PN Sequences and Scramblers

Random and Pseudo-Random Signals

Random signals have characteristics that are unpredictable. For example, the thermal noise generated by a resistor is considered random. Some statistics of the waveform, such as the average value (typically 0), the power (given by $kTBF$), and the spectrum (constant) may be known. But we may not be able to predict other characteristics of a random signal, for example whether the value of the waveform will be positive or negative at a given point in the future.

In some cases it is desirable to generate sequences that have some of the statistical properties of random signals (average, power, spectrum, etc.) but whose values *are* predictable. These types of signals are called "pseudo-random" signals. Because of their noise-like characteristics they are also called pseudo-noise (PN) signals.

A pseudo-random bit sequence (PRBS) is a two-valued PN signal. PN and PRBS signals have many important applications in communications systems. In this lecture we will study the properties of a type of PRBS called a maximal-length (ML) sequence, how these sequences are generated and one of their applications, "scrambling." Other applications include spread-spectrum systems and generating test signals.

Properties of a ML PRBS

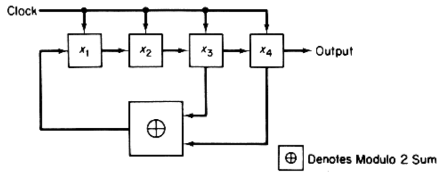
ML PRBS sequences, sometimes called m-sequences, have a number of interesting properties including:

- the sequence is called maximum-length because the period of a ML PRBS is $2^K - 1$ where K is the number of bits of state in the generator (shift register stages). This is one less than the maximum number of states of a K-bit counter.
- there are exactly 2^{K-1} ones and $2^{K-1} - 1$ zeros
- the sequence is approximately orthogonal to any shift of itself

Exercise 1: How many flip-flops would be required to generate a ML PRBS of period 16383? How many ones would the sequence have?

Generating a ML PRBS

A ML PRBS can be implemented using a shift register whose input the modulo-2 sum of other taps[1].



This is known as a linear-feedback shift register (LFSR) generator. There are published tables showing the LFSR tap connections that result in a ML PRBS generator.

If the contents of the shift register ever become all zero then all future values will be zero. This is why the generator has only $2^K - 1$ states – the state corresponding to all zeros is not allowed.

Scrambling

Much real-world data contains repetitive components. Examples include sequences of constant values such as a video image that doesn't change from one frame to the next, a document with sections that are all one level (e.g. white), or repetitive data values in a file or database.

Two possible problems are introduced by this non-random data:

- when these values are transmitted the periodic components of the signal result in peaks in the spectrum that have larger than average power. These strong discrete frequency components can cause interference.

[1]Diagram from Lindsey and Simon, *Telecommunication Systems Engineering*

- long sequences of certain values will result in a signal that may not have enough transitions to allow for clock recovery.

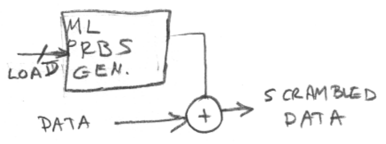
To solve these problems many communication systems use “scramblers” to remove undesirable patterns in the data. There are two types of scramblers as described below.

It is important to understand that a scrambler does not provide secrecy (encryption).

Exercise 2: Why not?

Frame-Synchronous Scramblers

The simplest type of scrambler consists of a ML PRBS generator whose output is exclusive-OR’ed with the data. These types of scramblers are called “additive” scramblers because the PN sequence is added, modulo-2, to the data.



Since the ML sequence needs to be synchronized between the transmitter and receiver, this type of scrambler is only practical for systems that have a frame structure. The state of the ML PRBS generator can be set to a specific value at the start of each frame. This value can be either a fixed value for every frame or it can be a value that is included in the frame’s preamble.

Self-Synchronizing Scramblers

Some protocols don’t use framing and operate on a continuous sequence of bits. A scrambler for such a system needs to synchronize the descrambler to the scrambler without any external information in order to recover from any loss of synchronization.

Self-synchronizing scramblers are sometimes called multiplicative scramblers because scrambling and descrambling are implemented using polynomial division and multiplication. The scrambled output, $S(x)$, is generated at the transmitter by dividing by the generator polynomial $G(x)$:

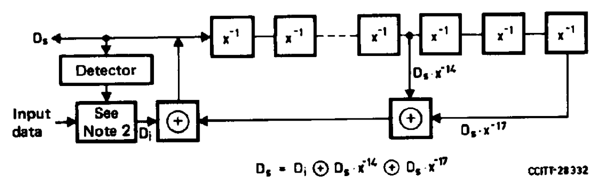
$$S(x) = \frac{M(x)}{G(x)}$$

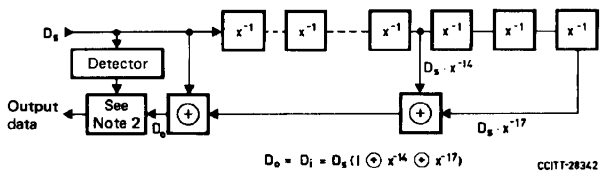
and transmitting the quotient (the remainder is ignored). The receiver de-scrambles the scrambled signal by multiplying by $G(x)$:

$$M(x) = S(x)G(x)$$

As shown in a previous lecture we can implement polynomial division and multiplication using shift registers and xor gates.

For example, the ITU-T V.22 modem specification includes the following self-synchronizing scrambler and descrambler:





One problem with self-synchronizing scramblers is that an error in the received data pattern can result in multiple errors in the de-scrambled errors. This is called error propagation.

Exercise 3: How many errors will appear in the output of a V.22 descrambler if there is one input error?

A possible problem is that certain input sequences could result in a long scrambled sequences with a constant value. However it is possible to design both the scrambler and descrambler to detect such undesirable sequences in the output and invert the next input bit when this is detected.