# EE521 Analog and Digital Communications

**Instructor:** James K Beard, PhD **Office:** Ft. Washington 115

**Email:**. jkbeard@temple.edu, jkbeard@comcast.net

**Office Hours**: Wednesdays 5:00 PM to 6:00PM

**Location:** Ft. Washington 107 **Time:** Wednesdays 6:00 PM – 8:30 PM

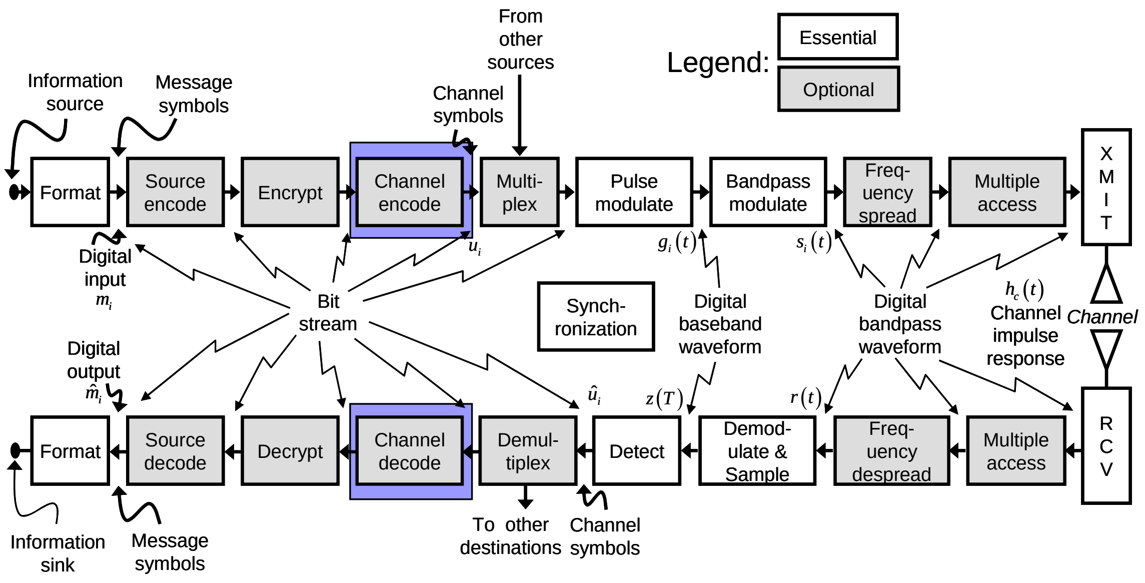
**Web Page**: http://temple.jkbeard.com

**Texts:**

- Bernard Sklar, Digital Communications, Second Edition, Prentice Hall P T R, 2001 (2004 printing), ISBN 0-13-084788-7
- Digital Communication Systems Using SystemVue, by Dr. Silage, ISBN 1-58-450850-7

## *Today's Topics*

# Channel Coding: Part 1



## 1 Homework Problems

### 1.1 Problem 4.3 Page 237

We have a noncoherent orthogonal FSK with an $E_b/N_0$ of 13 dB and a binary coherent PSK with an $E_b/N_0$ of 8 dB. We need to determine which has the best bit error rate (BER).

The equations in Table 4.1 page 219 give the BER for four common waveform and detection schemes for binary signals. Using Table 4.1, we have

$$BER_{NCOFSK}\left(\frac{E_b}{N_0}\right) = \frac{1}{2}\cdot\exp\left(-\frac{1}{2}\cdot\frac{E_b}{N_0}\right) \tag{1.1}$$

For our problem we have

$$BER_{NCOFSK}\left(13\,\text{dB}\right) = BER_{NCOFSK}\left(19.95\right) = 2.32\cdot 10^{-5} \tag{1.2}$$

For coherent PSK we have

$$BER_{CPSK}\left(\frac{E_b}{N_0}\right) = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{1.3}$$

For our problem we have

$$BER_{CPSK}\left(8\,\text{dB}\right) = BER_{CPSK}\left(6.31\right) = 1.9\cdot 10^{-4} \tag{1.4}$$

Therefore noncoherent orthogonal FSK has the better BER.

### 1.2 Problem 6.1 Page 375

Design an *(n,k)* single-parity code that will detect all 1-, 3-, 5-, and 7-error patterns in a block. Show the values of *n* and *k*, and find the probability of an undetected block error if the probability of channel symbol error is $10^{-2}$.

Sklar's Section 6.3.3.1 pages 321-322 defines single-parity codes as a parity bit added to a binary code word. Since we are required to detect up to 7 errors, or code length *k* is 7 bits, and our codeword length is one more, 8 bits, to account for the parity bit. An odd number of errors will cause the parity bit to be wrong, even if the error is the parity bit itself. Our logic is:

a) If the 7-bit message is OK or has an even number of errors and the parity is OK, but the parity bit is mis-read, then this will be detected as an error -- and we have an odd number of bit errors, counting the parity bit error.

b) b) If the 7-bit message has an odd number of bit errors and the parity bit shows an error, but the parity bit is mis-read, then we will not have a detectable error -- and we have an even number of bit errors, counting the parity bit error.

The probability of an undetected error is, from equation (6.17) page 322,

$$P_{nd} = \sum_{j=1}^{4} \binom{n}{2j} \cdot p^{2j} \cdot (1-p)^{n-2j} \tag{1.5}$$

For $p=10^{-3}$ we find the probability of an undetected error to be

$$\begin{aligned} P_{ND} &= P(2\text{ errors}) + P(4\text{ errors}) + P(6\text{ errors}) + P(8\text{ errors}) \\ &= \binom{8}{2} \cdot p^2 \cdot (1-p)^6 + \binom{8}{4} \cdot p^4 \cdot (1-p)^4 + \binom{8}{6} \cdot p^6 \cdot (1-p)^4 + p^8 \\ &= 28 \cdot (9.41 \cdot 10^{-5}) + 70 \cdot (9.61 \cdot 10^{-9}) + 28 \cdot (9.8 \cdot 10^{-13}) + 10^{-16} \\ &= 2.48 \cdot 10^{-7} \end{aligned} \tag{1.6}$$

Note that the first term sets the probability to four decimal places.

## 2 Linear Block Codes, Concluded

Today we will cover what we didn't get to last time and add some points not addressed in Sklar.

Linear block codes are *(n,k)* parity-check codes. The most general statement allows the *k* message digits to be any basis – binary, octal, decimal, hexadecimal, ASCII, etc. – but we will consider only binary codes here.

In the simplest sense, a sequence of *k*-bit binary sequences, which we will *k-tuples,* are mapped into a sequence of *n*-bit codewords or *n-tuples*. For the block codes considered here, this mapping is unique, and could be implemented with a lookup table. The extra bits reduce the data rate by a factor of the code rate, but the extra bits are used to improve the effective probability of bit error in the decoded k-tuples.

When the mapping of the k-tuples to the n-tuples can be represented by a linear transformation, we have a *linear block code*. When the message k-tuple is embedded in the encoded n-tuple, we have a *systematic* code.

## 2.1 Structured Sequences

Structured sequences replace blocks of binary data with larger blocks. We are looking at linear block codes today but most types of codes can be characterized as structured sequences. We denote the number of message bits in a block by *k* and the number of total number of bits in the encoded block including redundant bits as *n*. These codes are called *(n,k)* codes. The key parameters that characterize structured sequences are:

**Redundant bits, parity bits, or check bits** are equivalent terms for the additional number *(n-k)* of bits in the transmitted data block.

**Redundancy of the code** is the ratio *(n-k)/k*.

**Code rate** is the term for the ratio *k/n* of data bits to total bits.

Code rates of ½ or 2/3 are commonly used in linear block codes.

## 2.2 The Generator and Parity Check Matrices

Although a linear block code can in principle be implemented with a lookup table, a simple algorithm can generate linear block codes with minimal hardware and latency. The subspace property is the principle here; in principle, a set of *k* members of the subspace will form a basis set of vectors over the subspace – that is, every member of the subspace can be constructed from a linear combination of the *k* members of the basis set.

Let’s call the basis set

$$\text{Basis set:} \left\{ \underline{v}_1, \underline{v}_2, \underline{v}_3 \ldots \underline{v}_k \right\} \tag{3.1}$$

and denote the bits of the k-tuple as

$$k\text{-tuple:} \left\{ m_1, m_2, m_3 \ldots m_k \right\} \tag{3.2}$$

We can construct a set of *k* codewords by

$$\underline{u} = m_1 \cdot \underline{v}_1 + m_2 \cdot \underline{v}_2 + m_3 \cdot \underline{v}_3 + \ldots + m_k \cdot \underline{v}_k \tag{3.3}$$

We can represent this encoding operation as a vector-matrix multiplication, using the rules of logical multiplication and addition. We collect the message bits by a row vector

$$\underline{m} = \begin{bmatrix} m_1 & m_2 & \ldots & m_k \end{bmatrix} \tag{3.4}$$

where the underscore indicates that we are denoting a row vector, and we collect the basis of the subspace in a generator matrix *G*,

$$G = \begin{bmatrix} \underline{v}_1 \\ \underline{v}_2 \\ \vdots \\ \underline{v}_k \end{bmatrix} \tag{3.5}$$

and the codeword generation becomes

$$\underline{u} = \underline{m} \cdot G \tag{3.6}$$

Thus we have reduced the problem of defining a generator matrix for a block code as finding a basis set for a subspace of $2^k$ binary vectors. The algorithm to encode the k-tuples is implementation of (3.6).

We select the basis vectors so that the right hand *k* columns form an identity matrix. This allows us to partition the generator matrix G into a parity array portion *P* and an identity matrix,

$$G = \left[ P \middle| I_k \right] \tag{3.7}$$

where $I_k$ is an identity matrix of order *k*. We define the parity check matrix *H*, written here as its transpose, as

$$H^T = \begin{bmatrix} I_{n-k} \\ P \end{bmatrix} \tag{3.8}$$

Note that the parity array appears as well as an identity matrix of order *n-k*. The syndromes for single bit errors are the rows of $H^Y$.

Since

$$G \cdot H^T = \left[ P + P \right] = \left[ 0 \right] \tag{3.9}$$

then the transpose of the parity check matrix *H* is orthogonal to all valid codewords.

Selection of the basis vectors in the generator matrix *G* determines $d_{min}$ and thus the parity detection and correction capability of the code. We can begin by assigning the basis vectors so that none of them has a Hamming weight less than the desired or required $d_{min}$.. We still need to look at all the codewords to determine the $d_{min}$ of the code. We will discuss other factors in code generation next time.

# 3 Finite Fields and m-Sequences

This topic is treated in Sklar Chapter 8. I provide it here because it provides a simple basis for moving forward with the convolutional codes and demonstrates many principles that we will use in codes of several types.

## 3.1 Definition of an m-Sequence

A sequence of bits can be produced by a shift register containing memory for *k* bits and feedback to perform binary addition to various places in the shift register. There are many ways to do this, but it is apparent that the shift register can be in any one of $2^k$

staetes, one of which is all zeros. A feedback scheme that produces all $2^k$-1 states produces a maximal length sequence, or m-sequence.

## *3.2 Finite Fields of Order $2^k$-1 Produce m-Sequences*

There is a simple theory that produces simple results that helps us understand m-sequences and their properties and use. This is Galois finite field theory, and the specific part of this theory that we are interested in for error correcting codes is the theory of finite fields of $2^k$ elements. These fields are denoted as $GF(2^k)$, or $GF(q)$ where $q=2^k$.

Finite fields of order $2^k$ have elements that are vectors of $k$ bits (mathematicians and others have other representations that they find convenient, but we will use this here). These are the familiar vectors that we use to organize signals for linear block codes, basis vectors, and codewords for linear block codes. Here, to help us visualize finite field theory, we will denote them as polynomials, with the bits being the coefficients of the polynomials. Thus the $k$ bits are represented as polynomials of order $k-1$ with coefficients that are 1 and 0. This representation of binary signals as polynomials is one that you will see in many areas of advanced signal processing theory.

Arithmetic using these vectors is done using the rules of polynomial arithmetic, except that we use binary arithmetic like we have been doing with linear block codes instead of ordinary arithmetic.

## *3.3 Construction of GF(q)*

The field is constructed by defining arithmetic modulo a *generating polynomial*, sometimes called a *primitive polynomial* or an *irreducible polynomial*. This is necessary to define multiplication so that the result is a polynomial is of order $k-1$ and thus is part of the set. We start with an element, $x$, and all the elements of the set are produced by successive powers of $x$. Multiplication by $x$ is analogous to a left-shift by one bit in the shift register. The generating polynomial is of order $k$ so that any term of order $k$ or above in a polynomial product is mapped to lower order polynomials. The generating polynomial must have a nonzero coefficient for $x^0$ because if it did not, that bit would become zero after the first multiplication by $x$ and would never be set again, so that we could not obtain all $2^k$ states. At least one other bit must be set in the generator polynomial because if only the $x^0$ bit were set, we would have only $k$ states, not $2^k$. Other than that, there are no rules that assure us that we can simply write down a generator polynomial; a trial must be made to ensure that powers of $x$ do indeed repeat only after $2^k$-1 iterations. Sklar has a table of some generator polynomials for $k$ from 3 through 24 as Table 8.1 page 448. Interestingly he has none for $k$=2; we may add $1+x+x^2$ as a primitive polynomial for that case. For some values of $k$ there is more than one generator polynomial; Sklar lists only one. This is sufficient because Galois finite field theory tells us that any one of them will do because there is a one-to-one correspondence between the elements of any one Galois field of a given order and any other of the same order, in which all the arithmetic operations track.

In mathematics, a field is defined as a set of values that have these properties:

- Addition and multiplication are defined.

- Addition and multiplication are commutative and associative.
- There is a zero; multiplication by zero yields zero, and addition of a value to zero produces a sum equal to the value, unchanged – an additive identity.
- There is a one; multiplication of a value by one produces the value, unchanged – a multiplicative identity.
- There is a negative value for every value; addition of a value to its negative produces zero.
- There is a reciprocal for every value except zero; multiplication of a value by its reciprocal produces one.

The last two mean that subtraction and division are also defined, and that they are commutative and associative.

### 3.4 Division in GF(q)

The field that we have constructed has these properties; we need only note that the polynomial with all zero coefficients meets the requirements that there be a zero and find a reciprocal. We can find a reciprocal by noting that, for any element $\alpha$ of *GF(q),*

$$\alpha^q = \alpha \tag{3.10}$$

because sequences of powers of $\alpha$ must either step through all possible *q-1* nonzero values, or through a cycle that is a number that divides *q-1*. The number of values of the set that are produced by a sequence of powers is called the *order* of the element. The order of any element divides *q-1*. When the order is *q-1* we call that element a *primitive element* or *primitive root* of *GF(q).* We have defined our construction so that the polynomial *x* is a primitive element. Powers of *x* that do not divide *q-1* are also primitive elements.

From (3.10) we see that, for any element except the zero element,

$$\alpha^{q-1} = 1 \tag{3.11}$$

for any element of *GF(q)*. Thus, we can define the reciprocal of any element except zero as

$$\alpha^{-1} = \alpha^{q-2} \tag{3.12}$$

We will find that using feedback registers in generating convolutional codes produces overlapping code vectors that are members of *GF(q)* where $q=2^k$. We will use division to recover the message bits.

## 4 Assignment

### 4.1 Reading

Sklar, Chapter 6, Sections 6.7-6.9, pages 356-374.

## 4.2 SystemView

This assignment is part of your Quiz 2 and will be 25% of that grade.

Generate a test signal over the speech band, add some noise to it, sample it to 8 bits and convert it to a serial bit stream, then convert it back to PAM and scale to the original scale. Before you begin, block out your system and decide what the SystemView sample rate must be, and set it. Increase the number of samples using the power-of-two spin button until the end time is 1 second or more.

- Use a frequency sweep signal, from 30 Hz to 3000 Hz..
- Add noise, initially at 17 dB SNR.
- Digitize to a serial bit stream with these tokens:
    - Quantize to 8 bits, using the quantizer from the function library. Make sure that the input voltage range accommodates the voltage of the input, plus noise.
    - Use the sampling operator to sample at the engineering Nyquist rate for the signal chosen.
    - Use a custom algebraic function to turn the sampled, quantized output into a Symbol. This will require that the signal be shifted in level and scaled to become an unsigned integer.
    - Use the symbol-to-bit converter from the Comms library to convert the symbol to a bit stream.
- Convert the bit stream back to a symbol using the bit-to-symbol converter from the comms library.
- Use a custom function to re-scale the symbol to the original scale of the signal plus noise.
- Display the output in a system sink.

You will send me your files by e-mail or have them ready for me to load on my laptop when class starts. I will have a USB flash drive and a CD-ROM drive to accomplish this. I don't have a floppy drive, so if you can't bring your laptop with the file on it, or you can't burn a CD-ROM, send the file by e-mail.

You will write a one or two sentence description of each token. This description will tell me

- What the token is -- which token is used from which library.
- What the token does -- an algebraic equation or one or two words..
- What the data format is on the input.
- What the data format is on the output.

Data format parameters for token inputs and outputs will include

- Signal type -- analog, digital 0-1, digital (-1)-(+1), symbol unsigned integer, etc.
- Voltage range (-1v to +1v, 0 to 255 v, etc.) .
- Sample rate (SystemView sample rate, 6600 Hz, etc.) .

I'll entertain questions by e-mail.  You can send me your file and ask what's wrong just as always.