PN Sequences and Scramblers

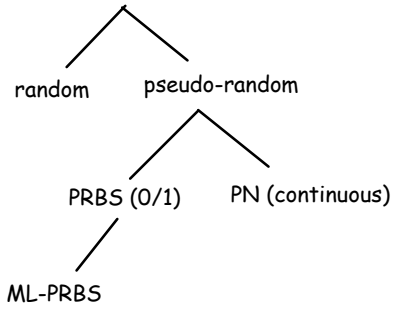
After this lecture you should be able to: distinguish between random and pseudo-random signals, classify signals as PN, PRBS, and/or ML PRBS signals according to their quantization, periodicity, mean value and maximum run lengths, draw the schematic of a LFSR ML PRBS generator, explain two reasons why scrambling may be desirable, select between scrambling and encryption based on the need for secrecy, select between additive and multiplicative scramblers based on the availability of framing information, explain the error patterns resulting from erroneous input to a self-synchronizing scrambler, and implement (draw schematic of) additive scramblers and self-synchronizing multiplicative scramblers.

Random and Pseudo-Random Signals

A random signal is one whose value cannot be predicted. An example is the thermal noise generated by a resistor or transistor. Some statistics of the noise such as the power and spectrum may be known, but we can't predict the future voltage of the the waveform.

It is sometimes useful to generate waveforms that appear random in some sense (e.g. having the same statistics) but whose values *are* predictable. These types of signals are called "pseudo-random" signals. If the pseudo-random signal is noise-like it's called a pseudo-noise (PN) signal, and if it's two-valued (0,1) it's called a pseudo-random bit sequence (PRBS).

So we have the following taxonomy of random signals:



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, learn how to generate these sequences and look at one of their applications – "scrambling." Other applications include spread-spectrum systems and the generation of test signals.

Properties of a ML PRBS

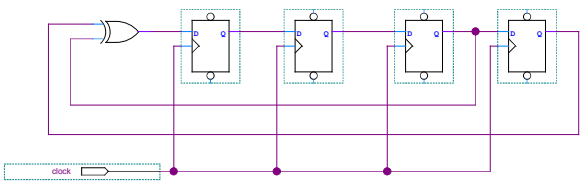
ML PRBS sequences, sometimes called m-sequences, have a number of interesting properties including:

- the sequence is called maximum-length because the sequence has a period of $2^m - 1$ where m is the number of bits of state in the generator. This is one less than the maximum number of states of an m-bit counter.
- there are 2^{m-1} ones and $2^{m-1} - 1$ zeros.
- one-half of the runs have length 1, one-quarter have length 2, etc. (except that there is one run of length m ones and one of length m-1 zeros)
- the generator runs through every possible set of states except all-zero,
- adding any (circular) shift of the sequence to itself is also an m-sequence

Exercise 1: Make a table showing the number of runs of a given length for $m = 6$. How many runs are there in total? How many bits are there in the sequence?

Generating a ML PRBS

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



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 $2^m - 1$ states – the state corresponding to all zeros is not allowed.

Exercise 2: How many flip-flops would be required to generate a ML PRBS of period 8191? How many ones would the sequence have? What is the longest sequence of 0's? How many runs of 5 ones are there?

Scrambling

Much real-world data contains repetitive components. Examples include padding/fill sequences transmitted when there is no data to be sent, a digitized image with consecutive scan lines of the same color, video that stays constant from one frame to the next, or repeated values (e.g. zeros) in a file being transmitted.

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

- Periodic components of a signal generate discrete spectral components that have larger than average power. These discrete frequency components can cause interference to wireless devices using that frequency and such a device will not get regulatory approval.
- Long sequences of certain values may result in a signal that may not have enough transitions to allow for clock recovery.

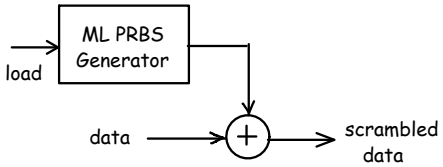
To solve these problems most communication systems use "scramblers" to remove periodicities and long constant sequences in the data. Two common types of scramblers are described below.

However, it is important to understand that a scrambling is not encryption and does not provide secrecy.

Exercise 3: 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 (i.e. it is exclusive-or'ed with the data).



Since the scrambling sequence needs to be the same at the transmitter and receiver, this type of scrambler is only practical for systems that have a frame structure that can be used to synchronise the sequences. 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 an arbitrary (typically pseudo-random) value transmitted in the frame's preamble or header.

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 so it can recover from a 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 the data by a generator polynomial $G(x)$:

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

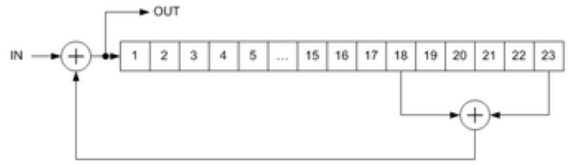
and transmitting the quotient. The division operation is carried out bit-by-bit and each step of the division results in a new scrambled bit. The receiver de-scrambles the scrambled signal by multiplying by the same generator polynomial:

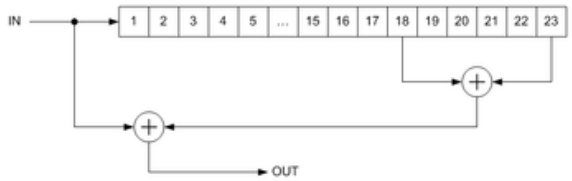
$$M(x) = S(x)G(x)$$

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.34 modem specification defines a self-synchronizing scrambler for

calling mode that uses the generating polynomial: $GPC(x) = 1 + x^{-18} + x^{-23}$ (negative powers of x are used to indicate that more negative orders correspond to more-delayed bits). The scrambler and descrambler can be implemented as shown in the following figures (the numbers in boxes are delays, not polynomial order):





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

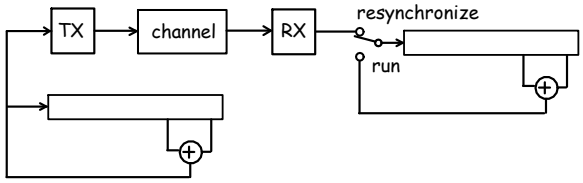
Exercise 4: How many errors will appear in the output of a V.34 descrambler if there is one input error?

Certain input sequences could set the scrambler state to zero and terminate the scrambling of long sequences of zero. Practical scramblers and descramblers count the number of consecutive '0' bits transmitted to detect this condition and invert the next bit.

PRBS Test Sequences

We can test a communication system by transmitting a PRBS sequence over the channel and comparing the received sequence to a locally-generated copy. Since the hardware to generate a very long ML-PRBS is very simple, it is practical to use long sequences for testing.

One problem that arises is how to synchronize the transmitter and a remote receiver. This can be done by loading the receiver PRBS generator's shift register with any m consecutive received bits. As long as there were no errors in these m bits then from that point on the transmit and receive generators will generate the same sequences:



If the receiver ever becomes un-synchronized with the transmitter the error rate will become very high. When this is detected at the receiver the local PRBS generator can resynchronize as above.

Exercise 5: In the diagram above, what two signals would the receiver compare to detect errors?