the characteristic sequence of the scrambler. For reasonable k , this characteristic sequence $\{g_i\}$ is "pseudorandom" and "induces" onto the output stream its random nature.

For the mini-scrambler, the length of the characteristic sequence is $2^1 - 1 = 1$, so the characteristic sequence is simply $\{1, 1, 1, 1, 1 \dots\}$. You can imagine in Eq. (3.7.1) that the input sequence is multiplied by

this light-weight characteristic sequence. Obviously, the sequence $\{1,1,1,1,1 \dots\}$ does not carry a lot of whiteness. It is like a bad detergent.

Nevertheless, the mini-scrambler is applied after the action of the main scrambler. Thus, the input to the mini-scrambler is already "white". This means that $p = 1/2$ and the individual bits emerging from the main scrambler are to a high degree statistically independent. We can therefore use the Lemma of Section 3.5.

This Lemma describes exactly the situation we have above. The output sequence of the mini-scrambler is a sum of n terms of the sequence $\{i\}$ in $GF(2)$, and the input bits are independent. Thus, we get:

$$\text{output stream probability of } 1 = P_n = (1/2) [1 - (1 - 2/2)^n] = 1/2$$

In other words, white-in gives white-out. If the input stream to the mini-scrambler is slightly off white, perhaps $p = 1/2 + \delta$ say, then we get

$$\text{output stream probability of } 1 = P_n = (1/2) [1 - (-2\delta)^n]$$

This shows that the P_n of the mini-scrambler output oscillates around the $P_n = 1/2$, always getting closer to it as time goes by. So in this sense, the whiteness of the main scrambler is somewhat improved by the mini-scrambler. However, the main issue here is the statistical independence of the bits, ie, the "higher order statistics" that you cannot see just looking at the first order statistical quantity P_n . It is the job of the main scrambler to generate this independence.

We summarize these remarks as follows:

Scrambler Summary:

(a) We first assume that the main scrambler does its job of producing a white output sequence. If the output of this scrambler is applied through an output driver to a line, the resulting spectrum is obtained from Eq. (2.6.2) for the single-ended Case 1 signal of Figure 2.6.1, and from Eq. (2.6.4) for the differential Case 2 signal of Figure 2.6.2. In each case, we set $P = \infty$ since we assume full whiteness.

$$\frac{\langle |X(\omega)|^2 \rangle}{T} = \frac{|X_{\text{pulse}}(\omega)|^2}{T_1} (1/4) \left\{ 1 + \sum_{m=-\infty}^{\infty} 2\pi\delta(\omega T_1 - 2\pi m) \right\} \quad (2.6.2) \text{ Case 1}$$

$$\frac{\langle |X(\omega)|^2 \rangle}{T} = \frac{|X_{\text{pulse}}(\omega)|^2}{T_1} \{ 1 \} \quad (2.6.4) \text{ Case 2}$$

Here, $X_{\text{pulse}}(\omega)$ is the Fourier Integral transform of the pulse shape used to drive the line at the output of the scrambler. T_1 is the time duration of one state (high or low). The LHS of each of these equations is the "spectral power density" $P(\omega)$, so that $P(\omega)d\omega$ is measured in watts.

(b) If the main scrambler input is set to all zeros -- a highly non-random data stream -- the above formulas are still correct with a very small amount of error. The error is in the form of the $1/P$ corrections appearing in (2.6.2) and (2.6.4), where P = length of main scrambler characteristic sequence.

(c) Under the assumption of (a), the insertion of the NRZI mini-scrambler between the main scrambler and the output driver circuit has zero effect on the statistics of the output signal, and zero effect on the above power spectrum of the output signal.

(d) If the line driving pulse is selected to be an ideal box of width T_1 and height V , and if we add a DC offset to the driven signal as needed to eliminate any DC component (add a capacitor), the Case 1 result above reduces as shown in Eq. (3.6.1),

$$\frac{\langle |X(\omega)|^2 \rangle}{T} = (V/2)^2 T_1 \text{sinc}^2(\omega T_1/2)$$

The power spectrum is continuous, there are no discrete lines whatsoever.

(e) Other pulse shapes can be evaluated using the above formulas and the Fourier Transform discussed in the Spectral notes.

(f) One of the main motivations for using a scrambler in the first place is to obtain the above continuous spectrum. Since power is smoothly distributed over a continuum of frequencies and there are no sharp lines, crosstalk between cables and circuits carrying scrambled signals is minimized. RFI emissions are also held in check.

(g) Notice that this is true even if the input signal to the main scrambler is all zeros. In this case, the main scrambler simply outputs its characteristic sequence.

(h) Another motivation for scrambling is to reduce the frequency of occurrence of large "edge-less" periods in the signal. Such streams make clock recovery more difficult. Because of the NRZI output circuit, only strings of zeros output by the main scrambler are of concern in this regard. According to our discussion in Section 2.4, the probability distribution of strings of n zeros output by the main scrambler (and also by the mini-scrambler) is independent of the distribution of such strings in the source signal, and is given to a high degree of accuracy by the simple formula $P(n) = 2^{-n}$. When the source input stream is all zeros, the maximum length of a zero output string is $k-1$. However, for real input data, arbitrarily long strings of zeros are possible with relative probability 2^{-n} . This subject will be discussed further in Section 3.11 below.

3.8. Matrix analysis of the polynomial multiplier (de-scrambler) of Figure 7 .

As noted earlier, the polynomial multiplier circuits are easier to analyze than the corresponding divider circuits mainly because the multipliers have no feedback. Since this is a Chapter on the Matrix Approach, and since we later want to prove by brute force that the scrambler-descrambler combination really does pass data unaltered, we pause here to analyze our multiplier circuits.

For the circuit shown in Figure 7, the equation of motion for the state vector is very simple, and the output is a linear combination of the contents of the registers. That is, the output is a function of the state vector. The data simply flows through the flipflops from left to right, so that $q_0' = q_1$, and $q_1' = q_2$, etc. When this is put in our usual matrix form we get,

$$\begin{array}{c|c|cccccc|c|c|c}
 q'_{k-1} & & 0 & 0 & 0 & \dots & 0 & 0 & q_{k-1} & & i \\
 q'_{k-2} & & 1 & 0 & 0 & \dots & 0 & 0 & q_{k-2} & & 0 \\
 q'_{k-3} & & 0 & 1 & 0 & \dots & 0 & 0 & q_{k-3} & & 0 \\
 q'_{k-4} & = & 0 & 0 & 1 & \dots & 0 & 0 & q_{k-4} & + & 0 \\
 \dots & & \dots & \dots & \dots & \dots & \dots & \dots & \dots & + & \dots \\
 q'_1 & & 0 & 0 & 0 & \dots & 0 & 0 & q_1 & & 0 \\
 q'_0 & & 0 & 0 & 0 & \dots & 1 & 0 & q_0 & & 0
 \end{array} \quad (3.8.1)$$

The matrix here has the same form as in Eq. (3.3.2) for the scrambler divider circuit of Figure 1, except here all the feedback entries in the first row are zero. As before, we use matrix/vector notation to rewrite the above as $\mathbf{q}' = \mathbf{C}\mathbf{q} + i\mathbf{1}$. We then re-install the time index n to get the vector difference equation and its solution:

$$\mathbf{q}_{n+1} = \mathbf{C} \mathbf{q}_n + i_n \mathbf{1} \quad (3.8.2)$$

$$\mathbf{q}_n = \mathbf{C}^n \mathbf{q}_0 + [i_0 \mathbf{C}^{n-1} + i_1 \mathbf{C}^{n-2} + \dots + i_{n-2} \mathbf{C} + i_{n-1} \mathbf{I}] \mathbf{1} \quad (3.8.3)$$

The column vector $\mathbf{q}_n$ has the following components, where we revert to a former notation of putting the time index as an argument,

$$[\mathbf{q}_n]_s = q_{k-s}(n) \quad (3.8.4)$$

For example, with $s=1$ the first component of the column vector $\mathbf{q}_n$ equals the register q_{k-1} of Figure 7 at time n .

We have named the above matrix "C", not to be confused with the companion C of Galois Chapter 8. The matrix C has the following matrix elements, as can be seen from (3.8.1),

$$C_{ij} = \delta_{i,j+1}, \quad [C^2]_{ij} = \delta_{i,j+2}, \quad \dots \quad [C^n]_{ij} = \delta_{i,j+n} \quad (3.8.5)$$

The matrix C^2 has the ones slid down to the next lower subdiagonal, and so on for higher powers. Finally, the matrix C^{k-1} has a single non-vanishing element, a 1 in the lower left corner. All higher powers $C^n = 0$ for $n \geq k$. Thus, in the solution (3.8.3), most terms vanish. Only those terms having powers less than k survive. We therefore rewrite (3.8.3) as,

$$\mathbf{q}_n = \mathbf{C}^n \mathbf{q}_0 + [i_{n-k} \mathbf{C}^{k-1} + i_{n-3} \mathbf{C}^2 + i_{n-2} \mathbf{C} + i_{n-1} \mathbf{I}] \mathbf{1} \quad \mathbf{I} = \mathbf{C}^0$$

$$= C^n \mathbf{q}_0 + \sum_{j=1}^k i_{n-j} [C^{j-1}] \mathbf{1} \quad (3.8.6)$$

Interpretation: If $n \geq k$, the current state vector of the multiplier is affected only by the most recent k input symbols. If $n < k$, the first term in (3.8.6) has not yet vanished, so some components of the starting state vector $\mathbf{q}_0$ still have an effect.

These comments are of course totally trivial, since the circuit is just a shift register. After the first k clocks of operation, the original contents (the starting state vector) have been shifted out and have no influence on anything that happens in the future. This is in sharp contrast to what happens in the polynomial divider circuit, where the starting state vector has a lingering influence that lasts forever.

An exercise in notation.

This section is included only to demonstrate how to deal with the strange matrix/vector notation used above and in earlier sections of this Chapter. The purpose of this exercise is to use the above matrix/vector notation and to try and recover the time-domain statement of how a polynomial multiplier works.

In an upcoming section we will be attempting a rigorous time-domain proof that a descrambler really does recover the source data stream that went into the scrambler. In other words, we will prove that a self-synchronizing scrambler/descrambler combination really works. In that proof, manipulations of the kind appearing below will be used a lot, so the following "exercise" is a sort of warm up.

Consider the s^{th} component of the vector equation (3.8.6),

$$[q_n]_s = \sum_{j=1}^k [C^n]_{s,j} [\mathbf{q}_0]_j + \sum_{j=1}^k i_{n-j} [C^{j-1}]_{s,1} \quad (3.8.7)$$

According to (3.8.4), we have $[C^n]_{s,j} = \delta_{s,n+j}$ and $[C^{j-1}]_{s,1} = \delta_{s,j}$ so both sums go away giving

$$[q_n]_s = [\mathbf{q}_0]_{s-n} + i_{n-s} = [\mathbf{q}_0]_{s-n} \theta(n < s) + i_{n-s} \theta(n \geq s) \quad (3.8.8)$$

Here we have added explicit θ functions [$\theta(A) = 1$ if A true, 0 if A false] to show that only one of the two terms can be non-vanishing for a given s . The vector index $s-n$ must be in the range $(1,k)$ so $s-n \geq 1$ $n < s$ for the first term. And the input sequence is assumed to begin with i_0 , hence $n-s \geq 0$ in the second term.

Now use (3.8.4) to rewrite this last result as

$$[q_n]_s = q_{k-s(n)} = q_{k-s+n(0)} \theta(n < s) + i_{n-s} \theta(n \geq s) \quad (3.8.9)$$

This is an elaborate statement of a trivial fact, see Figure 7. Consider register q_{k-s} . If the number of clocks n is less than s , then after n clocks from startup, register q_{k-s} contains the startup contents of the register n to the left. If $n > s$, register q_{k-s} contains i_{n-s} . If this still seems strange, pick a particular value of s (try $s=1$) and stare at Figure 7.

The output of the polynomial multiplier can be written in terms of the components of vector q_n :

$$o_n = \sum_{s=0}^{k-1} h_s [q_n]_{k-s} + h_k i_n \quad (3.8.10)$$

If the vector solution (3.8.9) is inserted into (3.8.10), we can solve for the output sequence o_n of our polynomial multiplier:

$$o_n = \sum_{s=0}^{k-n-1} h_s q_{s+n}(0) + \sum_{s=k-n}^k h_s i_{n+s-k} \quad (3.8.11)$$

As a test of this result, let's examine the first two terms in this output sequence,

$$\begin{aligned} o_0 &= \sum_{s=0}^{k-1} h_s q_s(0) + h_k i_0, \\ o_1 &= \sum_{s=0}^{k-2} h_s q_{s+1}(0) + h_{k-1} i_0 + h_k i_1 \end{aligned}$$

For $n \geq k$, the first sum contributes nothing, and the second sum has its full range:

$$o_n = \sum_{s=0}^k h_s i_{n+s-k} \quad n \geq k \quad (3.8.12)$$

Setting $n := n+k$ we get

$$o_{n+k} = \sum_{s=0}^k h_s i_{n+s} \quad n \geq 0 \quad (3.8.13)$$

Finally we have recovered our time domain result appearing in the box at the end of Section 1.8 in Chapter 1.

3.9 Matrix analysis of the standard polynomial multiplier of Figure 8 .

Looking at Figure 8, one can write down the following series of equations,

$$\begin{aligned}
 q'_1 &= h_0 i \\
 q'_2 &= q_1 + h_1 i \\
 q'_3 &= q_2 + h_2 i \\
 &\dots \\
 q'_{k-1} &= q_{k-2} + h_{k-2} i
 \end{aligned} \tag{3.9.1}$$

These equations may be combined into a single matrix equation,

$$\begin{array}{c|c|c|c|c|c|c|c|c|c|c|c}
 q'_1 & & 0 & 0 & 0 & \dots & 0 & 0 & q_1 & & h_0 \\
 q'_2 & & 1 & 0 & 0 & \dots & 0 & 0 & q_2 & & h_1 \\
 q'_3 & & 0 & 1 & 0 & \dots & 0 & 0 & q_3 & & h_2 \\
 q'_4 & = & 0 & 0 & 1 & \dots & 0 & 0 & q_4 & + i \times & h_3 \\
 \dots & & \dots & \dots & \dots & \dots & \dots & \dots & \dots & & \dots \\
 q'_{k-1} & & 0 & 0 & 0 & \dots & 0 & 0 & q_{k-1} & & h_{k-2} \\
 q'_k & & 0 & 0 & 0 & \dots & 1 & 0 & q_k & & h_{k-1}
 \end{array} \tag{3.9.2}$$

We recognize the matrix above as the same matrix C which appeared in the Figure 7 analysis. Notice that the rightmost vector used to be $\mathbf{1}$, but here it is a vector consisting of the first k components of the feedback polynomial coefficients. We call this vector $\mathbf{h}$.

As we have done many times before, we write the above as a vector difference equation and state the solution of this difference equation. We also write out a component of the vector $\mathbf{q}_n$:

$$\mathbf{q}_{n+1} = C \mathbf{q}_n + i_n \mathbf{h} \tag{3.9.3}$$

$$\mathbf{q}_n = C^n \mathbf{q}_0 + [i_0 C^{n-1} + i_1 C^{n-2} + \dots + i_{n-2} C + i_{n-1} I] \mathbf{h} \tag{3.9.4}$$

$$[\mathbf{q}_n]_s = q_s(n) \tag{3.9.5}$$

The comments made about the matrix C all apply here as well, so we can rewrite (3.9.4) as,

$$\begin{aligned}
 \mathbf{q}_n &= C^n \mathbf{q}_0 + [i_{n-k} C^{k-1} + i_{n-3} C^2 + i_{n-2} C + i_{n-1} I] \mathbf{h} & I = C^0 \\
 &= C^n \mathbf{q}_0 + \sum_{j=1}^k i_{n-j} [C^{j-1}] \mathbf{h}
 \end{aligned} \tag{3.9.6}$$

In Figure 7, the output o_n was a sum of many terms. For Figure 8, the output is the sum of only two terms,

$$\begin{aligned}
o_n &= q_k(n) + h_k i_n = [q_n]_k + h_k i_n \\
&= \sum_{j=1}^k [C^n]_{k,j} [q_0]_j + \sum_{j=1}^k \sum_{s=0}^{k-1} i_{n-j} [C^{j-1}]_{k,s} h_s + h_k i_n \quad (3.9.7)
\end{aligned}$$

According to (3.8.4), we have $[C^n]_{k,j} = \delta_{k,n+j}$ and $[C^{j-1}]_{k,s} = \delta_{k,j+s}$ so both j sums go away leaving

$$\begin{aligned}
o_n &= [q_0]_{k-n} + \sum_{s=k-n}^{k-1} h_s i_{n+s-k} + h_k i_n \\
&= \theta(n < k) [q_{k-n}(0) + h_k i_n] + \theta(n \geq k) \sum_{s=0}^k h_s i_{n+s-k} \quad (3.9.8)
\end{aligned}$$

The left term controls the output for the first n clocks. For example,

$$o_0 = q_k(0) + h_k i_0.$$

$$o_1 = q_{k-1}(0) + h_k i_1.$$

Notice that this is completely different from the output of the Figure 7 circuit. However, after n clocks have gone by, the second term takes over and we get the same results as for Figure 7, namely, equations (3.8.12) and (3.8.13).

This concludes our matrix exercises for the polynomial multiplier circuits. The obvious conclusion in either case is that nobody cares about the output of these circuits for the first k clocks. After that, the output is given by the fundamental result (3.8.13).

3.10. Proof that a Descrambler really descrambles the output of a Scrambler.

The proposed serial digital standard involves the following scrambler - descrambler structure:



Figure 3.10.1: Proposed scrambler structure for serial digital communication .

One can think of this as an operation $S_1 S_2 D_2 D_1$ acting on the signal.

We want to prove that a "descrambler" really descrambles the output of a scrambler. In terms of operations, this means that $S D = 1$, or $D = S^{-1}$. The "1" means there is no change to the signal after passing through a scrambler followed by a descrambler.

We will show this for an arbitrary scrambler. Then, with regard to the above picture, we will have shown that

$$S_1 S_2 D_2 D_1 = S_1 S_2 S_2^{-1} S_1^{-1} = S_1 (S_2 S_2^{-1}) S_1^{-1} = S_1 (1) S_1^{-1} = S_1 S_1^{-1} = 1$$

Here subscript 1 refers to the Main scrambler and descrambler, while 2 refers to the mini NRZI operations as described in Section 3.7 above.

Proof in the Z Domain

Back at the end of Section 1.8 in Chapter 1 we wrote down the z-domain and time-domain equations for polynomial dividers (scramblers) and multipliers (descramblers). Here they are again:

	<u>z-domain</u>	<u>time-domain</u>	
Poly Divider (scrambler)	$O(z) = I(z)/H(z)$	$i_n = \sum_{j=0}^k h_j o_{n+j}$	(3.10.1)

(Figure 1 or Figure 2)

$$\text{Poly Multiplier (descrambler)} \quad z^k O(z) = I(z) H(z) \quad o_{k+n} = \sum_{j=0}^k h_j i_{n+j} \quad (3.10.2)$$

(Figure 7 or Figure 8)

In the z domain, our proof is very simple and goes as follows. Let $I(z)$ represent the source data. Let $O(z)$ be the output of a scrambler with coefficients $H(z)$. Then ,

$$O(z) = I(z)/H(z)$$

Now run this $O(z)$ as the input to a descrambler, whose output we will call $O'(z)$:

$$z^k O'(z) = \{ O(z) \} H(z) = \{ I(z)/H(z) \} H(z) = I(z)$$

This says that the output $O'(z)$ of the descrambler replicates the input $I(z)$ to the scrambler, apart from a time delay of k clocks. This delay is characteristic of the multiplier circuit.

Recall that all these z -transform functions of z are just polynomials. Of course the ones other than $H(z)$ can be arbitrarily long, since their corresponding sequences can be long.

The above proof is correct, but somewhat unsatisfying. We would really like to see how this all works out in the time domain.

Proof in the Time Domain

In Section 3.3 we constructed an explicit formula for the output of a Figure 1 scrambler, using our matrix formalism. Equation (3.3.5) shows the status of the state vector $\mathbf{q}$ at time $t = n$ clocks:

$$\mathbf{q}_n = B^n \mathbf{q}_0 + \sum_{j=0}^{n-1} i_{n-1-j} B^j \mathbf{1} \quad (3.10.3)$$

Here $\mathbf{q}_0$ is the starting state vector, $\mathbf{1}$ is the unit column vector $(1, 0, 0, \dots)$, and B is the matrix discussed in Section 3.3. According to Eq. (3.3.2), the state of register k -s is given by the s^{th} component of the vector $\mathbf{q}_n$. Thus we can write,

$$o_n = q_{k-s}(n) = [\mathbf{q}_n]_s = [B^n \mathbf{q}_0]_s + \sum_{j=0}^{n-1} i_{n-1-j} [B^j]_{s,1} \quad (3.10.4)$$

Notice in Figure 1 that we can take the scrambled output from *any* of the k flipflops, the only difference is a time delay. So we will therefore regard $q_{k-s}(n)$ as shown above as the output o_n of our scrambler. Eq. (3.10.4) is an explicit expression for the scrambler output sequence o_n in terms of the scrambler input sequence i_n , and the starting state vector $\mathbf{q}_0$.

During the first k clocks of operation, we know that the *descrambler* clocks out whatever k garbage bits happened to be in its k registers. During this time, the first k bits of the above sequence o_n move into the descrambler registers. We can use (3.10.2) above to compute the descrambler output after this time, assuming that its input is the output of the scrambler:

$$o'_{k+n} = \sum_{r=0}^k h_r o_{n+r} \quad n \geq 0 \quad (3.10.5)$$

Our task is now "simply" to insert (3.10.4) with $n \rightarrow n+r$ into (3.10.5), and turn the crank.

The reader is warned that things are going to get a little tricky as we proceed, but in the end, the desired result will be obtained, and some insight will be gained along the way.

Doing the above insertion, we get

$$o'_{k+n} = \sum_{r=0}^k h_r \{ [B^{r+n} \mathbf{q}_0]_s + \sum_{j=0}^{n+r-1} i_{n+r-1-j} [B^j]_{s,1} \} \quad (3.10.6)$$

Consider the first term above, which can be expanded as ,

$$\sum_{r=0}^k h_r [B^{r+n} \mathbf{q}_0]_s = \sum_{i=0}^k [\mathbf{q}_0]_i \{ \sum_{r=0}^k h_r [B^{r+n}]_{s,i} \} = \sum_{i=0}^k [\mathbf{q}_0]_i \{0\} = 0 \quad (3.10.7)$$

According to Eq. (3.5.2), the sum over $r \{ \dots \}$ completely vanishes. This may be traced back to the fact that $h(B) = 0$. The reason this is true is that the matrix B has the "companion" form C_1 shown in Galois Chapter 8, Section 8.5, with $f_i = h_i$. The characteristic polynomial of C_1 is then $h(x)$, and according to Big Fact 2 of Section 8.1, it follows that $h(B) = 0$. This is the Cayley-Hamilton Theorem.

Since the term containing $\mathbf{q}_0$ completely vanishes in (3.10.6), we conclude that the output sequence o'_{k+n} is *completely independent* of the starting state vector $\mathbf{q}_0$ of the scrambler!! This is the kind of result that is hard to deduce from the z -domain analysis.

We are now left with the following double sum:

$$o'_{k+n} = \sum_{r=0}^k h_r \sum_{j=0}^{n+r-1} i_{n+r-1-j} [B^j]_{s,1} \quad (3.10.8)$$

Replace the j summation variable with a new summation variable $m = n+r-1-j$:

$$o'_{k+n} = \sum_{r=0}^k h_r \sum_{m=0}^{n+r-1} i_m [B^{r+n-m-1}]_{s,1} \quad (3.10.9)$$

The motivation for this less-than-obvious replacement was to get i_m . Consider now the shape of the region of the double summation in (3.10.8). Here is a picture:

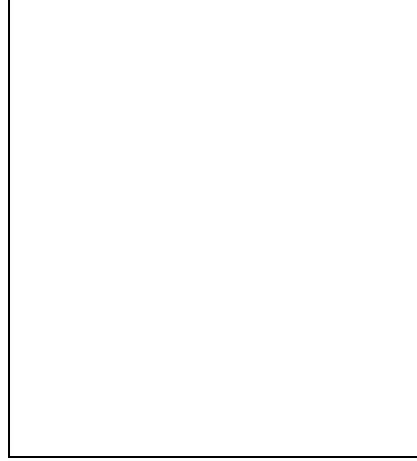


Figure 3.10.2: Summation domain for Eq. (3.10.9)

Consider first the summation over the rectangular region A. Here, r goes 0 to k , and m goes 0 to $n-2$. The line $m=n-1$ will be considered part of the upper triangle B. We get,

$$o'_{k+n} \text{ (over region A)} = \sum_{r=0}^k h_r \sum_{m=0}^{n-2} i_m [B^{r+n-m-1}]_{s,1} \quad (3.10.10)$$

Because the upper endpoint on the m summation is now independent of r , we can move the r summation to the right where it collides with the B matrix thing to give zero just as happened in (3.10.7):

$$o'_{k+n} \text{ (over region A)} = \sum_{m=0}^{n-2} i_m \left\{ \sum_{r=0}^k h_r [B^{r+n-m-1}]_{s,1} \right\} = 0 \quad (3.10.11)$$

Now we have only to worry about the triangular region A. So,

$$o'_{k+n} = o'_{k+n} \text{ (over region B)} = \sum_{r=0}^k h_r \sum_{m=n-1}^{n-2} i_m [B^{r+n-m-1}]_{s,1} \quad (3.10.12)$$

The next step is to replace index m with $j = m - n + 1$ which yields,

$$o'_{k+n} = \sum_{r=0}^k h_r \sum_{j=0}^r i_{j+n-1} [B^{r-j}]_{s,1} \quad (3.10.13)$$

This time we cannot move in the r summation and have it die against the matrix because r appears as the upper endpoint of the j summation. The summation domain for (3.10.13) is as follows:

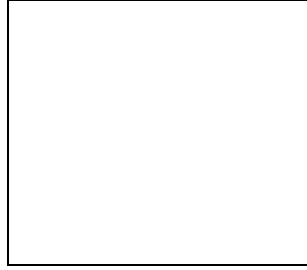


Figure 3.10.3: Summation domain for Eq. (3.10.13)

Based on this picture, we can rewrite the double summation as follows:

$$o'_{k+n} = \sum_{j=0}^k i_{j+n-1} \left\{ \sum_{r=j}^k h_r [B^{r-j}]_{s,1} \right\} \quad (3.10.14)$$

In Appendix 3.1 at the end of this Chapter, we prove the following extremely un-obvious fact about the quantity in brackets $\{..\}$ appearing above,

$$\left\{ \sum_{r=j}^k h_r [B^{r-j}]_{s,1} \right\} = \delta_{s, k-j+1} \quad (3.10.15)$$

Basically this is a property of the B matrices. Inserting (3.10.15) into (3.10.14) we get our final result:

$$o'_{k+n} = \sum_{j=0}^k i_{j+n-1} \{ \delta_{s, k-j+1} \} = i_{k+n-s} \quad (3.10.16)$$

Recall that we took our scrambler output from register q_{k-s} of the scrambler of Figure 1. If we now select $s=k$, the output comes from register q_0 exactly as shown in Figure 1. In this case, we get the result,

$$o'_{k+n} = i_n \quad n = 0, 1, 2, \dots \quad (3.10.17)$$

Conclusions:

(a) Eq. (3.10.17) says that if you run a sequence i_n through a scrambler, and then through the corresponding descrambler, you get out exactly what you put in, with k clock delays due to the way the multiplier starts up. During this startup phase, k garbage bits are emitted by the descrambler.

(b) If you decide to take your scrambler output from an earlier register of Figure 1, then you will lose the first few bits of the input stream during the garbage clock out phase. For example, if $(k-s) = 1$, so we are taking data from q_1 in Figure 1, we get $o'_{k+n} = i_{n+1}$, so the first non-garbage bit out of the descrambler will be i_1 and i_0 is lost.

(c) Conversely, suppose there are N extra latch delays inserted into the "line" of Figure 3.10.1. The N garbage bits held in these latches will appear right after the k garbage bits are shifted out of the descrambler latches. Then following these N extra garbage bits will come the input sequence starting with the first bit i_0 , so nothing is lost.

(d) In general, nobody cares whether the first few bits are lost as in (b), or whether there are a few extra startup garbage bits as in (c). The only possible concern is in the overall delay which is $N-(k-s)$. An interesting idea is to consider taking the scrambler output a few latches back from the end (ie, advanced) and attempt to cancel out any line delay N . That is, one could tune so that $N - (k-s) = 0$.

(e) Notice that in the above proof we assumed use of the Figure 1 scrambler with its B matrices, but we made no assumption about whether Figure 7 or Figure 8 was used as the descrambler.

3.11 The Kill Sequence Problem and Strings of Zeros

It has been noted that a long string of zeros emerging from the Main Scrambler shown in Figure 3.10.1 can cause a problem at the clock recovery circuit which must exist at the end of the "line". Unlike strings of 1's, strings of 0's pass right through the NRZI mini-scrambler and enter the line.

Question: What type of source data can cause the main scrambler to output a long string of zeros?

Answer: At any instant of time, the main scrambler contains some state vector q_n . As will be shown below, for any such state vector, there is a unique corresponding "kill vector" which, if shifted into the scrambler, will bring it to the all-zero state. Subsequent zeros at the input keep the scrambler in its zero state. This leads us to the following:

Fact 1: It is possible that a k -stage scrambler will output a string of $N+k$ zeros if a string of N zeros exists in the input stream. The probability of this happening, given the input stream of N zeros, and assuming the scrambler is in a random state, is $1/P$ where $P = 2^k - 1$. That's just because P is the number of states the scrambler can be in. For $k=9$, this probability is $1/511$, which is not a very small number in this context.

Proof: Imagine that the input data stream has the following form:

...xxxxxxxxx10000000000000...00000000001xxxxx...

Suppose that , at the instant in time indicated by the arrow, the scrambler happens to contain exactly the right k -bit vector $\mathbf{a}$ which gets "killed" by the next k bits of the input stream (underlined). Then, when the first 0 of the long string of zeros shown is at the scrambler input, the scrambler will contain k zeros. Suppose the string of zeros shown above has length N . By the time the "1" which ends the string is at the scrambler input, it will have output N zeros. But its state vector is still 0, so there are k more zeros still to come out. Thus, if N zeros are input, it is possible that $N+k$ zeros will be output.

Example 1: Suppose in D1 video there is a sync pattern consisting of the following four 10-bit words :

3FF, 000, 000, 000 001

Suppose this pattern is followed by a Y or a C sample, with an unknown bit ordering. In the very worst case (probably a few bits worse than could really happen), this pattern could be in effect a 1, and we have added that above. Thus, we have a string of 39 bits (in 10-bit video), and with a $k=9$ scrambler, there could be a string of 48 zeros in the scrambler output. [We address below the question of what vector is killed by the all-ones input vector.] In 8-bit video, we would lower this estimate from $30+9+9 = 48$ to $24 + 7 + 9 = 40$.

In 8-bit video, the normal blanking pattern is $YC = 0x1080$, so the longest zero strings are 10 bits. In 10-bit video, we presume this extends to 12 bits.

Example 2: Suppose in some video spec, one is allowed to have an entire active scanline of ancillary data that is all zeros. In the 601 world with 10-bit video, this would represent a string of $720*20 = 14,400$ Zeros. To this we might add $30 + 9$ as above, but who is counting? In HDTV we get $1920*20 = 38400$ zeros.

We now prove some of the claims referred to above.

Fact 2: Assume a scrambler is in some state $\mathbf{a}$. Consider the k -symbol pattern that next shifts in, call this pattern $\mathbf{i}$. After this pattern shifts in, the scrambler is in state $\mathbf{b}$. We claim that, given any patterns $\mathbf{a}$ and $\mathbf{b}$, there exists a unique pattern $\mathbf{i}$ which causes $\mathbf{a}$ to go to $\mathbf{b}$.

Corollary 2: If pattern $\mathbf{b}$ is the all-zero pattern $\mathbf{0}$, there exists a unique pattern $\mathbf{i}$ which takes $\mathbf{a}$ to $\mathbf{0}$.

Definition: Such a pattern is called a *kill vector*.

Proof of Fact: Our proof makes no assumption about the nature of the symbols carried on the lines of Figure 1. They could be in $GF(2)$, in $GF(p)$, or in $GF(p^m)$. To keep the proof general, we retain the minus signs on the h_i coefficients.

Stare at Figure 1. Assume that at $t=0$ pattern $\mathbf{a}$ is in the registers. The rightmost bit of pattern $\mathbf{a}$ is a_0 . Assume the the first incoming bit is called i_0 . We select i_0 such that the resulting contents of register q_{k-1} will be the rightmost bit b_0 of pattern $\mathbf{b}$. This is because b_0 sitting in register q_{k-1} will end up in register q_0 after k clocks.

A clock goes by and we now have some i_1 at the input. The feedback circuit is generating some feedback based on the remaining (shifted) bits of pattern **a** and our b_0 which is sitting in q_{k-1} . We now select i_1 so that when combined with the feedback, it generates the second-from-the-right bit b_1 of pattern **b**.

We keep going in this fashion, and each bit is in turn fully determined. Here are some equations:

$$i_0 - h_0 a_0 - h_1 a_1 - \dots - h_{k-1} a_{k-1} = b_0 \quad \text{solve for } i_0$$

$$i_1 - h_0 a_1 - h_1 a_2 - \dots - h_{k-2} a_{k-1} - h_{k-1} b_0 = b_1 \quad \text{solve for } i_1$$

$$i_2 - h_0 a_2 - h_1 a_3 - \dots - h_{k-2} b_0 - h_{k-1} b_1 = b_2 \quad \text{solve for } i_2$$

etc.

We can group all these equations into a matrix solution for the $\{i_n\}$ as follows:

$$\begin{array}{c|c|cccccccc|c|c|c} i_0 & & a_0 & a_1 & a_2 & a_3 & \dots & a_{k-3} & a_{k-2} & a_{k-1} & & -h_0 & & b_0 \\ i_1 & & a_1 & a_2 & a_3 & a_4 & \dots & a_{k-2} & a_{k-1} & b_0 & & -h_1 & & b_1 \\ i_2 & & a_2 & a_3 & a_4 & a_5 & \dots & a_{k-1} & b_0 & b_1 & & -h_2 & & b_2 \\ i_3 & = & a_3 & a_4 & a_5 & a_6 & \dots & b_0 & b_1 & b_2 & & -h_3 & + & b_3 \\ i_4 & & a_4 & a_5 & a_6 & a_7 & \dots & b_1 & b_2 & b_3 & & -h_4 & & b_4 \\ \dots & & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & & \dots & & \dots \\ i_{k-2} & & a_{k-2} & a_{k-1} & b_0 & b_1 & \dots & b_{k-5} & b_{k-4} & b_{k-3} & & -h_{k-2} & & b_{k-2} \\ i_{k-1} & & a_{k-1} & b_0 & b_1 & b_2 & \dots & b_{k-4} & b_{k-3} & b_{k-2} & & -h_{k-1} & & b_{k-1} \end{array}$$

Notice that all elements on any / diagonal are the same. We can write this in vector form as:

$$\mathbf{i} = -X(\mathbf{a}, \mathbf{b}) \mathbf{h} + \mathbf{b}$$

where X is the matrix shown, and we note that it is a function of vectors **a** and **b**.

Thus, we have explicitly constructed the input vector **i** which converts state **a** into state **b**. If we are looking for a kill vector, we set $\mathbf{b} = 0$, so all elements in X below the diagonal vanish, as does the add-on column vector on the right. Since we have an explicit answer for **i**, it must be unique, given **a** and **b**.

Example: For $k=9$, X is a 9×9 matrix, and for $h(x) = 1 + x^4 + x^9$ we have $h_0 = h_4 = 1$, all other h_i in the vector **h** vanish. In this case, each equation has only three terms. Here they are:

$$\begin{array}{ll} i_0 = a_0 + a_4 + b_0 & i_5 = a_5 + b_0 + b_5 \\ i_1 = a_1 + a_5 + b_1 & i_6 = a_6 + b_1 + b_6 \\ i_2 = a_2 + a_6 + b_2 & i_7 = a_7 + b_2 + b_7 \\ i_3 = a_3 + a_7 + b_3 & i_8 = a_8 + b_3 + b_8 \\ i_4 = a_4 + a_8 + b_4 & \end{array}$$

This is the general result for going from **a** to **b**. To find the *kill vector*, set $b_i = 0$ to get:

$$\begin{array}{ll}
 i_0 = a_0 + a_4 & i_5 = a_5 \\
 i_1 = a_1 + a_5 & i_6 = a_6 \\
 i_2 = a_2 + a_6 & i_7 = a_7 \\
 i_3 = a_3 + a_7 & i_8 = a_8 \\
 i_4 = a_4 + a_8 &
 \end{array}$$

Question: In the above example $k = 9$, what vector $\mathbf{a}$ is killed by an incoming vector consisting of all ones?

Answer: Reverse solve the last set of equations above. We quickly get that $a_8 = a_7 = a_6 = a_5 = 1$. Then from the earlier equations we get $a_4 = a_3 = a_2 = a_1 = 0$. Finally, the first equation then says $a_0 = 1$. The answer to the question is therefore:

$$\{a_8, a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0\} = \{1, 1, 1, 1, 0, 0, 0, 0, 1\} = 0x1E1$$

Discussion of the zeros problem.

In general, for each video standard, one must find the spec, and study the longest possible string of zeros that is allowed. Officially illegal values do sometimes occur, and there are all kinds of EAV, SAV codes and ancillary data and various indices to worry about, plus embedded sound. The general principle applies: "If it can happen, it will happen." If the clock recovery circuit drops out of lock during a long string of zeros, there will be some period of time determined by the PLL system during which data will be lost. If the lock-up time is very short, perhaps only a few bits at the end of an anomolous scan line will be lost, but this sounds unhealthy. It seems that a long time constant would be better than a short one, so you could live through an anomolous event without losing lock. A time constant that is long compared to a scanline but short compared to a field time would be good. In this case, one could survive an anomolous scanline.

The following fact does not really belong here, but is was omitted from earlier work.

Fact 3: If a k -stage scrambler has an even number of feedback taps, then if the input stream is set to all 1's, the scrambler behaves exactly as if the input stream were all 0's and there were an inverter on the final output.

Thus, a primitive polynomial scrambler with all 1's for input will cycle through an inverted characteristic sequence of the usual length $P = 2^k - 1$. The only pattern that does not appear at the output is then the pattern consisting of k ones. All characteristic sequence theorems quoted earlier then need to be adjusted to account to the effective inversion.

Proof: The input feed of all ones at the input XOR gate can be thought of as an inverter at the input of the first flip flop and an input stream of zeros. We can then mentally move this inverter to the right in Figure 1 one step at a time. At the output of a flip flop, the inverter "forks" into a pair of inverters, one moves to the next flipflop, and one moves up the feedback path. The lower inverter eventually ends up on the output signal. If there are an even number of feedback paths, then the feedback forked inverters all cancel out. Each such inverter can be regarded as adding a 1, and if you add a 1 an even number of times, it is like doing nothing.

Some final notes:

- (1) Long strings of zeros might have some bad effect on a dynamic equalizer.
- (2) For each extra bit that is added to a scrambler, the probability of an anomolous event is cut in half. Inside a custom chip, having 9 registers or having 19 in a scrambler does not make much difference, but the 19 bit scrambler has the "zeros problem" reduced by a factor of $2^{-10} = 10^{-3}$.
- (3) There is no data propagation problem to speak of with the descrambler. If a data loss occurs, the descrambler will "resynchronize" in k clock periods. In other words, if any garbage bits get into the descrambler, they do not have any long term affect.

Appendix 3.1: Proof of Eq. (3.10.15)

First, we state what we want to prove is true:

$$\left\{ \sum_{r=j}^k h_r [B^{r-j}]_{s,1} \right\} = \delta_{s, k-j+1} \quad (A.1)$$

The matrix B which goes with the Figure 1 circuit is shown in Eq. (3.3.2). We can write its elements as,

$$B_{ij} = \delta_{i, j+1} + \delta_{i,1} h_{k-j} \quad (A.2)$$

We are always working in the GF(2) binary world where $- = +$. Also, since we are assuming the $h(x)$ is order k , we must have $h_k = 1$.

We will prove (A.1) by induction.

(a) To start, we show that (A.1) is true for $j = k$. In this case, the LHS of (A.1) has one term and (A.1) says,

$$h_k I_{s,1} = \delta_{s, 1}$$

But this is trivially true since $h_k = 1$ and I = identity matrix $= B^0$. So, we are off and running.

(b) Now we show that if (A.1) is true for some j , then it is also true for $j-1$. Start with (A.1), and apply

the operation $\sum_{s=0}^k [B]_{n,s}$ to both sides. On the LHS of (A.1), this new B matrix glues onto the B^{r-j} matrix already present to give $B^{r-(j-1)}$. On the RHS the delta picks out the value $s = k-j+1$ for the second index of B, so we get:

$$\left\{ \sum_{r=j}^k h_r [B^{r-(j-1)}]_{n,1} \right\} = B_{n, k-j+1} = \delta_{n, k-(j-1)+1} + \delta_{n,1} h_{j-1} \quad (A.3)$$

where we have used (A.2) to write out the B matrix element. Now, if the sum on the LHS of (A.3) *had* a term for $r = j-1$, here is what that term would be:

$$h_{j-1} [B^0]_{n,1} = h_{j-1} \delta_{n,1}$$

But this is the second term on the right of (A.3). So (A.3) now becomes,

$$\left\{ \sum_{r=j-1}^k h_r [B^{r-(j-1)}]_{n,1} \right\} = \delta_{n, k-(j-1)+1} \quad (A.4)$$

If we now set $n=s$, (A.4) is the same as (A.1) with j replaced by $j-1$. We have therefore shown that (A.1) implies (A.4), which completes the second part of our induction proof.

Q.E.D.