RANDOMIZERS AND DERANDOMIZERS: IS THE PROCESS REVERSIBLE?

Michael Rice, Raymond Barrier, Niraj Sharma Chaulagain
Brigham Young University
Provo, Utah, USA

ABSTRACT

Mathematical descriptions of the IRIG 106 randomizer and derandomizer are developed and used to show the impact of shift register initial conditions on the input/output relations for a cascade of randomizer and derandomizer operations (the normal order) and a cascade of derandomizer and randomizer operations (the reverse order). The results show that for the normal order, mismatched randomizer and derandomizer shift register initial conditions impact *at most* the first 15 bits. But for the reverse order, mismatched randomizer and derandomizer initial conditions impact *all* of the bits. Consequently, the shift register initial conditions must be the same to recover data when the randomizer and derandomizer are operated in the reverse order. The mathematical representation is based on polynomials and the generating function concept and the results are confirmed using computer programs based on bit-level operations.

INTRODUCTION

Randomizers were introduced in aeronautical telemetry to deal with the problems caused by long sequences of consecutive ones or zeros in NRZ-L data. A long sequence of consecutive ones or zeros has a strong DC component. This is a problem because read/write heads for magnetic tape recorders/players have no DC response. When the DC level of the data sequence drifts due to a strong DC component, read/write errors occur. This is especially a problem at low data rates where a sequence of consecutive ones or zeros does not have to be that long to cause the problem. At high data rates this can still be a problem if a sequence of consecutive ones and zeros is long enough.

Randomizers solve the DC drift problem by breaking up long sequences of consecutive ones or zeros by "scrambling" or "randomizing" (hence the name) the data in a known way. Randomizers are defined in IRIG-106 by a shift register circuit with feedback connections. The corresponding derandomizer is a shift register circuit with feedforward connections in the same locations as the randomizer's feedback connections. Because long sequences of consecutive ones or zeros first caused problems in magnetic recording, the definitions for the randomizer and derandomizer are

found in Appendix D of IRIG 106.

Randomizers are also used in PCM/FM modulators. Early FM modulators had AC-coupled inputs and the issue of drifting DC levels for long sequences of consecutive ones or zeros caused problems in the modulator input circuitry similar to the problems in magnetic recording. Randomization produced additional benefits for PCM/FM.

- Long sequences of consecutive ones or zeros skew the PCM/FM spectrum. By breaking up sequences of consecutive ones or zeros, randomization produces a more balanced spectrum.
- Bit synchronizers can lose lock for long sequences of consecutive ones or zeros with NRZ-L encoding. The synchronization loss is due to the absence of waveform transitions. (For this reason, bit synchronization loss does not occur with bi-phase encoding.) Randomization creates a more "random" data pattern that contains sufficient transition density for the bit synchronizer to perform.

The spectrum issue carries over to the more advanced SOQPSK-TG and ARTM CPM modulations. Long sequences of consecutive ones or zeros create undesirable properties in the RF spectrum. The bit synchronization problem also carries over to the integrated symbol timing synchronization functions in SOQPSK-TG and ARTM CPM demodulators. This is why most SOQPSK-TG and ARTM CPM modulators have a provision for randomization. Because encryption tends to randomize the data, it is rare that both encryption *and* randomization are used simultaneously for the RF link. Given the fact that most aeronautical telemetry links are encrypted, the use of randomizers and derandomizers for the RF link have become less common.

Because randomization is used less frequently for the RF link, it is easy for a test engineer to lose track of whether or not the demodulator output is randomized. When this happens, the test engineer may apply a derandomizer to data that was never randomized and store only the derandomizer output. After a short period of fear and panic, the question that presents itself is,

> For the case where unrandomized data has been passed through a derandomizer, can the original data be recovered?

The temptation is to apply a randomizer to the derandomized data (i.e., apply the randomizer and derandomizer in the reverse order). A quick experiment using laboratory hardware shows that this rarely works. (This observation sometimes leads to an even longer period of fear and panic.) In this paper, we show that applying randomization and derandomization in the reverse only works if the randomizer (second step) has the same initial register settings as the derandomizer (first step).

To derive this answer, we develop mathematical descriptions of the randomization and derandomization processes. We apply these mathematical descriptions to the normal order (randomization followed by derandomization) to confirm the mathematical descriptions. Next we apply the descriptions in the reverse order and identify the conditions required to recover the data. In the examples that accompany the development, we use short versions of the randomizer and derandomizer to simplify the mathematics so that the equations fit within the margins of this paper. The short version consists of 3 registers. We show that the mathematical descriptions, along with the conclusions drawn from mathematical descriptions, carry directly over to the case of longer IRIG 106 randomizer and derandomizer comprising 15 registers.

PRELIMINARIES

Randomizers and derandomizers operate on bits, each of which assumes one of two binary values usually labeled 0 and 1. These binary values define an algebraic structure called a *finite field* in which addition and multiplication follow the rules of modulo-2 arithmetic. The addition and multiplication tables are

$$\begin{array}{c|cc} + & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 1 & 0 \end{array} \qquad \begin{array}{c|cc} \times & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \end{array} \tag{1}$$

The interesting feature of modulo-2 addition is that $1 + 1 = 0$. This means addition "+" and subtraction "$-$" are the same. In the development below, we make no distinction between "+" and "$-$" and always use "+".

For the purposes of answering the central question in this paper, the best way to mathematically describe the operations performed by randomizer and derandomizer is to use concept of the *generating function*. The generating function corresponding to the bit sequence

$$b_0, b_1, b_2, \ldots, b_n, \ldots$$

is

$$B(D) = b_0 + b_1 D + b_2 D^2 + \cdots + b_n D^n + \cdots . \tag{2}$$

The variable D in an *indeterminate*. This means D is never assigned a value. It is a symbol, or place holder, whose exponent indicates delay. For example, the generating function for the bit sequence 1 0 1 1 0 1 0 0 is

$$\begin{aligned} B(D) &= 1 + 0 \times D + 1 \times D^2 + 1 \times D^3 + 0 \times D^4 + 1 \times D^5 + 0 \times D^6 + 0 \times D^7 \\ &= 1 + D^2 + D^3 + D^5. \end{aligned} \tag{3}$$

In this context, a ratio of polynomials simply defines polynomial division. As an example, the generating function $1/(1 + D)$ means

$$\begin{array}{r} 1 + D + D^2 + D^3 \\ 1 + D \;\overline{)\; 1} \\ \underline{1 + D} \\ D \\ \underline{D + D^2} \\ D^2 \\ \underline{D^2 + D^3} \\ D^3 \\ \underline{D^3 + D^4} \\ D^4 \end{array}$$

Note that multiplication and subtraction operations at each stage follow the modulo-2 arithmetic rules (1). In this way

$$\frac{1}{1+D} = 1 + D + D^2 + D^3 + \cdots \tag{4}$$

defines the all-ones bit pattern. The generating function $1/(1+D^2)$ means

$$\begin{array}{r} 1 + D^2 + D^4 + D^6 + D^8 \\ 1 + D^2 \;\overline{\big|\; 1} \\ \underline{1 + D^2} \\ D^2 \\ \underline{D^2 + D^4} \\ D^4 \\ \underline{D^4 + D^6} \\ D^6 \\ \underline{D^6 + D^8} \\ D^8 \end{array}$$

This shows that

$$\frac{1}{1+D^2} = 1 + D^2 + D^4 + D^6 + \cdots \tag{5}$$

and it defines the alternating bit pattern 1 0 1 0 1 0 ....

RANDOMIZERS, DERANDOMIZERS, AND THEIR GENERATING FUNCTIONS

The randomizer circuit we consider here is illustrated in Figure 1. The output is given by the recursion

$$y_n = x_n + y_{n-2} + y_{n-3} \tag{6}$$

for $n \geq 3$ where the initial register values are r_2, r_1, and r_0. With the aid of the block diagrams in Figure 2, the first few outputs are

$$y_0 = x_0 + r_1 + r_0 \qquad y_1 = x_1 + r_2 + r_1 \qquad y_2 = x_2 + y_0 + r_2 \qquad y_3 = x_3 + y_1 + y_0. \tag{7}$$

The generating function for the output is

$$Y(D) = (x_0 + r_1 + r_0) + (x_1 + r_2 + r_1)D + (x_2 + y_0 + r_2)D^2 + (x_3 + y_1 + y_0)D^3 + \cdots \tag{8}$$

Multiplying through by the indeterminate D and rearranging produces

$$Y(D) = X(D) + D^2Y(D) + D^3Y(D) + (r_1 + r_0) + (r_2 + r_1)D + r_2D^2. \tag{9}$$

Solving for $Y(D)$ gives the final result

$$Y(D) = \frac{X(D) + (r_1 + r_0) + (r_2 + r_1)D + r_2D^2}{1 + D^2 + D^3}. \tag{10}$$

There are several important observations regarding this result.

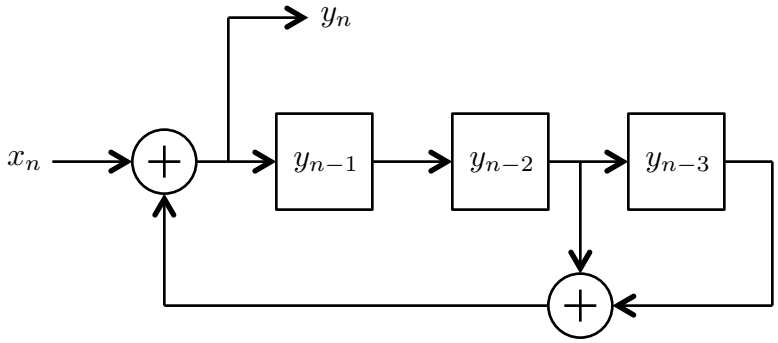

Figure 1: A block diagram of a short randomizer described by the recursion (6).

- The generating function for the randomizer output is a ratio of two polynomials in D. This defines an infinite length bit sequence if the denominator polynomial does not divide the numerator polynomial (which is always the case for real telemetry data). Note that this is true even if the input bit sequence is a finite length bit sequence. In other words, if the randomizer were clocked an infinite number of cycles, it would produce nonzero outputs long after the finite length input bit sequence contained anything other than zeros. In practice, the randomizer is only clocked as long as the input bit sequence is valid. This is why the randomizer output contains the same number of bits as the randomizer input.
- The denominator polynomial is consequence of the recursive relationship defining the output. The recursion is a consequence of the feedback paths in the circuit of Figure 1. Observe that the feedback connections in the block diagram of Figure 1 define the denominator polynomial in (10).
- The randomizer output is also a function of the initial register values r_2, r_1, and r_0. These three initial values form the coefficients of the second polynomial in numerator of (10).

From the generating function point of view, the randomizer divides a modified version of the input generating function by a degree-3 polynomial in D. The division is characteristic of shift register systems containing feedback.

The derandomizer circuit we consider here is illustrated in Figure 3. The output is given by

$$y_n = x_n + x_{n-2} + x_{n-3} \tag{11}$$

for $n \geq 3$ where the initial register values are d_2, d_1, and d_0. With the aid of Figure 4, the first few outputs are

$$y_0 = x_0 + d_1 + d_0 \qquad y_1 = x_1 + d_2 + d_1 \qquad y_2 = x_2 + x_0 + d_2 \qquad y_3 = x_3 + x_1 + x_0. \tag{12}$$

The generating function for the output is

$$Y(D) = (x_0 + d_1 + d_0) + (x_1 + d_2 + d_1)D + (x_2 + x_0 + d_2)D^2 + (x_3 + x_1 + x_0)D^3 + \cdots \tag{13}$$

Multiplying through by the indeterminate D and rearranging produces

$$Y(D) = X(D) + D^2X(D) + D^3X(D) + (d_1 + d_0) + (d_2 + d_1)D + d_2D^2 \tag{14}$$

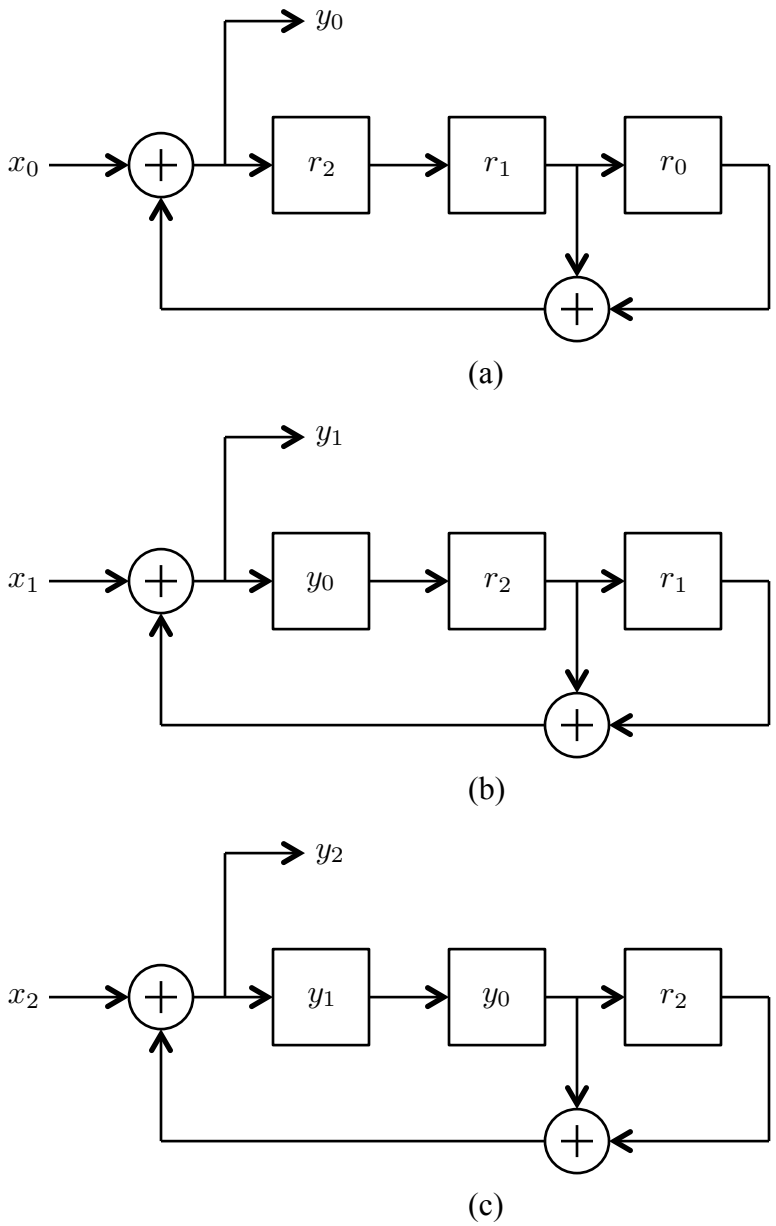

Figure 2: A block diagram of a short randomizer showing the first three outputs corresponding to the first three inputs. Also shown is the evolution of the register contents. Note that starting with y_3, the output is given by the recursion (6).

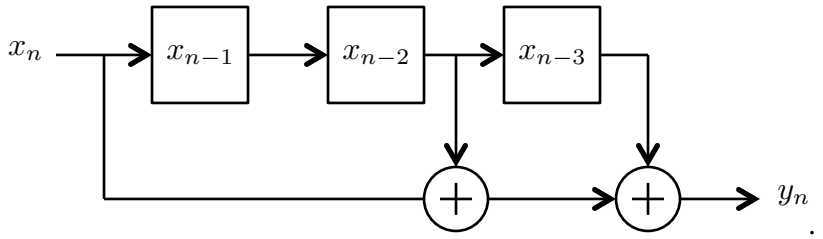

Figure 3: A block diagram of the short derandomizer described by (11).

from which we see that the generating function for the output may be expressed as

$$Y(D) = (1 + D^2 + D^3)X(D) + (d_1 + d_0) + (d_2 + d_1)D + d_2D^2. \tag{15}$$

Here we see that from the generating function point of view, the derandomizer multiples the generating function of the input sequence by a degree-3 polynomial in D. This is a characteristic of feedforward systems. The initial conditions are captured by the degree-2 polynomial on the right-hand side of (15). Observe that unlike the randomizer, the derandomizer initial conditions only impact the first three bits of the derandomizer output.

Close examination of the generating functions for the randomizer and derandomizer outputs, equations (10) and (15), respectively, shows that each is based on the same polynomial

$$G(D) = 1 + D^2 + D^3. \tag{16}$$

The randomizer divides the generating function of its input by $G(D)$ whereas the derandomizer multiplies the generating function of its input by $G(D)$. The generating function approach allows one to see that the derandomizer performs the inverse operation of the randomizer. This is why data recovery is possible. The randomizer output may be expressed as

$$Y(D) = \frac{X(D) + I_r(D)}{G(D)} \tag{17}$$

where

$$I_r(D) = (r_1 + r_0) + (r_2 + r_1)D + r_2D^2 \tag{18}$$

and $G(D)$ is given by (16). The derandomizer output may be expressed as

$$Y(D) = G(D)X(D) + I_d(D) \tag{19}$$

where

$$I_d(D) = (d_1 + d_0) + (d_2 + d_1)D + d_2D^2 \tag{20}$$

and $G(D)$ is given by (16).

We now show that the mathematical representations give the correct outputs. First, consider the application of randomization and derandomization in the proper order as illustrated in Figure 5 (a). The randomizer output is

$$Y(D) = \frac{X(D) + I_r(D)}{G(D)} \tag{21}$$

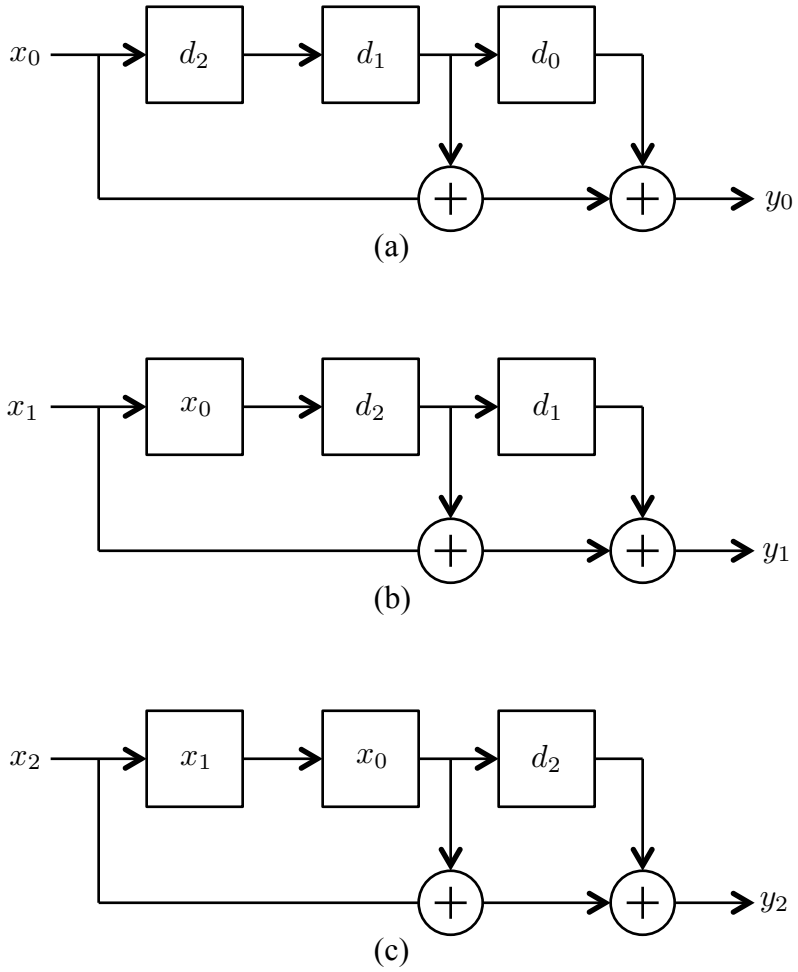

Figure 4: A block diagram of a short derandomizer showing the first three outputs corresponding to the first three inputs. Also shown is the evolution of the register contents. Note that starting with y_3, the output is given by (11).

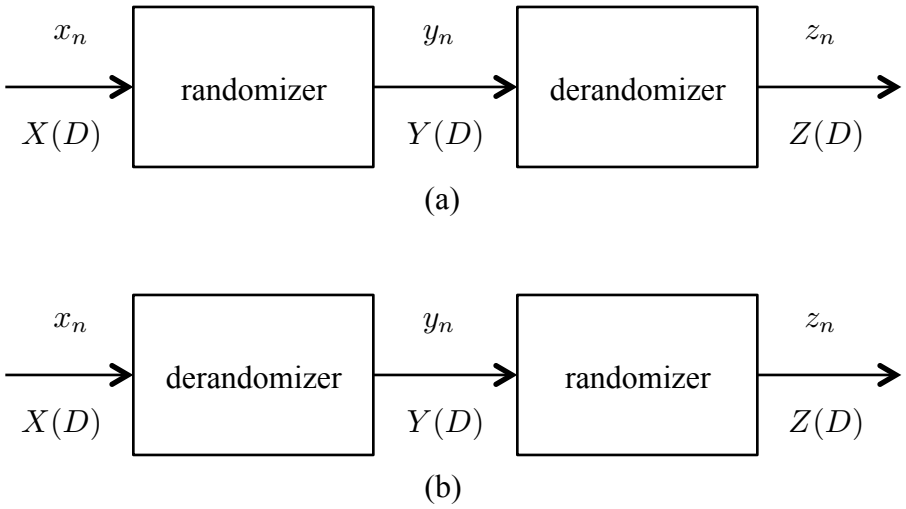

Figure 5: Randomizers and derandomizers along with their inputs and corresponding generating functions: (a) Randomization and derandomization applied in the intended order; (b) Randomization and derandomization applied in the reverse order.

and the derandomizer output is

$$\begin{aligned} Z(D) &= G(D)Y(D) + I_d(D) \\ &= G(D)\frac{X(D) + I_r(D)}{G(D)} + I_d(D) \\ &= X(D) + I_r(D) + I_d(D). \end{aligned} \tag{22}$$

The generating function for the derandomizer output given by (22) shows that the derandomizer output is the randomizer input plus a term defined by the initial register settings of the randomizer and derandomizer. Because the polynomials $I_r(D)$ and $I_d(D)$ are polynomials of at most degree 2, the initial register states impact at most the first 3 bits. The remaining bits are unaffected by the initial register states. This is why little attention is paid to the initial register settings. Note that if $I_r(D) = I_d(D)$, then $I_r(D) + I_d(D) = 0$ so that $Z(D) = X(D)$. This means the derandomizer output matches the randomizer input exactly.

The situation is different when the order is reversed. With reference to the block diagram shown in Figure 5 (b), the derandomizer output is

$$Y(D) = G(D)X(D) + I_d(D) \tag{23}$$

and the randomizer output is

$$\begin{aligned} Z(D) &= \frac{Y(D) + I_r(D)}{G(D)} \\ &= \frac{G(D)X(D) + I_d(D) + I_r(D)}{G(D)} \\ &= X(D) + \frac{I_d(D) + I_r(D)}{G(D)} \end{aligned} \tag{24}$$

The generating function for the randomizer output given by (24) shows that the randomizer output is the derandomizer input plus a term defined by the initial register settings of the randomizer and derandomizer. Unfortunately, this additive term is a rational expression and therefore defines an infinite length bit sequence. Consequently, this term can impact *all* of the derandomizer input bits. The only way to force $Z(D) = X(D)$ is to force this second additive term to be zero. The only way to force the second additive term to be zero is to force the numerator to be zero. The only way the numerator can be zero is when $I_d(D) = I_r(D)$. This leads us to the following condition:

> With reference to the reverse-order application of the randomizer and derandomizer functions illustrated in Figure 5 (b), the only way to obtain $Z(D) = X(D)$ is to force the initial register settings in both the randomizer and derandomizer to be the same.

We include some examples to illustrate how the polynomial representations based on generating functions and corresponding algebraic operations correspond to bit level operations.

Example 1. This example explores the system of Figure 5 (a). Let x be the bit sequence 1 0 1 1 0 1 0 0 whose corresponding polynomial is

$$X(D) = 1 + D^2 + D^3 + D^5.$$

Now consider the randomizer illustrated in Figure 1 with initial conditions $r_0 = 1$, $r_1 = r_2 = 0$ (that is, the right-most register is initialized to 1). These initial conditions are captured by the polynomial (18) which, for this example, is $I_r(D) = 1$ The polynomial representation of the randomizer output is

$$Y(D) = \frac{X(D) + I_r(D)}{G(D)} = \frac{(1 + D^2 + D^3 + D^5) + 1}{1 + D^2 + D^3} = D^2 + D^3 + D^4 + D^5 + D^8 + \cdots$$

A computer program implementing the shift register circuit of Figure 1 was used to confirm the results. The C++ code is listed in the Appendix. This program produces $y = 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0$. Note that the computer program was only clocked for 8 cycles corresponding to the practice of clocking the randomizer only as long as valid input data is available. The computer program output is equivalent to the result obtained using the generating functions.

Moving on to the derandomizer, suppose the bit sequence y is applied to the derandomizer circuit of Figure 3 whose initial register settings are $d_0 = d_1 = d_2 = 0$. The derandomizer initial conditions are captured by the polynomial (20) which, for this example, is $I_d(D) = 0$. The generating function of the output is

$$\begin{aligned} Z(D) &= G(D)Y(D) + I_d(D) \\ &= (1 + D^2 + D^3)(D^2 + D^3 + D^4 + D^5) + 0 \\ &= D^2 + D^3 + D^5. \end{aligned}$$

A computer program implementing the shift register circuit of Figure 3 was written to confirm the results. The C++ code is listed in the Appendix. This program produces $z = 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0$. There are two important observations here.

1. The bit level operations defined by the shift register circuit (and the computer program) produce exactly the same result as the polynomial operations based on the generating functions.
2. This example illustrates that when the initial conditions of the randomizer and derandomizer are not the same, the impact is confined to the first few bits. (The first 3 bits for our small example, the first 15 bits for the IRIG 106 randomzier and derandomizer.)

The second observation captured by Equation (22) which predicts that for the arrangement illustrated in Figure 5 (a), the output $Z(D)$ and the input $X(D)$ differ by $I_r(D) + I_d(D)$. In this example, $I_r(D) + I_d(D) = 1 + 0 = 1$, which predicts that the derandomizer output differs from the randomizer input in the first position only. This is exactly what was observed!

Example 2. This example explores the system of Figure 5 (b). Let x be the bit sequence 1 0 1 1 0 1 0 0 whose corresponding polynomial is

$$X(D) = 1 + D^2 + D^3 + D^5.$$

Now consider the derandomizer illustrated in Figure 3 with initial conditions $d_0 = d_1 = d_2 = 0$. The generating function for the derandomizer output is

$$\begin{aligned} Y(D) &= G(D)X(D) + I_d(D) \\ &= (1 + D^2 + D^3)(1 + D^2 + D^3 + D^5) + 0 \\ &= 1 + D^4 + D^5 + D^6 + D^7 + D^8 \end{aligned}$$

The derandomizer computer program, listed in the Appendix, produces $y = 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1$. The computer program output is identical to the first 8 terms in $Y(D)$. This is because the computer program was clocked for 8 bit cycles. Had the computer program been clocked 3 more clock cycles, the resulting bit sequence would be identical to $Y(D)$.

Now, moving to the second half of Figure 5 (b), suppose the bit sequence y is applied to the randomizer of Figure 1 with initial conditions $r_0 = 1$, $r_1 = r_2 = 0$ (that is, the right-most register is initialized to 1). The generating function for the output is

$$Z(D) = \frac{Y(D) + I_r(D)}{G(D)} = \frac{(1 + D^4 + D^5 + D^6 + D^7 + D^8) + 1}{1 + D^2 + D^3} = D^4 + D^5 + D^7 + \cdots$$

The randomizer computer program listed in the Appendix gives $z = 0\ 0\ 0\ 0\ 1\ 1\ 0\ 1$ which is identical to $Z(D)$. The important observation here is that the derandomizer input sequence x and the randomizer output sequence z are quite different. This difference is predicted by (24):

$$\begin{aligned} Z(D) &= X(D) + \frac{I_d(D) + I_r(D)}{G(D)} \\ &= 1 + D^2 + D^3 + D^5 + \frac{0 + 1}{1 + D^2 + D^3} \\ &= 1 + D^2 + D^3 + D^5 + (1 + D^2 + D^3 + D^4 + D^7 + \cdots) \\ &= D^4 + D^5 + D^7 + \cdots \end{aligned}$$

As explained above, the problem is with the second term. If the two initial conditions were the same, then the derandomizer input and the randomizer output would be identical.

For the purposes of illustration, suppose the initial conditions of the derandomizer used in this example were changed to $d_0 = 1$, $d_1 = d_2 = 0$. Then $I_d(D) = 1$ and the generating function for the derandomizer output is

$$\begin{aligned} Y(D) &= G(D)X(D) + I_d(D) \\ &= (1 + D^2 + D^3)(1 + D^2 + D^3 + D^5) + 1 \\ &= D^4 + D^5 + D^6 + D^7 + D^8 \end{aligned}$$

(The computer program in the Appendix gives $y = 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1$.) Applying the bit sequence y to the randomizer input with initial conditions $r_0 = 1$, $r_1 = r_2 = 0$ gives

$$Z(D) = \frac{Y(D) + I_r(D)}{G(D)} = \frac{(D^4 + D^5 + D^6 + D^7 + D^8) + 1}{1 + D^2 + D^3} = 1 + D^2 + D^3 + D^5.$$

(The computer program in the Appendix gives $z = 1\ 0\ 1\ 1\ 0\ 1\ 0\ 0$.) This example confirms the accuracy of the generating function based abstraction and shows that the conclusions drawn from the generating function based analysis are valid. Armed with this confidence, we now generalize to the IRIG 106 randomizer and derandomizer.

EXTENSION TO THE IRIG 106 RANDOMIZER AND DERANDOMIZER

All of the forgoing analysis applies to the IRIG 106 randomizer and derandomizer with the following changes. The polynomial describing the shift register connections is

$$G(D) = 1 + D^{14} + D^{15}. \tag{25}$$

This is a degree-15 polynomial because there are 15 stages in the IRIG 106 randomizer and derandomizer shift registers. The initial shift register contents of the IRIG 106 randomizer are (from left to right)

$$r_{14}, r_{13}, r_{12}, r_{11}, \ldots, r_1, r_0.$$

The corresponding "initial condition polynomial" for the IRIG 106 randomizer is the degree-14 polynomial

$$I_r(D) = (r_1 + r_0) + (r_2 + r_1)D + (r_3 + r_2)D^2 + \cdots + (r_{14} + r_{13})D^{13} + r_{14}D^{14}. \tag{26}$$

Likewise, the initial shift register contents of the IRIG 106 derandomimzer are (from left to right)

$$d_{14}, d_{13}, d_{12}, d_{11}, \ldots, d_1, d_0$$

and the corresponding "initial condition polynomial" for the IRIG 106 derandomizer is the degree-14 polynomial

$$I_d(D) = (d_1 + d_0) + (d_2 + d_1)D + (d_3 + d_2)D^2 + \cdots + (d_{14} + d_{13})D^{13} + d_{14}D^{14}. \tag{27}$$

Equipped with these changes, the input/output relationship for the randomizer-derandomizer ordering of Figure 5 (a) is

$$Z(D) = X(D) + I_r(D) + I_d(D). \tag{28}$$

This relationship shows that when the initial conditions for the randomizer and derandomizer shift registers are the same, $Z(D) = X(D)$. This means that the bits at the derandomizer output are identical to the bits at the randomizer input. When the initial conditions for the randomizer and derandomizer shift registers are not the same $I_r(D) \neq I_d(D)$. Because $I_r(D)$ and $I_d(D)$ are degree-14 polynomials, the randomizer input and the derandomizer output differ in *at most* the first 15 positions. Thereafter, the randomizer input and the derandomizer output are identical. Because 15 bits such a small fraction of the total number of bits and because the 15 incorrect bits almost always occur before frame synchronization has been achieved, there is little reason for a telemetry engineer to pay much attention to the initial register settings. In other words, devoting the effort to ensure the initial register settings are the same does not solve any problem that has presented itself.

The input/output relationship for the reverse order shown in Figure 5 (b) is

$$Z(D) = X(D) + \frac{I_d(D) + I_r(D)}{G(D)}. \tag{29}$$

If $I_d(D) = I_r(D)$, then $I_r(D) + I_d(D) = 0$ and $Z(D) = X(D)$. That is, the derandomizer input x and the randomizer output z are the same. If $I_d(D) + I_r(D)$ is anything other than the all-zeros polynomial, the second term in (29) defines an infinitely long non-zero bit sequence. This infinitely long non-zero bit sequence is the difference between the derandomizer input bit sequence x and the randomizer output bit sequence z. Because the infinitely long non-zero bit sequence is infinitely long, the derandomizer input bit sequence and the randomizer output bit sequence will never eventually coincide. Consequently, the initial settings for the derandomizer and randomizer registers must be the same if any data are to be recovered.

DISCUSSION AND CONCLUSION

The mathematical models developed in this paper show that randomizers divide a polynomial formed from the input data and the shift register initial conditions by the polynomial (25) and the derandomizer multiplies a polynomial formed from its input and the shift register initial conditions by the polynomial (25). These complementary operations (division and multiplication) are what make possible data recovery after randomization or derandomization.

A byproduct of the mathematical model is an explanation of the behavior of the IRIG 106 derandomizer when the input is the length-$(2^{15} - 1)$ PN sequence (often called the "PN-15" sequence). The PN-15 sequence is an example of a "maximal length shift register sequence." A maximal length shift register sequence based on a K-stage shift register with feedback connections.[1] The feedback connections are selected as follows: the contents of the K-stage shift register define the

[1]The shift register circuit shown in Figure 1 (with the input and accompanying adder removed) is an example that generates the maximal length shift register sequence of length 7 (the PN-3) sequence. To generate the PN-3 sequence, the initial register states must be non-zero.

state of the K-stage shift register. A K-stage shift register has 2^K possible states. The feedback connections are selected so that the $2^K - 1$ non-zero shift register states repeat with the maximum period of $2^K - 1$. Using the generating function concepts outlined in this paper, the PN-15 sequence may be represented as

$$\frac{I_{\text{PN}}(D)}{1 + D^{14} + D^{15}} \tag{30}$$

where $I_{\text{PN}}(D) \neq 0$ is a degree-14 polynomial defined by the initial shift register state. (Different values for $I_{\text{PN}}(D)$ generate cyclically shifted versions of the PN-15 sequence.) Note that the denominator polynomial is the polynomial (25) that defines the IRIG 106 randomizer and derandomizer. When applied to the IRIG 106 derandomizer input, the output is

$$G(D)\frac{I_{\text{PN}}(D)}{1 + D^{14} + D^{15}} + I_d(D) = I_{\text{PN}}(D) + I_d(D) \tag{31}$$

Because both $I_{\text{PN}}(D)$ and $I_d(D)$ are degree-14, all bits after the 15-th bit are zero. This is why the derandomizer output is all zeros when the PN-15 sequence is the input.

We have developed a mathematical model for the randomization and derandomization process and used this model to understand the conditions required to recover data when randomization and derandomization have been applied in the reverse order. The mathematical model uses polynomials and is based on the concept of generating functions. The "punch line" is Equation (29) which shows that the initial shift register conditions of the randomizer (second step) must be the same as the initial shift register conditions of the derandomizer (first step).

If a telemetry engineer is presented only with the derandomizer output, the practical issue is that most of the time, the initial shift register conditions are unknown. The initial shift register conditions might be unknown because the derandomizer shift registers were not reset when the data were applied. Or the derandomizer shift registers were reset, but the derandomizer output was not recorded until some time after the input data were applied — in this case the initial conditions are the 15 (unknown) data bits prior the input bit at the time the output started recording. The solution to this problem is relatively straight forward: there are $2^{15} = 32,768$ possible initial states. Each of the possible initial conditions can be applied to the randomizer. Only one of those possible initial conditions will produce data that contains valid frame synchronization words. While 32,768 might seem like a large number, it is not for a computer program.

ACKNOWLEDGEMENTS

We express our thanks and appreciation to those without whom this paper would not exist: Mr. Anthony Wirz (NSWC Corona) for bringing this issue to our attention and defining the problem; Mr. Robert Selbrede (JT3 AFFTC) for loaning us the hardware we used in our initial experiments and for many conversations that helped us understand how randomizers and derandomizers are used; Mr. Gene Law (retired, NAVAIR, Pt. Mugu) for providing historical background and context; Mr. Matthew Manwaring (BYU) for conducting the initial hardware experiments that revealed the connection between success and the initial conditions.

APPENDIX: CODE LISTING

```
#include <iostream.h>
using namespace std;

void randomize(string inputFile, string outputFile)
{
    FILE *in, *out;
    char chIn; //Character read in from file.
    int  chOut; //Value of output bit.
    int  temp;  //Used to store integer value of ASCII char read from file.
    int shiftReg[3] = {1,0,0}; //Size of shift register with initial values.

    if((in=fopen(inputFile.c_str(), "r"))==NULL)    // .c_str() convert to char*
        printf("Cannot open input file.\n");

    if((out=fopen(outputFile.c_str(), "w"))==NULL)    // .c_str() convert to char*
        printf("Cannot open output file.\n");

    //Get symbol in and convert from ASCII to int.
    chIn = getc(in);
    temp = chIn - 48;

    cout << endl << "Randomized Data = ";
    while(chIn!=EOF){
        if(temp == 1 || temp == 0)
        {
           //Next 4 lines perform shift register operations.
            chOut = (shiftReg[1]^shiftReg[2])^temp;
            shiftReg[2] = shiftReg[1];
            shiftReg[1] = shiftReg[0];
            shiftReg[0] = chOut;

            putc(chOut + 48, out); //Convert number to ASCII and write to file.
            cout << chOut;   //Print number to console.
        }
        chIn = getc(in);  //Get next symbol from file.
        temp = chIn - 48; //Convert from ASCII to int.
    }
    cout << endl;
    fclose(out);
    fclose(in);
}

void derandomize(string inputFile, string outputFile)
{
    FILE *in, *out;
    char chIn; //Character read in from file.
    int  chOut; //Value of output bit.
    int  temp;  //Used to store integer value of ASCII char read from file.
    int shiftReg[3] = {1,0,0}; //Size of shift register with initial values.
```

```
    if((in=fopen(inputFile.c_str(), "r"))==NULL)
        printf("Cannot open input file.\n");

    if((out=fopen(outputFile.c_str(), "w"))==NULL)
        printf("Cannot open output file.\n");

    //Get symbol in and convert from ASCII to int.
    chIn = getc(in);
    temp = chIn - 48;

    cout << endl << "Derandomized Data = ";
    while(chIn!=EOF){
        if(temp == 1 || temp == 0)
        {
            //Next 4 lines perform shift register operations.
            chOut = (shiftReg[1]^shiftReg[2])^temp;
            shiftReg[2] = shiftReg[1];
            shiftReg[1] = shiftReg[0];
            shiftReg[0] = temp;

            putc(chOut + 48, out); //Convert number to ASCII and write to file.
            cout << chOut;   //Print number to console.
        }
        chIn = getc(in);  //Get next symbol from file.
        temp = chIn - 48; //Convert from ASCII to int.
    }
    cout << endl;
    fclose(out);
    fclose(in);
}

int main(int argc, char *argv[])
{
    string inputFile, inputFile2, outputFile;
    int menuSelection, numberOfBits;
    while(1){
        cout << endl << "Please select from the below options." << endl;
        cout << "1. Randomize Data" << endl;
        cout << "2. Derandomize Data" << endl;
        cout << "3. Exit" << endl;
        cout << ">";
        cin >> menuSelection;

        if(menuSelection==1)
        {
        //Randomize the data
        cout << endl << endl << "RANDOMIZE" << endl;
        cout << "Please enter the input filename:";
        cin >> inputFile;
        cout << "Please enter the output filename:";
        cin >> outputFile;
        randomize(inputFile, outputFile);
        }
```

```
        else if(menuSelection==2)
        {
        //Derandomize the data
        cout << endl << endl << "DERANDOMIZE" << endl;
        cout << "Please enter the input filename:";
        cin >> inputFile;
        cout << "Please enter the output filename:";
        cin >> outputFile;
        derandomize(inputFile, outputFile);
        }
        else if(menuSelection==3)
        {
        return EXIT_SUCCESS;
        }
    }
}
```