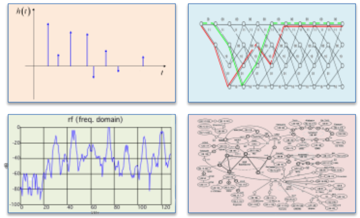


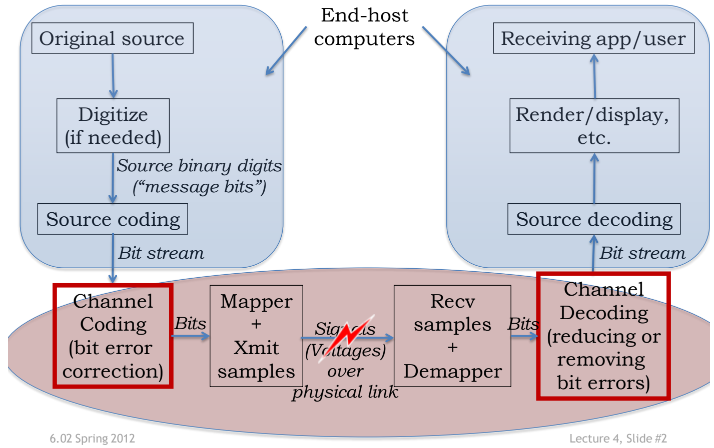
INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

6.02 Spring 2012
Lecture #4

- Linear block codes - Properties
- Rectangular parity, Hamming

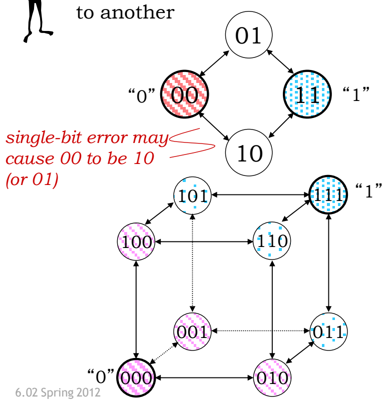
Single Link Communication Model



Idea: Embedding for Structural Separation

Encode so that the codewords are "far enough" from each other

Likely error patterns shouldn't transform one codeword to another



Code: nodes chosen in hypercube + mapping of message bits to nodes

If we choose 2^k out of 2^n nodes, it means we can map all k-bit message strings in a space of n-bit *codewords*. The ***code rate*** is **k/n**.

Linear Block Codes

Block code: ***k*** message bits encoded to **n** code bits i.e., each of $\mathbf{2^k}$ messages encoded into a unique **n-bit** combination via a *linear transformation.*
Set of parity equations (in GF(2)) represents code.

Key property: Sum of any two codewords is *also* a codeword → necessary and sufficient for code to be linear.

(n,k) code has rate **k/n**.
Sometime written as **(n,k,d)**, where **d** is the Hamming Distance of the code.

Examples: What are n, k, d here?

{000, 111} (3,1,3). Rate= 1/3.

{0000, 1100, 0011, 1111} (4,2,2). Rate = ½.

{00000} {5,0,_). Rate = 0!

{1111, 0000, 0001}

{1111, 0000, 0010, 1100}

Not linear codes!

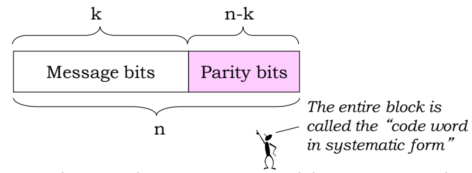
0000000	1100001	1100110	0000111
0101010	1001011	1001100	0101101
1010010	0110011	0110100	1010101
1111000	0011001	0011110	1111111

(7,4,3) code. Rate = 4/7.

The HD of a linear code is the number of "1"s in the non-zero codeword with the smallest # of "1"s

(n,k) Systematic Linear Block Codes

- Split data into ***k***-bit blocks
- Add **(*n*-*k*)** parity bits to each block using **(*n*-*k*)** linear equations, making each block ***n*** bits long



- Every linear code can be represented in systematic form

Example: Rectangular Parity Codes

Idea: start with rectangular array of data bits, add parity checks for each row and column. Single-bit error in data will show up as parity errors in a particular row and column, pinpointing the bit that has the error.

D_1	D_2	P_1
D_3	D_4	P_2
P_3	P_4	

P_1 is parity bit for row #1

(n,k,d)=?

P_4 is parity bit for column #2

0 1 1
1 1 0
1 0

Parity for each row and column is correct ⇒ no errors

0 1 1
1 0 0
1 0

Parity check fails for row #2 and column #2 ⇒ bit D_4 is incorrect

0 1 1
1 1 1
1 0

Parity check only fails for row #2 ⇒ bit P_2 is incorrect

Rectangular Code Corrects Single Errors

Claim: The HD of the rectangular code with ***r*** rows and ***c*** columns is **3**. Hence, it is a single error correction (SEC) code.

Code rate = *rc* / (*rc* + *r* + *c*).

If we add an overall parity bit P, we get a (rc+r+c+1, rc, 4) code

Improves error detection but not correction capability

D_1	D_2	D_3	D_4	P_1
D_5	D_6	D_7	D_8	P_2
D_9	D_{10}	D_{11}	D_{12}	P_3
P_4	P_5	P_6	P_7	P

Proof: Three cases.
(1) Msgs with HD 1 → differ in 1 row and 1 col parity
(2) Msgs with HD 2 → differ in either row OR col or both → HD >= 4 here.
(3) Msgs with HD 3 or more → systematic code so differ in that many bits

Decoding Rectangular Parity Codes

Receiver gets possibly corrupted word, ***w***.

Calculates all the parity bits from the data bits.

If no parity errors, return ***rc*** bits of data.

Single row or column parity bit error → ***rc*** data bits are fine, return them

If parity of row ***x*** and parity of column ***y*** are in error, then the data bit in the **(*x*,*y*)** position is wrong; flip it and return the ***rc*** data bits

All other parity errors are *uncorrectable*. Return the data as-is, flag an "uncorrectable error"

Let's do some rectangular parity decoding

D1	D2	P1
D3	D4	P2
P3	P4	

Received codewords

1	0	1
0	1	0
0	1	

1. Decoded message bits: **1011**

0	0	0
1	1	1
1	1	

2. Decoded message bits: **0011**

P2 parity error

0	0	1
0	1	0
0	0	

3. Decoded message bits: **0001**

"uncorrectable"

How Many Parity Bits Do We Really Need?

- We have **n-k** parity bits, which collectively can represent $\mathbf{2^{n-k}}$ possibilities
- For **single-bit error** correction, parity bits need to represent two sets of cases:
  - Case 1: No error has occurred (1 possibility)
  - Case 2: Exactly one of the code word bits has an error (**n** possibilities, not **k**)
- So we need $\mathbf{n+1 \le 2^{n-k}}$

  $n \le 2^{n-k} - 1$ (Hamming bound)
- Rectangular codes do not satisfy the equality
- Hamming codes correct single errors with this minimum number of parity bits

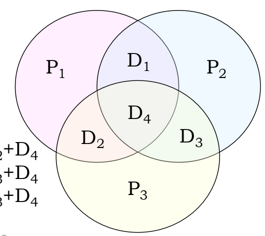
  (7,4,3), … , $(2^m-1, 2^m-1-m, 3)$

Towards More Efficient Codes: (7,4,3) Hamming Code Example

- Use multiple parity bits, each covering a subset of the data bits.
- No two message bits belong to exactly the same subsets, so a <u>single-bit error</u> will generate a unique set of parity check errors.

Modulo-2 addition, aka XOR

$P_1 = D_1+D_2+D_4$
$P_2 = D_1+D_3+D_4$
$P_3 = D_2+D_3+D_4$



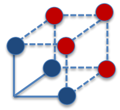
Suppose we check the parity and discover that P1 and P3 indicate an error?
bit D2 must have flipped

What if only P2 indicates an error?
P2 itself had the error!

Logic Behind Hamming Code Construction

- Idea: Use parity bits to cover each axis of the binary vector space
  - That way, all message bits will be covered with a **unique** combination of parity bits

Index	1	2	3	4	5	6	7
Binary index	001	010	011	100	101	110	111
(7,4) code	**P1**	**P2**	**D1**	**P3**	**D2**	**D3**	**D4**



P_1 with binary index 00**1** covers

D_1 with binary index 01**1**
D_2 with binary index 10**1**
D_4 with binary index 11**1**

$P_1 = D_1+D_2+D_4$
$P_2 = D_1+D_3+D_4$
$P_3 = D_2+D_3+D_4$