EE521 Analog and Digital Communications

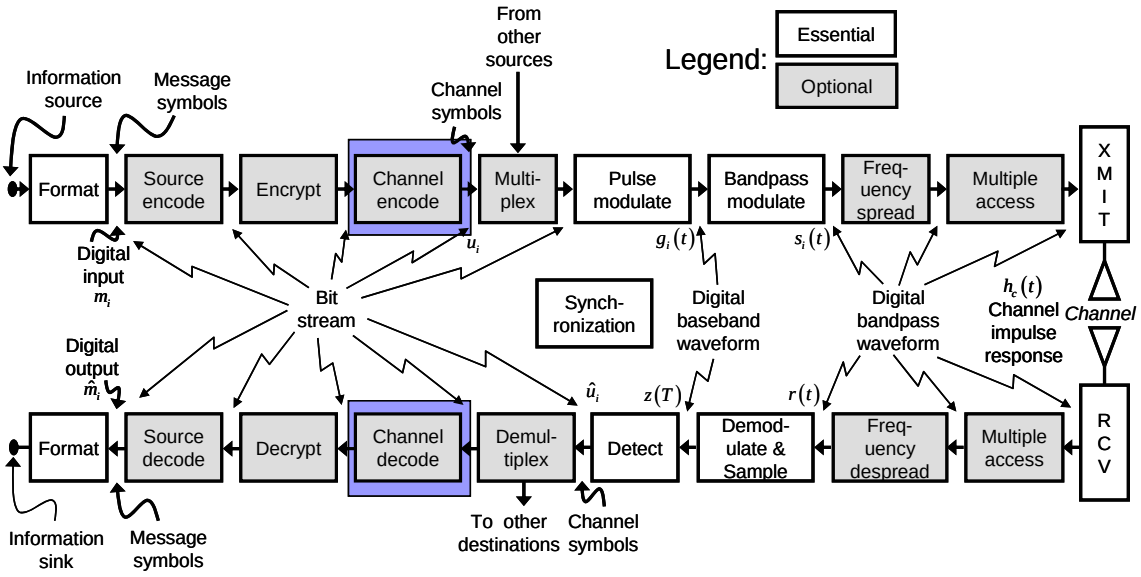
Instructor: James K Beard, PhD **Office:** Ft. Washington 115
Email:. jkbeard@temple.edu, jkbeard@comcast.net
Office Hours: Wednesdays 5:00 PM to 6:00PM
Location: Ft. Washington 107 **Time:** Wednesdays 6:00 PM – 8:30 PM
Web Page: http://temple.jkbeard.com

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 Waveform Coding and Structured Sequences

This is actually two topics. Waveform coding is replacing a simple pulse with a complex pulse. If the complex pulse is one of a set of orthogonal binary codes, then other systems can use the same band using other codes and each will look like noise to the other, and they can share the band. Another key advantage is that the signal uses a lot more bandwidth, so problems from narrow-band interference and from selective fading are lessened.

1.1 Codewords as Binary Vectors

An easy understanding signal processing of code words is made possible by working with them as vectors of binary quantities. We will define the operations of addition and multiplication, and the metrics we use most, Hamming weight and Hamming diatance.

Addition of codewords: The sum of codewords *U* and *V* is the bit-by-bit logical exclusive OR, or XOR. This can also be viewed as the bit-by-bit sum without carrying; note that this is why the exclusive OR is also called the half-add.

Multiplication of codewords: The product of codewords *U* and *V* is the bit-by-bit logical AND.

The Hamming weight and Hamming distance are metrics that we use to evaluate the orthogonality and performance of binary codes. The definitions are simple:

Hamming weight: The Hamming weight $w(U)$ of a code word U is the number of nonzero elements; for a binary code this is the number of ones in the sequence of ones and zeros.

Hamming distance: The Hamming distance $d(U,V)$ between two codewords *U* and *V* is the number of elements in which they differ. Thus the Hamming distance is the Hamming weight of the sum of the codewords:

$$d(U,V) = w(U+V) \tag{1.1}$$

We see that the Hamming distance between a codeword and the zero vector is the Hamming weight of the codeword.

A key simplification is the definition of the condition of orthogonality, which becomes the condition that the Hamming distance between distinct codewords in a set is half the length of the codewords. With this definition, the orthogonality of the columns (or rows) of the Hadamard matrix becomes evident on inspection.

1.2 Orthogonal, Antipodal, and Biorthogonal Codes

The definition of orthogonality is that codes shall produce a zero output of a matched filter. For codewords with total energy *E* and duration *T*, we define the output of matched filter *i* with pulse *j* as z_{ij},

$$z_{ij} = \frac{1}{E} \cdot \int_0^T s_i(t) \cdot s_j(t) \cdot dt \tag{1.2}$$

With the definition of codewords of *M* bits as binary vectors U_i, we can define z_{ij} as

$$z_{ij} = 1 - \frac{d(U_i,U_j)}{M} = 1 - \frac{w(U_i+U_j)}{M} \tag{1.3}$$

An *orthogonal* set of codewords is one for which

$$\text{Orthogonal: } z_{ij} \begin{cases} = 1, i = j \\ = 0, i \neq j \end{cases} \tag{1.4}$$

An *antipodal* set of codewords is one for which we have

$$\text{Antipodal: } z_{ij} \begin{cases} = 1, i = j \\ = -1, i \neq j \end{cases} \tag{1.5}$$

The principal antipodal signal that we study here is the BPSK signal, which has only two codewords.

A *biorthogonal* set of codewords is one that consists of a combination of orthogonal and antipodal signals. In general, they satisfy the condition of orthogonality except that each codeword has another codeword in the set with which it is biorthogonal.

A set of *M* biorthogonal codewords can be constructed from *M/2* orthogonal codewords by augmenting each binary vector with its complement. If we have an *M* by *M* square matrix of binary quantities and the rows are orthogonal, we can build a matrix with $2 \cdot M$ rows that are biorthogonal by adding the complement of the original matrix as a second set of *M* rows:

$$B_k = \begin{bmatrix} H_{k-1} \\ \hline H_{k-1} \end{bmatrix} \tag{1.6}$$

The overbar denotes the complement. The rows of B_k are biorthogonal:

$$\text{Biorthogonal: } z_{ij} \begin{cases} = 1, i = j \\ = -1, |i - j| = \dfrac{M}{2} \\ = 0, i \neq j \text{ and } |i - j| \neq \dfrac{M}{2} \end{cases} \tag{1.7}$$

A very important advantage of biorthogonal codes is that this simple relaxation of the rule of orthogonality results in an advantage of two to one in code length. Note that the matrix B_k has half as many columns as H_k and that the rows of B_k comprise a set of $M=2^k$ biorthogonal codewords, each of length $M/2$. This compares to the rows of H_k which comprise a set of M codewords, each of length M. Thus the use of the rows of B_k provides M codewords with half the bits.

1.3 Hadamard Codes

A good example of a binary orthogonal code, and one that is used often in today's digital communications, is the Hadamard codes. A Hadamard code is a square matrix of zeros and ones. A Hadamard code of order k is denoted by $\mathrm{H_k}$. They are defined recursively as follows.

Initialization

$$\mathrm{H}_0 = [0] \tag{1.8}$$

Recursion

$$\mathrm{H}_k = \begin{bmatrix} \mathrm{H}_{k-1} & \mathrm{H}_{k-1} \\ \mathrm{H}_{k-1} & \overline{\mathrm{H}_{k-1}} \end{bmatrix} \tag{1.9}$$

Note that rows of the right half of the Hadamard matrix is biorthogonal, and that inclusion of the left half results in orthogonal rows. The first few Hadamard matrices are

$$\mathrm{H}_1 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\mathrm{H}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

$$\mathrm{H}_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{1.10}$$

Note that the Hadamard code matrices are symmetrical, so that the rows are equal to the respective columns. The Hadamard code bit sequences are the Hadamard codes, and can be seen to be orthogonal. The recursion relation can be used with an inductive argument to prove that Hadamard codes of order *k* provide a sequence of *M* orthogonal bit sequences, where

$$M = 2^k \tag{1.11}$$

A binary pulse that is replaced by a Hadamard code of order *k* will have its bandwidth multiplied by *M*. The probability of bit error changes, and is now more complex. We can bound it above:

$$P_B(M) \le \frac{M}{2} \cdot Q\left(\sqrt{\frac{E_s}{N_0}}\right) \tag{1.12}$$

where E_s is the energy per Hadamard code word,

$$E_s = k \cdot E_b \tag{1.13}$$

This is the energy in a transmitted pulse before encoding with a Hadamard code.

Structured sequences result from replacing the character or token stream with another

2 Types of Error Control

Error control is detection and correction of bit errors. There are two fundamental types:

- Error detection and retransmission, usually using parity bits; when an error is detected, the receiver requests that the data packet be re-transmitted.
- Forward error correction (FEC) uses error-correcting codes; redundant bits are used to detect errors, and when an error is detected, the redundant bits are used to correct it.

The type of terminal connectivity is designed along with the error control. There are three types of terminal connectivity:

- Simplex, in which data passes only in one direction.
- Half-duplex, in which data can pass in both directions, but only one direction at a time.
- Full-duplex, in which data can pass in both directions simultaneously.

3 Structured Sequences

Structured sequences replace blocks of binary data with larger blocks. 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.

4 Linear Block Codes

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.

4.1 Linear Block Codes Map to Vector Subspaces

Note that the block code consists of 2^k codewords out of the 2^n possible n-tuples. The set of all possible n-tuples, V_n, is the *vector space* over the binary field. A vector space *S* must have these two properties:

- The all-zeros vector is in *S*.
- The sum of any two vectors in *S* is also in *S*; this is called the *closure property*.

Linear block codes are defined so that the set of 2^k codewords that are used by the block code is a valid subspace. A code which maps into a set of n-tuples that is not a valid subspace is not a valid linear block code.

4.2 The Generator Matrix

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

$$\text{Basis set: } \{\underline{v}_1, \underline{v}_2, \underline{v}_3 \ldots \underline{v}_k\} \tag{4.1}$$

and denote the bits of the k-tuple as

$$k\text{-tuple: } \{m_1, m_2, m_3 \ldots m_k\} \tag{4.2}$$

We can construct a set of *k* codewords by

$$\underline{u} = m_1 \cdot \underline{v}_1 + m_2 \cdot \underline{v}_2 + m_3 \cdot \underline{v}_3 + \ldots + m_k \cdot \underline{v}_k \tag{4.3}$$

where the multiplication and addition is in the sense described in Section 1.1; multiplication is a logical AND, and bits are added in pairs with the result of each addition an XOR of the two bits without carry.

We can represent this encoding operation as a vector-matrix multiplication, using the rules of logical multiplication and addition of Section 1.1. We collect the message bits by a row vector

$$\underline{m} = \begin{bmatrix} m_1 & m_2 & \ldots & m_k \end{bmatrix} \tag{4.4}$$

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

$$G = \begin{bmatrix} \underline{v}_1 \\ \underline{v}_2 \\ \vdots \\ \underline{v}_k \end{bmatrix} \tag{4.5}$$

and the codeword generation becomes

$$\underline{u} = \underline{m} \cdot G \tag{4.6}$$

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 (4.6).

4.2.1 Properties of a Basis Set

A basis set of 2^k vectors, each an n-tuple, has these properties:

- None of the vectors can be found as a linear combination of the others; this property is *linear independence* and is the opposite of the property of closure, and is necessary for the code to work.

- None of the vectors is the all-zero vector; this actually follows logically from the first property.

Note that a basis set is not unique to a subspace. In fact, a basis set is any *k* n-tuples that are linearly independent.

4.2.2 Basis Sets for Systematic Codes and the Parity Check Matrix

A systematic block code will have an identity matrix for a portion of the generator matrix. The convention we will use here is that the rightmost *k* columns of the generator matrix are an order *k* identity matrix for a systematic block code. The leftmost *n-k* columns of the generator matrix for a systematic block matrix are called the *parity array portion P* of the generator matrix. An example of a generator matrix *(6,3)* systematic code in the codeword generation algorithm is

$$\underline{u} = \underline{m} \cdot \left[\begin{array}{ccc|ccc} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \qquad \begin{matrix} P & \quad & I_3 \end{matrix} \tag{4.7}$$

Note that the right hand three bits will be the message bits, so that only the parity generator portion of the generator matrix need be implemented as a binary arithmetic vector-matrix multiplication in the codeword generator algorithm.

4.2.3 The Parity-Check Matrix and the Syndrome

We define a matrix *H* that we call the *parity-check matrix* from the parity generator portion of the generator matrix:

$$H = \left[I_{n-k} \middle| P^T \right] \tag{4.8}$$

Because of our convention of using row vectors and implementing vector-matrix multiplication as left-multiplying a matrix by a row vector, we will need the transpose of the parity check matrix

$$H^T = \begin{bmatrix} I_{n-k} \\ P \end{bmatrix} \tag{4.9}$$

We find that the product

$$\begin{aligned} \underline{u} \cdot H^T &= \underline{m} \cdot G \cdot H^T \\ &= \underline{m} \cdot \begin{bmatrix} P & I \end{bmatrix} \cdot \begin{bmatrix} I \\ P \end{bmatrix} \\ &= \underline{m} \cdot \left[P + P \right] = \underline{0} \end{aligned} \tag{4.10}$$

This means that, at the receiver, we can left-multiply a received n-tuple into H^T and conclude that the received n-tuple is correct if the result is zero. We call the product, a vector with *k* components, a *syndrome*.

4.2.4 Syndrome Testing

Suppose that we have a bit error on receive. We denote a vector that is all zeros except for a 1 in the bit error position as $\underline{e}$. Then the received vector $\underline{r}$ is

$$\underline{r} = \underline{U} + \underline{e} \tag{4.11}$$

When we find the syndrome,

$$\underline{s} = \underline{r} \cdot H^T = \left(\underline{u} + \underline{e}\right) \cdot H^T = \underline{e} \cdot H^T \tag{4.12}$$

we see that we can tabulate the syndromes for each of the *n* bit errors and look for that product to tell which bit is in error. For these vectors to be unique, we require that

- No column of *H* can be all zeros, because this would mean that we could not detect a bit error in $\underline{r}$ at that position.
- All columns of *H* must be unique so that the table of syndromes will not have duplicates. Otherwise bit errors in the positions of the duplicated columns would be indistinguishable.

Note that the syndrome has *n-k* bits and thus 2^{n-k} possible values. One of them, all zeros, corresponds to no errors, and the rest correspond to correctable errors. In our example, we can correct seven errors, but we have only six possible one-bit errors. This means that we can correct one case of two bit errors. This is not unique, and any pair of bits is equally likely to be in error, so we can ignore this capability or we can select one case that we will use.

4.3 Error-Detecting and Correction Capability

4.3.1 Minimum Distance of a Linear Code

The minimum Hamming distance between any two codewords of a subspace V_n that is used in a *(n,k)* code is called the *minimum distance* of the code, and is denoted by d_{min}. This defines the pairs of codewords that can be undetectably confused, and the minimum number of bit error that is necessary to produce this ambiguity.

Finding d_{min} is done as follows. We know that, for at least one pair of codewords $\underline{u}$ and $\underline{v}$,

$$\begin{aligned} d_{\min} &= d\left(\underline{u}, \underline{v}\right) \\ &= d\left(\underline{u} + \underline{v}, \underline{0}\right) \\ &= w\left(\underline{u} + \underline{v}\right) \end{aligned} \tag{4.13}$$

Since the subspace is closed, $\underline{u}+\underline{v}$ is in the subspace, and d_{min} is the Hamming weight of a member of the subspace. Thus, d_{min} is the smallest Hamming weight in the subspace used for the code, exclusive of the zero vector.

4.3.2 Number of Bit Errors that Can Be Corrected

If detection of errors is the only goal, we have shown in the previous Section that we can detect up to *e* bit errors:

$$e = d_{\min} - 1 \tag{4.14}$$

If our error correction corrects by selecting the element of the subspace that has the smallest Hamming distance from the received codeword, then the maximum number of bit errors that can be detected is

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor \tag{4.15}$$

where the square bracket here denotes truncation, or dropping any fractional part to obtain an integer. A code with a d_{min} of three can correct one bit error and detect two bit errors, and is called a "correct one, detect two" code.

Any received code word that is not in the subset used in the code will be detected as an error. Thus any of the 2^n-2^k codewords that are not in the subset will be detected as errors. The proportion of errors that are detected is $1-1/2^{n-k}$. As $n-k$ increases, this proportion rapidly approaches 100%, so excellent error detection is available for high code rates as n increases.

When d_{min} is odd, the syndrome from $t+1$ bit errors is ambiguous, e.g. more than one pair of bit errors can produce the same syndrome. Although we can assign one of the corrections to the syndrome, we have a maximum probability that the correction is the right one of 50%.

4.4 Usefulness of the Standard Array

The *standard array* is a matrix which has as its top row the codewords in the subspace, beginning with the zero vector. The first column is all of the error vectors e that can be corrected. Thus the first column is a codeword heading entries of the codeword plus the correctable error vectors. Subsequent columns follow this example. The result is an array of all 2^n possible n-uples that can be received. Each row, called a *coset*, consists of all possible codewords plus error for a particular error vector e_j. There are 2^{n-k} cosets. Each coset corresponds to only one syndrome S_j.

We have seen that examples for n up to about 10 are easy to construct and understand, and thus make good examples or samples for working with test implementations. However, we have found that for good error detection and correction performance, we need large n, which would make actual implementation of the standard array impractical. However, the standard array is an excellent basis for analysis of properties of block codes.

One trade relationship that is not yet defined is how to define k for a specified or required d_{min}. One equation relating k and t is the *Hamming bound*, given by

$$2^{n-k} \geq \sum_{j=0}^{t} \binom{n}{j} \tag{4.16}$$

The Hamming bound is found from the standard array; see Peters and Weldon in the References for a derivation. Note that the sum can be found using the incomplete beta function,

$$I_p(a, n-a+1) = \sum_{j=a}^{n} \binom{n}{j} \cdot p^j \cdot (1-p)^{n-j} \tag{4.17}$$

where

$$I_x(a,b) = \frac{B_x(a,b)}{B(a,b)} = 1 - I_{1-x}(b,a) \tag{4.18}$$

and *B(a,b)* is the incomplete beta function,

$$B_x(a,b) = \int_0^x t^{a-1} \cdot (1-t)^{b-1} \cdot dt \tag{4.19}$$

Thus

$$\sum_{j=0}^{t} \binom{n}{j} = 2^n \cdot I_{0.5}(n-t, t+1) \tag{4.20}$$

See Abramowitz & Stegun p. 263 in the References for definitions and suggestions on computing the Incomplete Beta Function.

The Hamming bound is an approximation that is most accurate for high code rate codes. For lower code rates, we need the *Plotkin bound*,

$$d_{\min} \le \frac{n \cdot 2^{k-1}}{2^k - 1} \tag{4.21}$$

This limit is also an approximation; see Peters and Weldon for a derivation.

5 Example of a (6,3) Linear Block Code

5.1 Defining the Basis Set and Generator Matrix

We will examine this code in an Excel spreadsheet file. This file is available on the web site on the EE521 page as Linear_Block_Code.xls.

We will use the generator matris $\underline{u}$ of (4.7). The parity check matrix is

$$H = \left[I_3 \middle| P^T \right] = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right] \tag{5.1}$$

5.2 The Codewords

The code words are shown in the table below.

Table 1 Codewords

Codewords	b2	b1	b0	c5	c4	c3	c2	c1	c0	d
0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	1	0	0	1	3
2	0	1	0	0	1	1	0	1	0	3
3	0	1	1	1	1	0	0	1	1	4
4	1	0	0	1	1	0	1	0	0	3
5	1	0	1	0	1	1	1	0	1	4
6	1	1	0	1	0	1	1	1	0	3
7	1	1	1	0	0	0	1	1	1	3

From the table, the minimum Hamming distances from the zero vector, d_{MIN}, is 3. Thus this is a correct 1, detect 2 code. Note that 2^{n-k} is 8, so we have only 8 codewords out of a possible 2^n or 64, so that only 1 of 8 possibilities is a legitimate code word. This means that when the BER is high and more than two bit errors is possible, the mean probability that three or more bit errors will result in a legitimate codeword with no error detected is about 1 in 8 for this code. In general this probability is $1/2^{n-k}$. This means that a high BER is easily detected and the BER estimated from the percentage of detected codeword errors.

5.3 Syndromes and Error Correction

We will generate the syndromes from H^T:

$$H^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \tag{5.2}$$

The syndromes are shown below in the table.

Table 2 Syndromes

Bit Errors	b5	b4	b3	b2	b1	b0		Syndromes			Value
0	0	0	0	0	0	1		1	0	1	5
1	0	0	0	0	1	0		0	1	1	3
2	0	0	0	1	0	0		1	1	0	6
3	0	0	1	0	0	0		0	0	1	1
4	0	1	0	0	0	0		0	1	0	2
5	1	0	0	0	0	0		1	0	0	4
4,0	0	1	0	0	0	1		1	1	1	7

5.4 Error Correction

Error correction is done by left-multiplying the parity check matris H^T as given by (5.2) by the received code vector and examining the syndrome, then checking the syndrome against those listed in Table 2. For the purposes of our example, we have produced the value of the syndrome interpreted as a three-bit binary number under the heading “Value” iu the table.

In firmware executed by even the simplest ALU of 3 or more bits, the operation could be a table scan. For an FPGA programmed for maximum speed without an ALU, a parallel XOR over the syndromes would provide the correction bit.

Note that we have eight possible syndromes but only six bits. One syndrome is zero, which corresponds to no detected bit error. This leaves us seven, which is one more than the number of bits. In Table 1 we have assigned that syndrome to a set of two bit errors that is consistent with the remaining syndrome. This remaining syndrome is also consistent with (5,1), which means that two corrections have the same Hamming distance from a valid codeword, and thus the same likelihood of being the correct codeword for the same syndrome.

5.5 Examining the Example in the Spreadsheet

The spreadsheet lets us vary the basis vectors and examine d_{MIN} and to test the code for various received codewords.

6 Assignment

6.1 Reading

Sklar, Chapter 6, Sections 6.1-6.6, pages 304-356.

6.2 Homework

Design the block code for Sklar problem 6.1 page 375. We will look at the probability of bit error for these codes next time.

6.3 SystemView

Class discussion. Next time be prepared to discuss your solutions to 2.4 and the quiz Problem 4 in class, with your simulation projected on the screen.

6.4 References

Peters, W. W. and Weldon, E. J., *Error Correcting Codes*, Second Edition, MIT Press (1072).

Abramowitz, M. and Stegun, I., *Handbook of Mathematical Functions*, U.S. Government Printing Office (1964; Second Printing 1972), also available from Dover.