evolution, including equalization, and Qureshi [1985] gives an excellent development of adaptive equalization and digital receiver structures.

8.7 Data Scramblers

The performance of data transmission systems must be independent of the specific bit sequence being transmitted. If allowed to occur, repeated bit sequences can cause wide variations in the received power level as well as difficulties for adaptive equalization and clock recovery. Since all these problems are eliminated if the bit sequence is "random" (has no discernible pattern), many modems employ a data scrambler to produce a pseudorandom sequence for any given input bit sequence. The scrambler usually takes the form of a shift register with feedback connections, while the unscrambler is a feedforward-connected shift register. The following example clearly illustrates the operation of data scramblers.

EXAMPLE 8.7.1

A data scrambler and unscrambler are shown in Fig. 8.7.1. The scrambler operates in the following fashion. The initial shift register contents are arbitrary but unspecified and fixed to be the same in both the scrambler and unscrambler. The first bit in sequence \( s_1 \) (note that the subscript does not indicate a time sequence here but simply denotes bit sequence number 1) is summed modulo-2 with the modulo-2 sum of locations 2 and 5 in the shift register. This sum becomes the first bit in bit sequence \( s_2 \). As this bit is presented to the channel, the contents of the shift register are shifted up one stage as follows: 5 \( \rightarrow \) out, 4 \( \rightarrow \) 5, 3 \( \rightarrow \) 4, 2 \( \rightarrow \) 3, 1 \( \rightarrow \) 2. The first bit in \( s_2 \) is also placed in shift register stage 1. The next bit of sequence \( s_1 \) arrives, and the procedure is repeated.

![Figure 8.7.1](image-url) Data scrambler and unscrambler for Example 8.7.1.
### Table 8.7.1 Scrambled Bit Sequence for Example 8.7.1

<table>
<thead>
<tr>
<th>Time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$s_2$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The unscrambler operates as follows. The initial contents of the shift register are fixed. When the first bit of sequence $s_2'$ arrives, this bit is summed mod-2 with the mod-2 sum of the initial values of stages 2 and 5. This sum then becomes the first bit of sequence $s_3$. At this time instant the contents of the shift register are shifted up one as follows: 5 → out, 4 → 5, 3 → 4, 2 → 3, 1 → 2. The first bit of sequence $s_2'$ is then put in stage 1, and the next bit in $s_2'$ is presented to the unscrambler. The procedure is then repeated.

As an illustration, consider the input sequence $s_1$ and the initial shift register contents shown in Table 8.7.1 for the scrambler of Fig. 8.7.1. Following the procedure previously outlined for the scrambler, we can compute the scrambler output sequence and shift register contents as shown in the table. Note that in Table 8.7.1, the output bit at any time instant is computed from the current input bit and the current shift register contents. A shift then takes place and the shift register contents at the next time instant are generated. By inspection of $s_2$, we can see clearly that $s_2$ is very different from $s_1$. Whether $s_2$ has any special randomness properties is not immediately evident.

Assuming that no bit errors occur on the channel, we have $s_2' = s_2$, and Table 8.7.2 illustrates the operation of the unscrambler. We compute $s_3$ from the current input bit ($s_2'$) and the current shift register contents. A shift is then performed to prepare the shift register for the next bit in $s_2'$. Note that since $s_1 = s_3$, the data are unscrambled.

### Table 8.7.2 Unscrambled Bit Sequence for Example 8.7.1

<table>
<thead>
<tr>
<th>Time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_2'$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$s_3$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
We have not said anything as yet about how the scrambler and unscrambler circuits are selected. To begin this discussion, we introduce the unit delay operator \( D \), which represents delaying the sequence by one bit. Thus, in this notation, \( Ds_2 \) represents the contents of stage 1 in the scrambler shift register, \( D^2s_2 \) represents stage 2, and so on. We can thus represent the scrambler circuit in Fig. 8.7.1 as

\[
s_2 = D^2s_2 \oplus D^5s_2 \oplus s_1, \tag{8.7.1}
\]

where the terms in \( D, D^3, \) and \( D^4 \) are absent due to the lack of feedback connections at these stages. Taking all terms in \( s_2 \) to the left side of the equality, we obtain

\[
[1 \oplus D^2 \oplus D^5]s_2 = s_1, \tag{8.7.2}
\]

or in the form of a transfer function relationship,

\[
s_2 = \frac{1}{1 \oplus D^2 \oplus D^5} s_1. \tag{8.7.3}
\]

By inspection of Eq. (8.7.3), we do not get an indication that a single bit produces a much longer sequence, as is evident in Table 8.7.1. To see this effect, we can perform synthetic division on Eq. (8.7.3) to reveal

\[
s_2 = [1 \oplus D^2 \oplus D^4 \oplus D^5 \oplus D^6 \oplus D^8 \oplus D^9 \oplus \cdots]s_1. \tag{8.7.4}
\]

Hence, from Eq. (8.7.4), we can generate the \( s_2 \) sequence from \( s_1 \) without the shift register stages.

A representation for the unscrambler can be written similarly as

\[
s_3 = [1 \oplus D^2 \oplus D^5]s'_2. \tag{8.7.5}
\]

Note that for no channel errors, \( s'_2 = s_2 \), and if we substitute \( s_2 \) in Eq. (8.7.3) into Eq. (8.7.5), we find that \( s_3 = s_1 \).

This is a special case of a more general result that if we have representations for two circuits as

\[
s_2 = F(D)s_1 \tag{8.7.6}
\]

and

\[
s_3 = G(D)s'_2, \tag{8.7.7}
\]

these two circuits can be used as a scrambler/unscrambler pair whenever

\[
F(D)G(D) = 1. \tag{8.7.8}
\]

Thus any pair of feedback- and feedforward-connected shift registers that satisfy Eq. (8.7.8) are suitable for use as a scrambler and an unscrambler pair. In Fig. 8.7.1 we chose the feedback connection as the scrambler and the feedforward device as the unscrambler. Equation (8.7.8) indicates that we could have used the feedforward connection for scrambling and the feedback connection for unscrambling.
A primary reason for the choice in Fig. 8.7.1 is bit error propagation. A single bit error into a feedforward connection affects a successive number of bits equal to the shift register length, while for a feedback connection, the effect can be much longer.

For any given shift register length $M$, there are obviously $2^M$ possible linear mod-2 sums that can be formed from its contents. All of these connections will not produce a "good" pseudorandom sequence. Furthermore, the output of any linear shift register connection with $M$ stages is periodic with a period of $2^M - 1$ or less. An output sequence with period $2^M - 1$ is a special sequence and is called a \textit{maximal-length linear shift register sequence} [Golomb, 1964]. For the linear feedback shift register connection serving as a scrambler in Fig. 8.7.1, a maximal-length sequence would have a period of $2^5 - 1 = 31$.

Further discussion of scramblers, unscramblers, error propagation, and maximal-length sequences is deferred to the problems and Appendix G.

\section*{8.8 Carrier Acquisition and Symbol Synchronization}

Carrier acquisition and symbol synchronization or timing extraction are critical to the operation of high-performance modems. In this section we discuss briefly some of the many possible approaches for obtaining this information.

For noncoherent modems, such as those employing FSK, carrier acquisition is unnecessary. Further, we have seen that the very popular DPSK systems circumvent coherent reference difficulties by using the received carrier from the immediately preceding symbol interval. Important modulation methods such as VSB and QAM do require a coherent reference to be acquired in some fashion.

Early modems that employed VSB modulation transmitted a carrier tone in phase quadrature or transmitted pilot tones at the edges of the data spectrum. For the latter method, the received signal can be represented as [Lucky, Salz, and Weldon, 1968]

$$s(t) = m(t) \cos [\omega_c t + \Delta \omega t + \theta(t) + \phi] + \hat{m}(t) \sin [\omega_c t + \Delta \omega t + \theta(t) + \phi] + \alpha \cos [\omega_L t + \Delta \omega t + \theta(t)] + \alpha \cos [\omega_H t + \Delta \omega t + \theta(t)],$$

(8.8.1)

where $\Delta \omega$ denotes frequency distortion, $\theta(t)$ denotes phase jitter, and $\omega_L$ and $\omega_H$ denote the lower and upper pilot tones. The pilot tone components in Eq. (8.8.1) are removed by narrowband filters, multiplied together, and low-pass filtered to yield a cosine term of frequency $\omega_H - \omega_L$. This term is then fed to a phase-locked loop. During the startup time a carrier tone is transmitted and fixed phase differences between the $\cos [\omega_H - \omega_L]$ term and the carrier terms are adjusted out. To complete the carrier extraction, the frequency $\omega_H - \omega_L$ is divided by an integer $N$ and multiplied by the upper pilot tone. The difference frequency component is retained so that

$$\omega_c = \frac{(N-1)\omega_H + \omega_L}{N}.$$  

(8.8.2)