ECE 6640
Digital Communications

Dr. Bradley J. Bazuin
Assistant Professor
Department of Electrical and Computer Engineering
College of Engineering and Applied Sciences
Chapter 6


1. Waveform Coding.
2. Types of Error Control.
4. Linear Block Codes.
5. Error-Detecting and Correcting Capability.
6. Usefulness of the Standard Array.
7. Cyclic Codes.
8. Well-Known Block Codes.
Sklar’s Communications System

Notes and figures are based on or taken from materials in the course textbook:
Bernard Sklar, Digital Communications, Fundamentals and Applications,
Signal Processing Functions

Notes and figures are based on or taken from materials in the course textbook:
Bernard Sklar, Digital Communications, Fundamentals and Applications.
Encoding and Decoding

- Data block of k-bits is encoded using a defined algorithm into an n-bit codeword
- Codeword transmitted
- Received n-bit codeword is decoded/detected using a defined algorithm into a k-bit data block
Waveform Coding Structured Sequences

- **Waveform Coding:**
  - Transforming waveforms into “better” waveform representations
  - Make signals antipodal or orthogonal

- **Structured Sequences:**
  - Transforming waveforms into “better” waveform representations that contain redundant bits
  - Use redundancy for error detection and correction

  - Bit sets become longer bit sets (redundant bits) with better “properties”
  - The required bit rate for transmission increases.
Antipodal and Orthogonal Signals

- **Antipodal**
  - Distance is twice “signal voltage”
  - Only works for one-dimensional signals

\[
d = 2 \cdot \sqrt{E_b} \quad z_{ij} = \frac{1}{E} \int_0^T s_i(t) \cdot s_j(t) \cdot dt = \begin{cases} 1 & \text{for } i = j \\ -1 & \text{for } i \neq j \end{cases}
\]

- **Orthogonal**
  - Orthogonal symbol set
  - Works for 2 to N dimensional signals

\[
d = \sqrt{2 \cdot E_b} \quad z_{ij} = \frac{1}{E} \int_0^T s_i(t) \cdot s_j(t) \cdot dt = \begin{cases} 1 & \text{for } i = j \\ 0 & \text{for } i \neq j \end{cases}
\]
M-ary Signals Waveform Coding

- Symbol represents $k$ bits at a time
  - Symbol selected based on $k$ bits
  - $M$ waveforms may be transmitted \( M = 2^k \)

- Allow for the tradeoff of error probability for bandwidth efficiency

- Orthogonality of $k$-bit symbols
  - Number of bits that agree = Number of bits that disagree
    \[
    z_{ij} = \sum_{k=1}^{K} \text{sum}(b_k^i = b_k^j) - \sum_{k=1}^{N} \text{sum}(b_k^i \neq b_k^j) = \begin{cases} 
    1 & \text{for } i = j \\
    0 & \text{for } i \neq j
  \end{cases}
    \]
Hadamard Matrix Orthogonal Codes

\[ H_D = \begin{bmatrix} H_{D-1} & H_{D-1} \\ H_{D-1} & \overline{H_{D-1}} \end{bmatrix} \]

- Start with the data set for one bit and generate a Hadamard code for the data set

\[
\begin{bmatrix} 0 \\ 1 \end{bmatrix} \Rightarrow H_1 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}
\]

\[
\begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \Rightarrow H_2 = \begin{bmatrix} H_1 & H_1 \\ H_1 & \overline{H_1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}
\]

\[ Z_{ij} = \begin{cases} 1 & \text{for } i = j \\ 0 & \text{for } i \neq j \end{cases} \]
Hadamard Matrix Orthogonal Codes (2)

\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 0 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
1 & 1 & 1 \\
\end{bmatrix}
\Rightarrow H_3 = \begin{bmatrix} H_2 & H_2' \\ H_2' & H_2 \end{bmatrix} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
\end{bmatrix}
\]

Data Set ➔ Codeword

See Hadamard in MATLAB
\( H_4 = \text{hadamard}(16) \)
\( H_4 \times \text{cor} = H_4' \times H_4 = H_4 \times H_4' \)
\( \text{ans} = 16 \times \text{eye}(16) \)

ECE 6640
Hadamard Matrix Orthogonal Codes (2)

• The Hadamard Code Set generates orthogonal code for the bit representations

\[ z_{ij} = \begin{cases} 
1 & \text{for } i = j \\
0 & \text{for } i \neq j 
\end{cases} \]

Letting

\[ 0 = +1V \]
\[ 1 = -1V \]

• For a D-bit pre-coded symbol a \(2^D\) bit symbol is generated from a \(2^D \times 2^D\) matrix

– Bit rate increases from \(D\) bits/sec to \(2^D\) bits/sec!

• For equally likely, equal-energy orthogonal signals, the probability of codeword (symbol) error (\(P_E\)) can be bounded as

\[ P_E(M) \leq (M-1) \cdot Q\left(\sqrt{\frac{E_s}{N_0}}\right) \]

\[ M = 2^D \]

\[ E_s = D \cdot E_b \]
Symbol error to Bit Error

- Discussed in chapter 4 (Equ 4.112) – k-bit set (codeword)
  - 4.9.3 Bit error probability versus symbol error probability for *orthogonal signals*

\[
\frac{P_B(k)}{P_E(k)} = \frac{2^{k-1}}{2^k - 1} \quad M = 2^k \quad \frac{P_B(M)}{P_E(M)} = \frac{M/2}{M - 1}
\]

- Substituting bit error probability for symbol error probability bounds previously stated

\[
P_E(M) \leq (M - 1) \cdot Q\left(\sqrt{\frac{E_s}{N_0}}\right)
\]

\[
P_B(k) \leq \left(2^{k-1}\right) \cdot Q\left(\sqrt{\frac{k \cdot E_b}{N_0}}\right) \quad P_B(M) \leq \left(\frac{M}{2}\right) \cdot Q\left(\sqrt{\frac{E_s}{N_0}}\right)
\]
Derivation of Equ 6.6

• In orthogonal signaling, a conditional probability for a message error can be stated as (1 of \( M-1 \) possible errors where \( m=2:M \))

\[
P(s_m \text{ received} \mid s_1 \text{ sent}) = \frac{P_e}{M-1} = \frac{P_e}{2^k-1}
\]

• If \( s_1 \) is a data sequence of \( k \) bits with a 0 is the first bit, there are \( 2^{k-1} \) possible sequences. The probability of the single bit error causing a message error at this location must be related to (for \( k \) large)

\[
P_{b_0} = 2^{k-1} \cdot P(s_M \text{ received} \mid s_1 \text{ sent}) = 2^{k-1} \cdot \frac{P_e}{M-1} = 2^{k-1} \cdot \frac{P_e}{2^k-1} \approx \frac{1}{2} \cdot P_e
\]

\[
P_B = \frac{M/2}{M-1} \cdot P_e = \frac{2^{k-1}}{2^k-1} \cdot P_e
\]
Biorthogonal Codes

- Code sets with cross correlation of 0 or -1

\[
z_{ij} = \begin{cases} 
1 & \text{for } i = j \\
-1 & \text{for } i \neq j, |i - j| = \frac{M}{2} \\
0 & \text{for } i \neq j, |i - j| \neq \frac{M}{2}
\end{cases}
\]

- Based on Hadamard codes (2-bit data set shown)

\[B_D = \begin{bmatrix} H_{D-1} \\ \overline{H}_{D-1} \end{bmatrix}
\]

\[
\begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \Rightarrow B_2 = \begin{bmatrix} H_1 \\ \overline{H}_1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}
\]
Biorthogonal Codes (2)

\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 0 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
1 & 1 & 1 \\
\end{bmatrix}
\Rightarrow B_3 = \begin{bmatrix}
H_2 \\
\overline{H_2}
\end{bmatrix} =
\begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 \\
\end{bmatrix}
\Rightarrow B_3 = \begin{bmatrix}
H_3 \\
\overline{H_3}
\end{bmatrix} =
\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 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
B3 Example

\[ H2 = \text{hadamard}(4) \]
\[ B3 = [H2; -H2] \]
\[ B3* B3' \]

\[
\begin{array}{cccccccc}
4 & 0 & 0 & 0 & -4 & 0 & 0 & 0 \\
0 & 4 & 0 & 0 & 0 & -4 & 0 & 0 \\
0 & 0 & 4 & 0 & 0 & 0 & -4 & 0 \\
0 & 0 & 0 & 4 & 0 & 0 & 0 & -4 \\
-4 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\
0 & -4 & 0 & 0 & 0 & 4 & 0 & 0 \\
0 & 0 & -4 & 0 & 0 & 0 & 4 & 0 \\
0 & 0 & 0 & -4 & 0 & 0 & 0 & 4
\end{array}
\]
Biorthogonal Codes (3)

- Biorthogonal codes require half as many code bits per code word as Hadamard.
  - The required bandwidth would be one-half
  - Due to some antipodal codes, they perform better

- For equally likely, equal-energy orthogonal signals, the probability of codeword (symbol) error can be bounded as

\[
P_E(M) \leq (M - 2) \cdot Q\left(\sqrt{\frac{E_s}{N_0}}\right) + Q\left(\sqrt{\frac{2 \cdot E_s}{N_0}}\right)
\]

\[
P_B(M) \approx \frac{P_E(M)}{2}, \quad \text{for } M > 8
\]

\[
M = 2^{k-1}
\]

\[
E_s = k \cdot E_b
\]
Biorthogonal Codes (4)

- Comparing to Hadamard to Biorthogonal
  - Note that for D bits, \( M \) is \( \frac{1}{2} \) the size for Biorthogonal

\[
P^\text{Biorthogonal}_B(M) \leq \left( \frac{M - 2}{2} \right) \cdot Q\left( \frac{\sqrt{E_s}}{N_0} \right) + Q\left( \frac{2 \cdot E_s}{N_0} \right)
\]

\( M = 2^{D-1} \)

\( M = 2^k \)

\( E_s = k \cdot E_b \)

\[
P^\text{Hadamard}_B(M) \leq \left( \frac{M}{2} \right) \cdot Q\left( \frac{\sqrt{E_s}}{N_0} \right)
\]

\( M = 2^k \)

\( E_s = k \cdot E_b \)
Waveform Coding System Example

• For a k-bit precoded symbol, one of M generators creates an encoded waveform.
• For a Hadamard Code, $2^k$ bits replace the k-bit input.
Waveform Coding System Example (2)

- This bit sequence can be PSK modulated (an antipodal binary system) for transmission over $M$ time periods of length $T_c$. $T = 2^k \cdot T_c = M \cdot T_c$

- The symbol rate is $1/M \cdot T_c$ $R_s = \frac{1}{T} = \frac{1}{2^k \cdot T_c}$

- For real-time transmission, the code word transmission rate and the $k$-bit input data rate must be identical.
  - Codeword bits shorter than original message bits
    
    $R_k = \frac{1}{k} \cdot T_b = R_s = \frac{1}{T} = \frac{1}{2^k \cdot T_c}$
    
    $T = k \cdot T_b = 2^k \cdot T_c$
    
    $T_b = \left(\frac{2^k}{k}\right) \cdot T_c$
Waveform Coding System Example (3)

- The coherent detector now works across multiple PSK bits representing a symbol.

Figure 5.5  Waveform-encoded system with coherent detection (receiver).
Waveform Coding System Example (4)

Stated Improvement

• For \( k=5 \), detection can be accomplished with about 2.9 dB less \( \text{Eb/No} \)!
  – Required bandwidth is \( 32/5=6.4 \) times the initial data word rate.

• Homework problem 6.28
  – Compare \( P_B \) of BPSK to the \( P_B \) for orthogonally encoded symbols at a predefined BER. Solution set uses \( 10^{-5} \).

\[ T_b = \left( \frac{2^k}{k} \right) \cdot T_C \]
Types of Error Control

• Error detection and retransmission
  – Parity bits
  – Cyclic redundancy checking
  – Often used when two-way communication used. Also used when low bit-error-rate channels are used.

• Forward Error Correction
  – Redundant bits for detection and correction of bit errors incorporated into the sequence
  – Structured sequences.
  – Often used for one-way, noisy channels. Also used for real-time sequences that can not afford retransmission time.
Structured Sequence

• Encoded sequences with redundancy

• Types of codes
  – Block Codes or Linear Block Codes (Chap. 6)
  – Convolutional Codes (Chap. 7)
  – Turbo Codes (Chap. 8)
Channel Models

• Discrete Memoryless Channel (DMC)
  – A discrete input alphabet, a discrete output alphabet, and a set of conditional probabilities of conditional outputs given the particular input.

• Binary Symmetric Channel
  – A special case of DMC where the input alphabet and output alphabets consist of binary elements. The conditional probabilities are symmetric \( p \) and \( 1-p \).
  – Hard decisions based on the binary output alphabet performed.

• Gaussian Channels
  – A discrete input alphabet, a continuous output alphabet
  – Soft decisions based on the continuous output alphabet performed.
Code Rate Redundancy

- For clock codes, source data is segmented into k bit data blocks.
- The k-bits are encoded into larger n-bit blocks.
- The additional n-k bits are redundant bits, parity bits, or check bits.
- The codes are referred to as (n,k) block codes.
  - The ratio of redundant to data bits is (n-k)/k.
  - The ratio of the data bits to total bits, k/n is referred to as the code rate. Therefore a rate ½ code is double the length of the underlying data.
Binary Bit Error Probability

- Defining the probability of \( j \) errors in a block of \( n \) symbols/bits where \( p \) is the probability of an error.

\[
P(j, n) = \binom{n}{j} \cdot p^j \cdot (1 - p)^{n-j}
\]

Binomial Probability Law

- The probability of \( j \) failures in \( n \) Bernoulli trials

\[
E[x] = n \cdot p \quad \quad E[x^2] = n \cdot p \cdot (1 - p)
\]
Code Example Triple Redundancy

• Encode a 0 or 1 as 000 or 111
  – Assume single bit errors detected and corrected

\[
P(j,3) = \left( \frac{3!}{j!(3-j)!} \right) \cdot p^j \cdot (1-p)^{3-j}
\]

• Assume BER of \( p = 10^{-3} \)

\[
P(0,3) = \left( \frac{3!}{0!(3-0)!} \right) \cdot (0.999)^3 = (0.999)^3 = 0.997
\]

\[
P(1,3) = \left( \frac{3!}{1!(3-1)!} \right) \cdot (0.001)^1 \cdot (0.999)^2 = 3 \cdot (0.001)^1 \cdot (0.999)^2 = 0.002994
\]

\[
P(2,3) = \left( \frac{3!}{2!(3-2)!} \right) \cdot (0.001)^2 \cdot (0.999)^1 = 3 \cdot (0.001)^2 \cdot (0.999)^1 = 0.000002997
\]

\[
P(3,3) = \left( \frac{3!}{3!(3-3)!} \right) \cdot (0.001)^3 = (0.001)^3 = 10^{-9}
\]
Code Example Triple Redundancy

- Probability of two or more errors
  \[ \Pr(x \geq 2) = P(2,3) + P(2,3) = 2.998 \cdot 10^{-6} \approx 3 \cdot 10^{-6} \]

- Probability of one or no errors
  \[ \Pr(x < 2) = P(0,3) + P(1,3) = 1 - \Pr(x \geq 2) = 1 - 2.998 \cdot 10^{-6} \approx 0.999997 \]

- If the raw bit error rate of the environment is \( p=10^{-3} \) …
- The probability of detecting the correct transmitted bit becomes \( 3 \cdot 10^{-6} \)
Parity-Check Codes

• Simple parity – single parity check code
  - Add a single bit to a defined set of bits to force the “sum of bits” to be even (0) or odd (1).
  - A rate k/(k+1) code
  - Useful for bit error rate detection

• Rectangular Code (also called a Product Code)
  - Add row and column parity bits to a rectangular set of bits/symbols and a row or column parity bit for the parity row or column
  - For an m x n block, m+1 x n+1 bits are sent
  - A rate mn/(m+1)(n+1) code
  - Correct signal bit errors!
(4,3) Single-Parity-Code

- (4,3) even-parity code example
  - 8 legal codewords, 8 “illegal” code words to detect single bit errors
  - Note: an even number of bit error can not be detected, only odd
  - The probability of an undetected message error

\[
\sum_{j=1}^{n/2(n\text{even})} P(2j, n) = \sum_{j=1}^{(n-1)/2(n\text{odd})} \left( \frac{n!}{2j!(n-2j)!} \right) \cdot p^{2j} \cdot (1 - p)^{n-2j}
\]

<table>
<thead>
<tr>
<th>Message</th>
<th>Parity</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>0 000</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>1 001</td>
</tr>
<tr>
<td>010</td>
<td>1</td>
<td>1 010</td>
</tr>
<tr>
<td>011</td>
<td>0</td>
<td>0 011</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>1 100</td>
</tr>
<tr>
<td>101</td>
<td>0</td>
<td>0 101</td>
</tr>
<tr>
<td>110</td>
<td>0</td>
<td>0 110</td>
</tr>
<tr>
<td>111</td>
<td>1</td>
<td>1 111</td>
</tr>
</tbody>
</table>
Matlab Results

\[ p = 1e^{-3}; \]

\[ n = 4 \]

Perror = 0
for \( j = 2:2:n \)
    \[ jfactor = \frac{\text{factorial}(n)}{\text{factorial}(j) \times \text{factorial}(n-j)} \]
    \[ jPe = jfactor \times p^j \times (1-p)^{n-j} \]
    \[ \text{Perror} = \text{Perror} + jPe \]
end

\[ n = 4.00000000000000e+000 \]
\[ \text{Perror} = 0.00000000000000e+000 \]
\[ jfactor = 6.00000000000000e+000 \]
\[ jPe = 2 \text{ bit errors} \]
\[ 5.98800600000000e-006 \]
\[ \text{Perror} = 5.98800600000000e-006 \]
\[ jfactor = 1.00000000000000e+000 \]
\[ jPe = 4 \text{ bit errors} \]
\[ 1.00000000000000e-012 \]
\[ \text{Perror} = 5.98800700000000e-006 \]

Probability of an undetected message error
Matlab Results

- Computing the probability of a message error
  - Channel bit size the parity is added to 2 to 16
  - Raw BER = 10^{-3}
  - Compare with parity and without (n x p)
Rectangular Code

- Add row and column parity bits to a rectangular set of bits/symbols and a row or column parity bit for the parity row or column.
- For an $m \times n$ block, $m+1 \times n+1$ bits are sent.
- A rate $mn/(m+1)(n+1)$ code.
- Correct signal bit errors!
  - Identify parity error row and column, then fix the “wrong bit”

\[
\begin{array}{ccc|c}
  m_1 & m_2 & m_3 & c_1 \\
  m_4 & m_5 & m_6 & c_2 \\
  m_7 & m_8 & m_9 & c_3 \\
  c_4 & c_5 & c_6 & \\
\end{array}
\]
Rectangular Code Example

- 5 x 5 bit array $\rightarrow$ 25 total bits
- 6 x 6 encoded array $\rightarrow$ 36 total bits
- A (36,25) code

- Compute probability that there is an undetected error message
  - The block error or word error becomes: $j$ errors in blocks of $n$ symbols and $t$ the number of bit errors. (note $t=1$ corrected)

$$P_M = \sum_{j=t+1}^{n} \binom{n}{j} \cdot p^j \cdot (1-p)^{n-j}$$

$$P_M \approx \binom{n}{t+1} \cdot p^{t+1} \cdot (1-p)^{n-t-1}$$
Error-Correction Code Tradeoffs

• Improve message error/bit error rate performance

• Tradeoffs
  – Error performance vs. bandwidth
    • More bits per sec implies more bandwidth
  – Power Output vs. bandwidth
    • Lowering Eb/No changer BER, to get it back, encode and increase bandwidth

• If communication need not be in real time … sending more bits increases BER at the cost of latency!
Coding Gain

- For a given bit-error probability, the relief or reduction in required $E_b/No$

\[
CodeGain(dB) = \left( \frac{E_b}{N_o} \right)_{uncoded} - \left( \frac{E_b}{N_o} \right)_{coded}
\]

Note:
Code gain exists for a given BER. (CB @ 1e-4 or ED @ 1e-6 in the figure)
CG may be different at different BERs!
Data Rate vs. Bandwidth

- From previous SNR discussions

\[
\frac{E_b}{N_0} = \frac{P_s}{P_N} \cdot (W \cdot T) = \frac{P_s}{P_N} \cdot \left(\frac{W}{R}\right), \quad R = \frac{1}{T}
\]

\[
\frac{E_b}{N_0} = \frac{P_s}{N_0 \cdot W} \cdot \left(\frac{W}{R}\right) = \frac{P_s}{N_0} \cdot \left(\frac{1}{R}\right)
\]

- The tradeoff between Eb/No and bit rate
Linear Block Codes

• Encoding a “message space” k-tuple into a “code space” n-tuple with an (n,k) linear block code
  – maximize the distance between code words
  – Use as efficient of an encoding as possible
  
  – $2^k$ message vectors are possible (for example $M$ k-tuples to send)
  – $2^n$ message codes are possible (M n-tuples after encoding)

• Encoding methodology
  – A giant table lookup
  – Generators that can span the subspace to create the desired linear independent sets can be defined
Linear Block Code Structure

- k-tuple encode into an n-tuple of an n-dimensional space.
- Error probability based on distance between selected encoded values.
- Distance to be maximized by design.

*Figure 6.10* Linear block-code structure.
Generator Matrix

- We wish to define an n-tuple codeword $U$

$$U = m_1 \cdot V_1 + m_2 \cdot V_2 + \cdots m_k \cdot V_k$$

where $m_i$ are the message bits (k-tuple elements), and $V_i$ are k linearly independent n-tuples that span the space.

- This allows us to think of coding as a vector-matrix multiple with $m$ a 1 x k vector and $V$ a k x n matrix

- If a generator matrix composed of the elements of $V$ can be defined

$$U = m \cdot G = m \cdot \begin{bmatrix} V_1 \\ V_2 \\ \vdots \\ V_k \end{bmatrix} = m \cdot \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{k1} & v_{k2} & \cdots & v_{kn} \end{bmatrix}$$
Table 6.1 Example

% Simple single bit parity
n = 6; k = 3; % Set codeword length and message length.
msg = [0 0 0; 0 0 1; 0 1 0; 0 1 1; ... 1 0 0; 1 0 1; 1 1 0; 1 1 1]; % Message is a binary matrix.

gen = [ 1 1 0 1 0 0; ... 0 1 1 0 1 0; ... 1 0 1 0 0 1];

code = rem(msg * gen, 2)

<table>
<thead>
<tr>
<th>Message vector</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000000</td>
</tr>
<tr>
<td>100</td>
<td>110100</td>
</tr>
<tr>
<td>010</td>
<td>011010</td>
</tr>
<tr>
<td>110</td>
<td>101110</td>
</tr>
<tr>
<td>001</td>
<td>101001</td>
</tr>
<tr>
<td>101</td>
<td>011101</td>
</tr>
<tr>
<td>011</td>
<td>110011</td>
</tr>
<tr>
<td>111</td>
<td>000111</td>
</tr>
</tbody>
</table>

To Match:
- swap c2 and c3
- swap c4 and c6

A Systematic Code

A Systematic Code with shifted message
“Systematic” Linear Block Codes

- The generator preserves part of the message in the resulting codeword

\[
G = [P|I] = \begin{bmatrix}
    p_{11} & p_{12} & \cdots & p_{1,(n-k)} & 1 & 0 & \cdots & 0 \\
    p_{21} & p_{22} & \cdots & p_{2,(n-k)} & 0 & 1 & \cdots & 0 \\
    \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
    p_{k1} & p_{k2} & \cdots & p_{k,(n-k)} & 0 & 0 & \cdots & 1 \\
\end{bmatrix} \begin{bmatrix}
    v_{11} & v_{12} & \cdots & v_{1n} \\
    v_{21} & v_{22} & \cdots & v_{2n} \\
    \vdots & \vdots & \ddots & \vdots \\
    v_{k1} & v_{k2} & \cdots & v_{kn} \\
\end{bmatrix}
\]

- \(P\) is the parity array and \(I\) regenerates the message bits
  - Note the previous example uses this format

- Matlab generators
  - hammgen
  - \(POL = \text{cyclpoly}(N, K, \text{OPT})\) finds cyclic code generator polynomial(s) for a given code word length \(N\) and message length \(K\).
“Systematic” Code Vectors

• Because of the structure of the generators

\[ G = [P | I_k] \]

• The codeword becomes

\[ U = m \cdot G = [p \mid m] \]

where p are the parity bits generated for the message m and m is the embedded message of a systematic code.
Parity Check Matrix

- The parity check matrix allows the checking of a codeword
  - It should be trivial if there are no errors …
    but if there are?! Syndrome testing

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

$$U \cdot H^T = m \cdot [P \mid I_k] \cdot H^T = m \cdot [P \mid I_k] \cdot \begin{bmatrix} I_{n-k} \\ P \end{bmatrix} = 0_{mr, n-k}$$
Parity Check Matrix

• Let the message be I

\[ U \cdot H^T = I_k \cdot [P \mid I_k] \cdot H^T = [P \mid I_k] \cdot \left[ \frac{I_{n-k}}{P} \right] = [P + P] = 0 \]

• The parity check matrix results in a zero vector, representing the “combination” of the parity bits.

\[
\begin{align*}
[\text{parmat}, \text{genmat}] &= \text{hammgen}(3) \\
\text{rem}(\text{genmat} \times \text{parmat}', 2)
\end{align*}
\]

\[
\begin{align*}
[\text{parmat}, \text{genmat}] &= \text{cyclgen}(7, \text{cyclpoly}(7, 4)) \\
\text{rem}(\text{genmat} \times \text{parmat}', 2)
\end{align*}
\]
Syndrome Testing

• Describing the received n-tuples of $U$

$$r = U + e$$

– where $e$ are potential errors in the received matrix

• As one might expect, use the check matrix to generate a
the syndrome of $r$

$$S = r \cdot H^T = (U + e) \cdot H^T = U \cdot H^T + e \cdot H^T$$

• but from the check matrix

$$S = e \cdot H^T$$

– if there are no bit errors, the result is a zero matrix!
Syndrome Testing

- If there are no bit error, the Syndrome results in a 0 matrix
- If there are errors, the requirement of a linear block code is to have a one to one mapping between correctable errors and non-zero syndrome results.

- Error correction requires the identification of the corresponding syndrome for each of the possible errors.
  - generate a $2^n \times n$ array representing all possible received n-tuples
  - this is called the standard array
Standard Array Format (n,k)

- The actual codewords are placed in the top row.
  - The 1st code word is an all zeros codeword.
  - It also defines the 1st coset that has zero errors
- Each row is described as a coset with a coset leader describing a particular error.
  - for an n-tuple, there will be n-k coset leaders, one of which is zero errors.
Figure 6.11

<table>
<thead>
<tr>
<th></th>
<th>000000</th>
<th>000001</th>
<th>000010</th>
<th>000100</th>
<th>001000</th>
<th>010000</th>
<th>100000</th>
<th>010001</th>
<th>100001</th>
<th>100101</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>110100</td>
<td>011011</td>
<td>101111</td>
<td>101000</td>
<td>011100</td>
<td>110010</td>
<td>000110</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000001</td>
<td>110101</td>
<td>011000</td>
<td>101100</td>
<td>101011</td>
<td>011111</td>
<td>110001</td>
<td>000101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000010</td>
<td>110110</td>
<td>011100</td>
<td>101010</td>
<td>101101</td>
<td>011001</td>
<td>110111</td>
<td>000011</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000100</td>
<td>110000</td>
<td>011110</td>
<td>101001</td>
<td>101111</td>
<td>010011</td>
<td>111011</td>
<td>001111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>001000</td>
<td>111100</td>
<td>010010</td>
<td>100110</td>
<td>100001</td>
<td>010101</td>
<td>111011</td>
<td>001111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>010000</td>
<td>100100</td>
<td>001010</td>
<td>111110</td>
<td>111001</td>
<td>001101</td>
<td>100011</td>
<td>010111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100000</td>
<td>010100</td>
<td>111010</td>
<td>001110</td>
<td>001001</td>
<td>111101</td>
<td>010011</td>
<td>100111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>010001</td>
<td>100101</td>
<td>001011</td>
<td>111111</td>
<td>111000</td>
<td>001100</td>
<td>100010</td>
<td>010110</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6.11 Example of a standard array for a (6, 3) code.
Decoding with the Standard Array

• The n-tuple received can be located somewhere in the standard array.
  – If there is an error, the corrupted codeword is replaced by the codeword at the top of the column.
  – The standard array contains all possible n-tuples, it is $2^{(n-k)} \times 2^k$ in size; therefore, all n-tuples are represented.
  – There are $2^{(n-k)}$ cosets
The Syndrome of a Coset

• Computing the syndrome of the jth coset for the ith codeword

\[ S = (U_i + e_j) \cdot H^T = U_i \cdot H^T + e_j \cdot H^T = e_j \cdot H^T \]

  – the syndrome is identical for the entire coset
  – Thus, coset is really a name defined for
    “a set of numbers having a common feature”
  – The syndrome uniquely defines the error pattern
Now that we know that the syndrome identifies a coset, we can identify the coset and perform error correction regardless of the n-tuple transmitted.

The procedure
1. Calculate the Syndrome of \( r \)
2. Locate the coset leader whose syndrome is identified
3. This is the assumed corruption of a valid n-tuple
4. Form the corrected codeword by “adding” the coset to the corrupted codeword n-tuple.
Error Correction Example

- see Table61_example
  - each step of the process is shown in Matlab for the (6,3) code of the text
Decoder Implementation

Figure 6.12 Implementation of the (6, 3) decoder.
Hamming Weight and Distance

- The Hamming weight of a code/codeword defines the performance.
- The Hamming weight is defined as the number of nonzero elements in \( U \).
  - For binary, count the number of ones.
- The Hamming difference between two codewords, \( d(U, V) \), is defined as the number of elements that differ.

<table>
<thead>
<tr>
<th>( U ) code =</th>
<th>w(( U )) = d(( U_0, U ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>1 0 1 0 0 1</td>
<td>3</td>
</tr>
<tr>
<td>0 1 1 0 1 0</td>
<td>3</td>
</tr>
<tr>
<td>1 1 0 1 1 1</td>
<td>4</td>
</tr>
<tr>
<td>1 1 0 1 0 0</td>
<td>3</td>
</tr>
<tr>
<td>0 1 1 1 0 1</td>
<td>4</td>
</tr>
<tr>
<td>1 0 1 1 1 0</td>
<td>4</td>
</tr>
<tr>
<td>0 0 0 1 1 1</td>
<td>3</td>
</tr>
</tbody>
</table>
Minimum Distance

• The minimum Hamming distance is of interest
  – \( d_{\text{min}} = 3 \) for the \((6,3)\) example

• Similar to distances between symbols for symbol constellations, the Hamming distance between codewords defines a code effective performance in detecting bit errors and correct them.
  – maximizing \( d_{\text{min}} \) for a set number of redundant bits is desired.
  – error-correction capability
  – error-detection capability
Hamming Distance Capability

- Error-correcting capability, t bits

\[ t = \left\lfloor \frac{d_{\text{min}} - 1}{2} \right\rfloor \]

- for (6,3) \( d_{\text{min}}=3 \): \( t = 1 \)

- Codes that correct all possible sequences of t or fewer errors may also be able to correct for some \( t+1 \) errors.
  - see the last coset of (6,3) – 2 bit errors!
  - a \( t \)-error-correcting (n,k) linear code is capable of correcting \( 2^{(n-k)} \) error patterns!

- an upper bound on the probabilility of message error is

\[
P_M \leq \sum_{j=t+1}^{n} \binom{n}{j} \cdot p^j \cdot (1-p)^{n-j}
\]
Hamming Distance Capability

- Error-detection capability
  \[ e = d_{\text{min}} - 1 \]
  - a block code with a minimum distance of d_{\text{min}} guarantees that all error patterns of d_{\text{min}}-1 or fewer error can be detected.
  - for (6,3) d_{\text{min}}=3: e = 2 bits

- The code may also be capable of detecting errors of d_{\text{min}} bits.
- An (n,k) code is capable of detecting \( 2^n-2^k \) error patterns of length n.
  - there are \( 2^k-1 \) error patterns that turn one codeword into another and are thereby undetectable.
  - all other patterns should produce non-zero syndrom
  - therefore we have the desired number of detectable error patterns
  - (6,3) \( \rightarrow 64-8=56 \) detectable error patterns
Codeword Weight Distribution

- The weight distribution involves the number of codewords with any particular Hamming weight
  - for (6,3) 4 have $d_{\text{min}}=3$, 3 have $d=4$
  - If the code is used only for error detection, on a BSC, the probability that the decoder does not detect an error can be computed from the weight distribution of the code.

$$P_{nd} = \sum_{j=1}^{n} A_j \cdot p^j \cdot (1 - p)^{n-j}$$

- for (6,3) $A(0)=1$, $A(3)=4$, $A(4)=3$, all else $A(i)=0$

$$P_{nd} = 4 \cdot p^3 \cdot (1 - p)^3 + 3 \cdot p^4 \cdot (1 - p)^2$$

$$= (4 \cdot (1 - p) + 3 \cdot p) \cdot p^3 \cdot (1 - p)^2$$

$$= (4 - p) \cdot p^3 \cdot (1 - p)^2$$

- for $p=0.001$: $P_{nd} \sim 3.9 \times 10^{-6}$
Simultaneous Error Correction and Detection

• It is possible to trade capability from the maximum guaranteed (t) for the ability to simultaneously detect a class of errors.

• A code can be used for simultaneous correction of alpha errors and detection of beta errors for \( \beta \geq \alpha \) provided

\[
d_{\text{min}} \geq \alpha + \beta + 1
\]
Visualization of (6,3) Codeword Spaces
MATLAB References

- see http://www.mathworks.com/help/comm/error-detection-and-correction.html
(8,2) Block Code

see: BlockCode82.m

A general script for clock codes
Block Codes

• Now that you have seen how to work with block codes …

• How do we select or generate a code?
  – from p. 348
  – “We would like for the space to be filled with as many codewords as possible (efficient utilization of the added redundancy), and we would like the codewords to be as far from one another as possible. Obviously, these goals conflict.”
Code Design: Hamming Bound

• The Hamming Bound is describes:
  – the number of parity bits needed
    \[ n - k \geq \log_2 \left\{ 1 + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{t} \right\} \]
  – the number of cosets is then
    \[ 2^{n-k} \geq \left\{ 1 + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{t} \right\} \]
  – Note that fewer cosets than \( 2^{n-k} \) may be used, for example cosets corresponding on 1 to t errors and not “additional” slots available for t+1 errors. (LinBlkCodeTestXX.m examples)
Defining Code Length

- $t$ described the bit corrections
  \[ d_{\text{min}} = 2 \cdot t + 1 \]

- Minimum non-trivial $k = 2$
  - if selected an $(n,2)$ code is needed

- Allow $t=2$
  - we will need $n - k \geq \log_2 \left\{ 1 + \binom{n}{1} + \binom{n}{2} \right\}$
    - Hamming bound might suggest $n=7$ (standard array size of $4 \times 29 < 2^7$
    - But for small codes, the Plotkin bound must also be met

  \[ d_{\text{min}} \leq \frac{n \cdot 2^{k-1}}{2^k - 1} \]
Defining Code Length

- Using the Plotkin bound for $n=7$

\[
d_{\text{min}} \leq \frac{n \cdot 2^{k-1}}{2^k - 1} = \frac{7 \cdot 2}{4-1} = \frac{14}{3} = \frac{2}{3}
\]

- But we need 5 or more …

- Therefore $n=8$ is the smallest possible code

\[
d_{\text{min}} \leq \frac{n \cdot 2^{k-1}}{2^k - 1} = \frac{8 \cdot 2}{4-1} = \frac{16}{3} = \frac{1}{3}
\]

- as we saw, the (8,2) code corrects 1, 2 and some 3 bit errors
MATLAB Examples

• LinBlkCodeTestXX.m
  – (3,1), (6,3), (7,4), and (8,2)
  – code in linblkcoderxXX.m explicitly defines coset matrix “E”,
  – but only 0:t bit error conditions are defined, there are coset errors
    conditions that may not be included
    (no extras for (3,1) or (7,4) … can you tell me why?)
  • (3,1) t=1 … 3 single bit errors $2^{n-k} = 4$ cosets available
  • (6,3) t=1 … 6 single bit errors $2^{n-k} = 8$ cosets available
  • (7,4) t=1 … 7 single bit errors $2^{n-k} = 8$ cosets available
  • (8,2) t=2 … 8 single bit and 28 two bit errors $2^{n-k} = 64$ cosets
  • (n,k) t=z … n single bit errors … $2^{n-k}$ cosets available
Coset Comments

- There is a syndrome created by the cosets
- The previous MATLAB examples defined the cosets and performed an exhaustive search of the syndrome table created from the coset to determine the error.

- However, the order of the coset is really arbitrary! (see linblkcoderx31.m you can switch the order!)
- The syndrome has $2^{n-k}$ possible values …
- Why not order the cosets so that the syndrome is a perfect counting sequence from 0 to $2^{n-k} - 1$ in binary?
  - This is what MATLAB’s syndtable.m does!
  - The exhaustive table search is no longer necessary.
Coset defined by syndtable

- When the coset errors are reorder so that the syndrome table is a binary counting sequence,
- The coset used to correct for errors is simply the row defined by the binary to decimal conversion of the syndrome.
  - 0 is still no errors and all zeros
  - all others are
    \[
    \text{correction\_code} = \text{coset(binary2decimal(syndrome),:)};
    \]
- See LBC_Test.m
- Also BlockCode63.m, BlockCode63b.m, and BlockCode82.m
  - look at the cosets for the additional bit error patterns corrected!
What About BER

• See ChCode_MPSK_BJB
  – uses linblkcoderxXX.m to decode!
  – code the simulation do better?
  – If the coset is only partially filled, yes … otherwise, no.
Cyclic Codes

- An important subclass of linear clock codes
- Easily implemented using a feedback shift register
  - all cyclic shifts of a codeword form another codeword
- The components of the codeword can be treated like a polynomial

\[ U(X) = u_0 + u_1 \cdot X + u_2 \cdot X^2 + \cdots + u_{n-1} \cdot X^{n-1} \]
Algebraic Structure of Cyclic Codes

- Based on
  \[ U(X) = u_0 + u_1 \cdot X + u_2 \cdot X^2 + \cdots + u_{n-1} \cdot X^{n-1} \]
  - \( U(X) \), an \((n-1)\) degree polynomial code word with the property
  \[ \frac{X^i \cdot U(X)}{X^n + 1} = q(X) + \frac{U^{(i)}(X)}{X^n + 1} \]
  \[ X^i \cdot U(X) = q(X) \cdot (X^n + 1) + U^{(i)}(X) \]
  - where the final term is the remainder of the division process and it is also a codeword
  - The remainder can also be described in terms of modulo arithmetic as
  \[ U^{(i)}(X) = X^i \cdot U(X) \mod (X^n + 1) \]
Example

\[ U(X) = u_0 + u_1 \cdot X + u_2 \cdot X^2 + \cdots + u_{n-1} \cdot X^{n-1} \]

- Multiplying a codeword by \(X\) and adding “two” \(u_{n-1}\) which is equal to zero in a binary sense

\[ X \cdot U(X) = u_{n-1} + u_0 \cdot X + u_1 \cdot X^2 + u_2 \cdot X^2 + \cdots + u_{n-2} \cdot X^{n-1} + u_{n-1} \cdot X^n + u_{n-1} \]

\[ X \cdot U(X) = u_{n-1} + u_0 \cdot X + u_1 \cdot X^2 + u_2 \cdot X^2 + \cdots + u_{n-2} \cdot X^{n-1} + u_{n-1} \cdot (X^n + 1) \]

- The final term module \(X^{n+1}\) results in a new codeword

\[ X \cdot U(X) = u_{n-1} + u_0 \cdot X + u_1 \cdot X^2 + u_2 \cdot X^2 + \cdots + u_{n-2} \cdot X^{n-1} \]

- Repeat for the rest of the codewords …. 
Binary Cyclic Codes

• We can generate a cyclic code using a generator polynomial in much the same way that we generated a block code using a generator matrix.
  – The generator polynomial $g(x)$ for an $(n,k)$ cyclic code is unique and is of the form (with the requirement that $g_0=1$ and $g_p=1$)
    $$g(X) = g_0 + g_1 \cdot X + g_2 \cdot X^2 + \cdots + g_p \cdot X^p$$
  – Every codeword in the space is of the form
    $$U(X) = m(X) \cdot g(X)$$
  – with the message polynomial
    $$m(X) = m_0 + m_1 \cdot X + m_2 \cdot X^2 + \cdots + m_{n-p-1} \cdot X^{n-p-1}$$
  – such that $k-1=n-p-1$ or the number of parity bits is
    $$p = n - k$$
Binary Cyclic Codes

• U is said to be a valid codeword of the subspace S if, and only if, g(X) divides into U(X) without a remainder.

\[ U(X) = m(X) \cdot g(X) \]

\[ \frac{U^{(i)}(X)}{g(X)} = m(X) \]

• A generator polynomial g(X) of an (n,k) cyclic code is a factor of \( X^n + 1 \). (A result of the codewords being cyclic).

• Therefore some example generators are

\[ X^7 + 1 = (X^3 + X + 1) \cdot (X^4 + X^2 + X + 1) \]

– g(X) for (7,3) with \( p=3 \)

– g(X) for (7,4) with \( p=4 \)
Systematic Encoding

- Output message and feed message to LFSR
- When message done, the parity bits are in the encoder and can be output.
- Initialize the encoder and restart for the next message.
- Syndrome computation is performed in a similar manner.
Cyclic Block Code

- With proper manipulation, the cyclic block codes can use linear combinations of codes to recreate the generator and parity matrices of block codes.

- Then processing could also be performed as previously shown.
Well-Known Block Codes

• Hamming Codes
  – minimum distance of 3, single error correction codes.
  – known as perfect codes, codes where the standard array has all patterns of $t$ or fewer errors and no others as coset leaders (that is no residual error-correcting capacity)

• Extended Golay Codes

• Bose-Chadhuri-Hocquenghem (BCH) Codes
  – powerful class of cyclic codes
  – generalization of Hamming codes that allow multiple error correction
Hamming Codes

- Hamming codes are characterized by a single integer \( m \) that creates codes of
  \[
  (n, k) = \left(2^m - 1, 2^m - 1 - m\right)
  \]
  - \((3,1), (7,4), (15,11), (31,26),\) etc.
  - \(d_{\text{min}} = 3, t=1, e=2\) … a perfect code

- \(2^{(n-k)} = 2^m\) therefore 1 all zeros coset leader
  and \(2^m - 1 = n\) single bit errors. Always a perfect code!
Code Generation Algorithm

- In MATLAB see the function hamgen
  - Produce parity-check and generator matrices for Hamming code.
  - \( H = \text{hammgen}(M) \) produces the parity-check matrix \( H \) for a given integer \( M \), \( M \geq 3 \). The code length of a Hamming code is \( N=2^M-1 \). The message length is \( K = 2^M - M - 1 \). The parity-check matrix is an \( M \)-by-\( N \) matrix.
  - \([H, G, N, K] = \text{hammgen}(...)\) produces the parity-check matrix \( H \) as well as the generator matrix \( G \), and provides the codeword length \( N \) and the message length \( K \).
**MATLAB hamgen**

\[ [H, G, N, K] = 	ext{hammgen}(3) \]

\[
H = \\
1 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 1
\]

\[
G = \\
1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 1
\]

\[
N = \\
7
\]

\[
K = \\
4
\]
Performance Improvement

- If we assume BPSK transmission with coherent detection.
- The channel symbol error and bit error probability of

\[
P_S = \sum_{j=t+1}^{n} \binom{n}{j} \cdot p^j \cdot (1 - p)^{n-j}
\]

\[
P_B \approx \frac{1}{n} \cdot \sum_{j=t+1}^{n} \binom{n}{j} \cdot p^j \cdot (1 - p)^{n-j}
\]

- For performance over a Gaussian channel we have

\[
p = Q\left(\sqrt{\frac{2E_c}{N_0}}\right)
\]

\[
\frac{E_c}{N_0} = k \cdot \frac{E_b}{N_0}
\]

- with the results plotted in on Figure 6.22 and the following slide.
Comparative Code Performance
Figure 6.22

Figure 6.22 $P_e$ versus $E_b/N_0$ for coherently demodulated BPSK over a Gaussian channel for several block codes.
Extended Golay Code

- The binary (24,12) code generated from the perfect (23,12) Golay code.
  - A rate \( \frac{1}{2} \) code. \( d_{\text{min}} = 8 \)
  - Correct 3 bit errors, detect 7 bit errors

\[
P_B \approx \frac{1}{24} \cdot \sum_{j=4}^{24} \binom{24}{j} \cdot p^j \cdot (1-p)^{24-j}
\]
BCH Codes

- Bose-Chadhuri-Hocquenghem (BCH)
  - One of the key features of BCH codes is that during code design, there is a precise control over the number of symbol errors correctable by the code. In particular, it is possible to design binary BCH codes that can correct multiple bit errors.
  - BCH codes are used in applications such as satellite communications,[2] compact disc players, DVDs, disk drives, solid-state drives[3] and two-dimensional bar codes.

- Table 6.4 provides generators of primitive BCH Codes based on the desired (n,k) and t
BCH Code Performance
Figure 6.23

Figure 6.23 $P_e$ versus $E_b/N_0$ for coherently demodulated DQPSK over a Gaussian channel using BCH codes. (Reprinted with permission from L. J. Weng, "Soft and Hard Decoding Performance Comparisons for BCH Codes," Proc. Int. Conf. Commun., 1979, Fig. 3, p. 25.5.5. © 1979 IEEE.)
Matlab Block Code Examples

- ChCode_MPSK_BJB
  - BPSK, QPSK or 8-PSK Time Symbols at a fixed 1 Mbps channel symbol bit rate (1 Msp for BPSK, ½ Msp for QPSK, and 1/3 Msp for 8-PSK)
  - Linear Block Channel Encoding
    - (3,1) with t=1, (6,3) with t=1, (7,4) with t=1, and (8,2) with t=2
  - Comparison curve based on symbol bit error rate versus corrected bit-error rates for the data stream
BPSK Simulation Results

\[(3,1) \ t=1, \ rate \ 1/3 \quad (6,3) \ t=1, \ rate \ 1/2\]
BPSK Simulation Results (2)

(7,4) t=1, rate 4/7

(8,2) t=2, rate 1/4