ELEC1200: A System View of Communications: from Signals to Packets
Lecture 11

- Review of (8,4) code
- (9,4,4) code
- Burst Error Correction
  - Interleaving
  - Framing
  - Bit Stuffing

Some content taken with permission from material for the course EECS6.02 by C. Sodini, M. Perrot and H. Balakrishnan at MIT

Review: Error Correcting Coding

1. Take an input message stream (here 12 bits).

2. Break the message stream into k-bit blocks (here 3 blocks of k = 4 bits).

3. Add (n-k) parity bits to form n-bit codeword (here n = 8).

4. Transmit data through noisy channel and receive codewords with some errors

5. Perform error correction

6. Extract the k message bits from each corrected codeword.

011011101101

0110
1110
1101

01101111
11100101
11010110

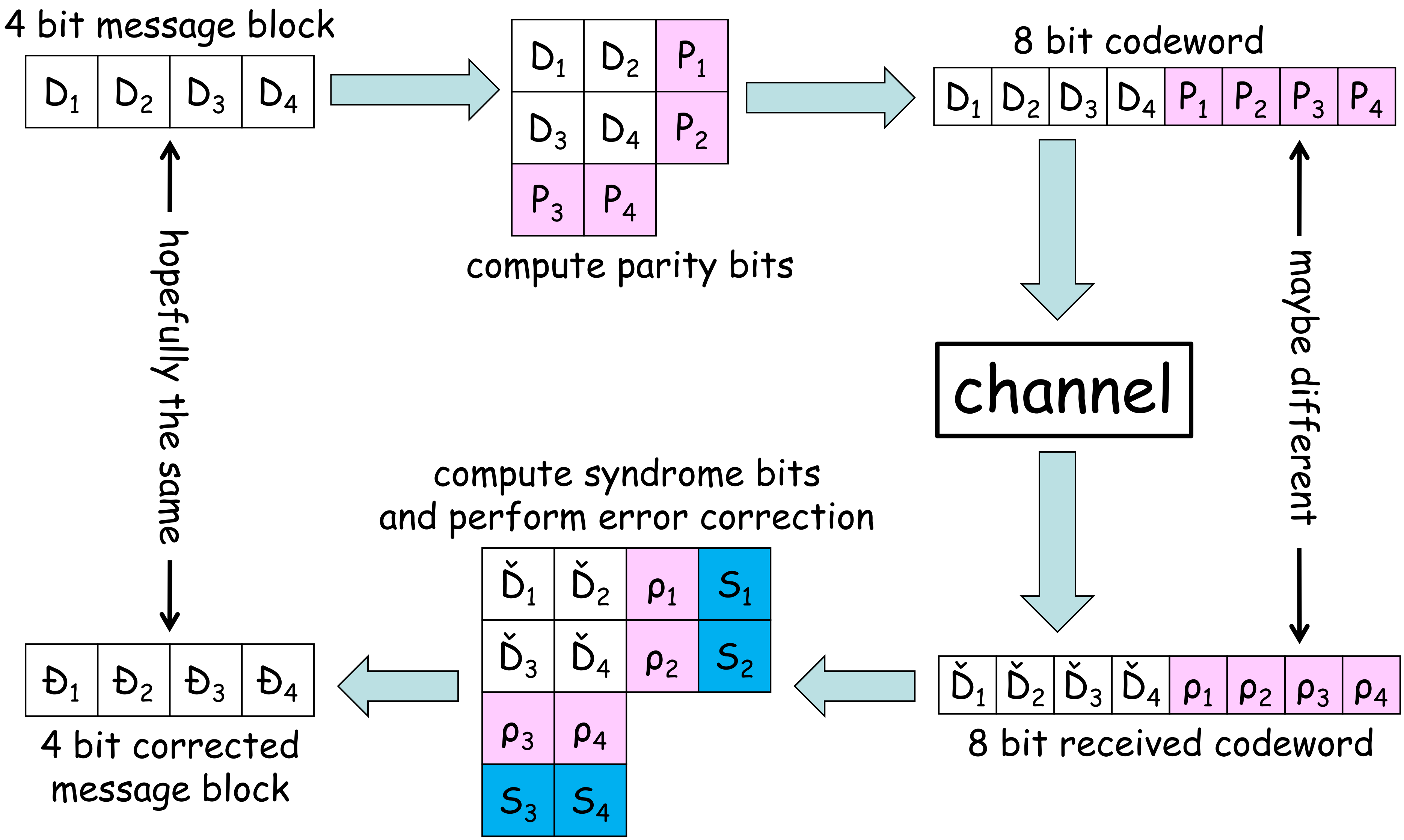
channel

01100111
11110101
11000110

0110 1110 1101

Some content taken with permission from material for the course EECS6.02 by C. Sodini, M. Perrot and H. Balakrishnan at MIT

Review: (8,4) code



Performing Error Correction

- Take the code word, and rearrange it

D_1	D_2	D_3	D_4	P_1	P_2	P_3	P_4

D_1	D_2	P_1
D_3	D_4	P_2
P_3	P_4	

- Compute the syndrome bit required for each row/column to be even parity

D_1	D_2	P_1	S_1
D_3	D_4	P_2	S_2
P_3	P_4		
S_3	S_4		

- Check the syndrome bits.
  - If all are 0, there is no error
  - If only syndrome bits for column i and row j are 1 (e.g. S_1 and S_3), there is a bit error in the data bit at i,j (e.g. D_1)
  - If only one syndrome bit is 1 (e.g. S_2), there is an error in the corresponding parity bit (e.g. P_2).

(n,k,d) Block Codes

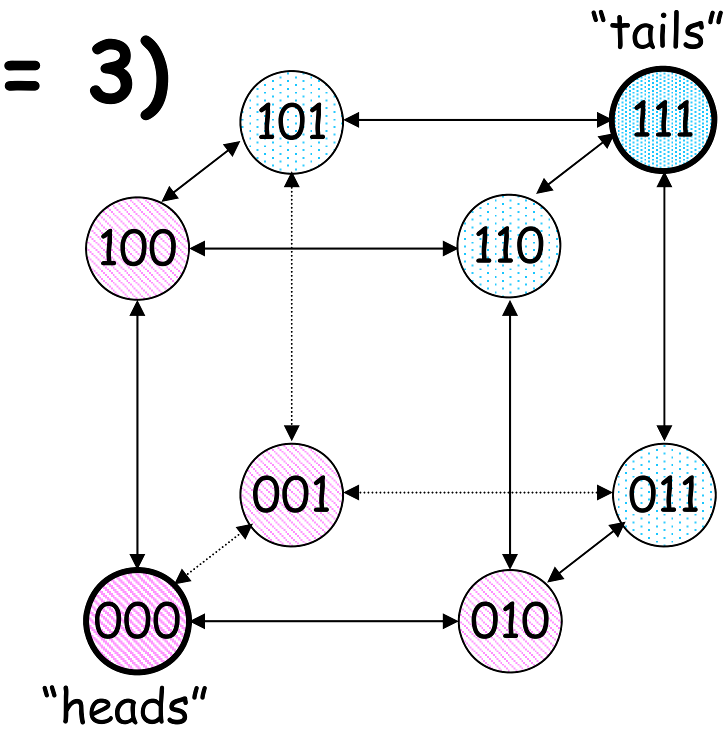
- **These are the same as (n,k) codes, except that we indicate the minimum Hamming distance, d, between valid code words**
  - **For example, the (8,4) code studied before is an (8,4,3) code.**
- **The minimum Hamming distance determines how many bit errors we can detect or correct.**
  - **We can <u>detect but not correct</u>, errors in at most d-1 bits. (If d=3, we can detect 1 or 2 bit errors.)**

OR

  - **We can <u>detect and correct</u> errors in at most (d-1)/2 bits. (If d=3, we can detect and correct only 1 bit errors)**
- **Note we must choose which one of the above options we want to do.**

Some content taken with permission from material for the course EECS6.02 by C. Sodini, M. Perrot and H. Balakrishnan at MIT

Example (d = 3)



- With a (n,k,3) code, we can EITHER
  - Detect 1-bit and 2-bit errors

    OR
  - Detect and correct 1-bit errors

  but not both!
- Suppose we observe the codeword 100 in the (3,1,3) code. EITHER
  - 000 was sent and a 1 bit error occurred

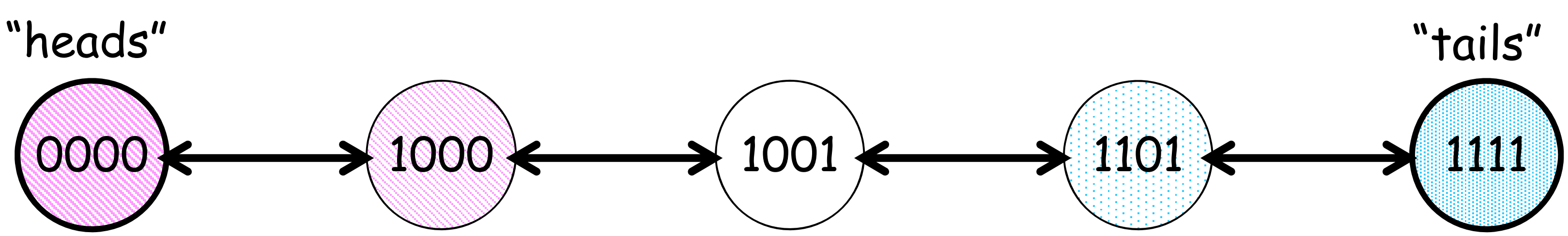
    OR
  - 111 was sent and a 2 bit error occurred.
- If we choose to correct 100 to 000, we are assuming that 2 bit errors never occur.

Some content taken with permission from material for the course EECS6.02 by C. Sodini, M. Perrot and H. Balakrishnan at MIT

Example (d = 4)

- With a (n,k,4) code, we can EITHER
  - Detect 1-, 2-, and 3-bit errors

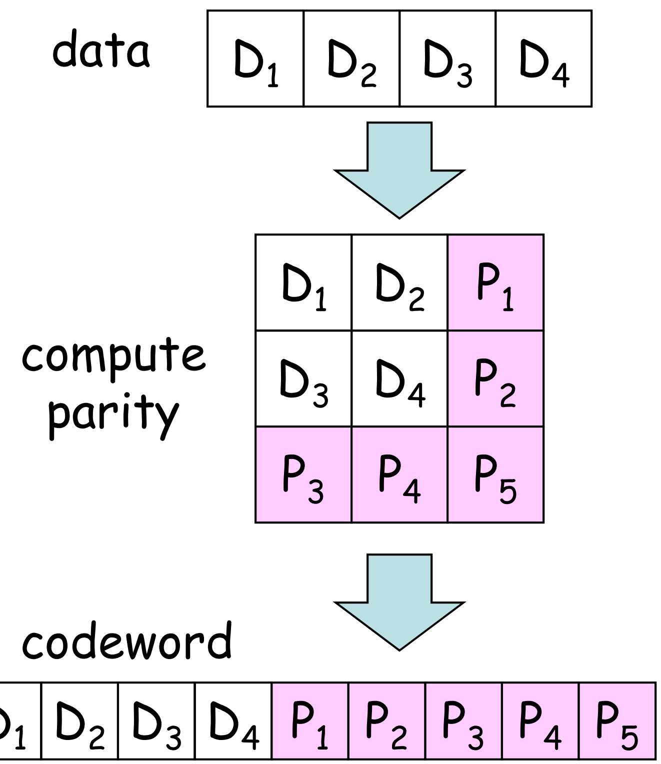
    OR
  - Detect and correct 1-bit errors and detect 2- bit errors
- For example, consider the (4,1,4) code:
  - If we observe 1001, we cannot determine whether 0000 or 1111 was transmitted.
  - If we observe 1000, then either a 1 bit or a 3 bit error occurred. If we correct, we assume the 3 bit error did not occur.



Some content taken with permission from material for the course EECS6.02 by C. Sodini, M. Perrot and H. Balakrishnan at MIT

A (9,4,4) code

- We can increase the minimum Hamming distance in the (8,4,3) code to 4 by adding an overall parity bit P5, which is chosen so that the codeword always has even parity.
- Examples
  - Data Block: 1010
    - Compute Parity: 101
      101
      000
    - Codeword: 101011000
  - Data Block: 1110
    - Compute Parity: 110
      101
      011
    - Codeword: 111001011



Performing Error Correction

D_1	D_2	P_1	S_1
D_3	D_4	P_2	S_2
P_3	P_4	P_5	
S_3	S_4		S_5

- The (9,4,4) code allows us to
  - Detect and correct 1 Bit Errors

    AND
  - Detect 2 Bit Errors
- To do this
  - Compute the syndrome bits by checking each row/column and the entire codeword for even parity
  - Check the syndrome bits.
    - If all are 0, there is no error
    - If S_5=1, a 1 bit error occurred. Check the other syndrome bits to see which bit to correct (note that the parity bits may contain errors)
    - If S_5=0 but one or more of the other syndrome bit are nonzero, a 2 bit error occurred.
- Alternatively, we could detect 1,2 or 3 bit errors.

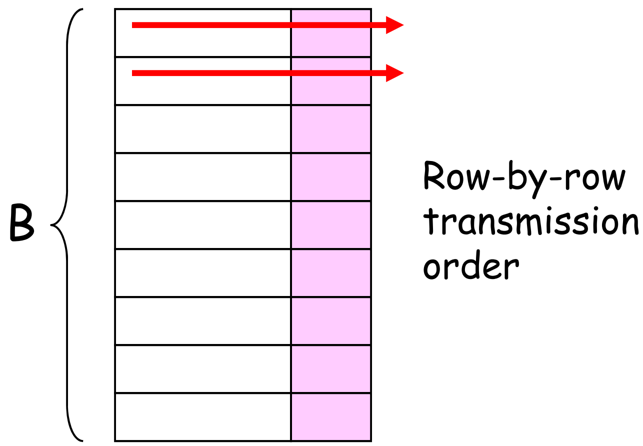
Examples of Error Correction

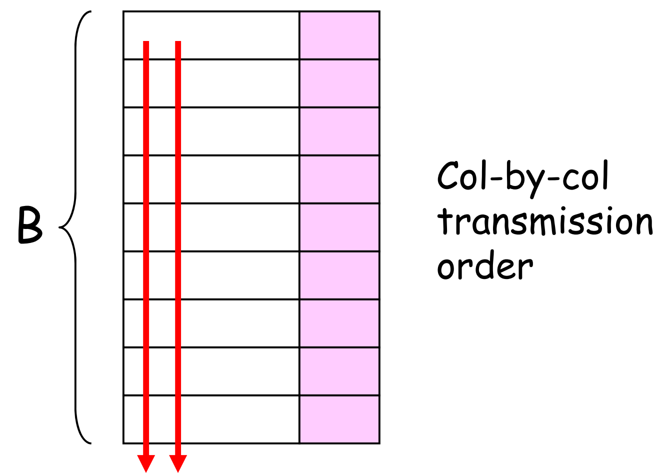
Received Codeword	Rearranged Codeword	Computed Syndrome	Corrected Data
011110101	0 1 1 1 1 0 1 0 1	0 1 1 0 1 1 0 0 1 0 1 0 0 0	0111 no errors
011010101	0 1 1 1 0 0 1 0 1	0 1 1 0 1 0 0 1 1 0 1 0 1 1	0111 D_4 incorrect
011111111	0 1 1 1 1 1 1 1 1	0 1 1 0 1 1 1 1 1 1 1 0 1 0	???? 2 bit error

Burst Errors

Correcting single-bit of few-bit errors is nice, but in many situations errors come in bursts many bits long (e.g., damage to storage media, burst of interference on wireless channel, …). How does single-bit or few-bit error correction help with that?

Well, can we think of a way to turn a B-bit error burst into B single-bit errors?





Problem: Bits from a particular codeword are transmitted sequentially, so a B-bit burst produces multi-bit errors.

Solution: interleave bits from B different codewords. Now a B-bit burst produces 1-bit errors in B different codewords.

Some content taken with permission from material for the course EECS6.02 by C. Sodini, M. Perrot and H. Balakrishnan at MIT

Framing

- **Looking at a received bit stream, how do we know where a block of interleaved codewords begins?**
- **Physical indication (transmitter turns on, beginning of disk sector, separate control channel)**
- **Place a unique bit pattern (frame sync sequence) in the bit stream to mark start of a block**
  - **Frame = sync pattern + interleaved code word block**
  - **Search for sync pattern in bit stream to find start of frame**
  - **Bit pattern can't appear elsewhere in frame (otherwise our search will get confused), so have to make sure no legal combination of codeword bits can accidentally generate the sync pattern**
  - **Sync pattern can't be protected by ECC, so errors may cause us to lose a frame every now and then, a problem that will need to be addressed at some higher level of the communication protocol.**

Bit Stuffing

- Each frame begins and ends with a special bit pattern called a sync pattern. For example, the sync pattern might consist of 6 consecutive ones surrounded by zeros

  [01111110]

- Inside each frame, whenever the sender encounters *five consecutive ones* in the data stream, it stuffs a 0 bit into the outgoing stream.
- When the receiver sees *five consecutive incoming ones* followed by a 0, it unstuffs (removes) the 0 bit.

Bit Stuffing Example

Input Stream

0110111111100111110111111111100000

Stuffed Stream

0110111110110011111001111101111100000

Stuffed bits

Unstuffed Stream

0110111111100111110111111111100000

Some content taken with permission from material for the course EECS6.02 by C. Sodini, M. Perrot and H. Balakrishnan at MIT

Summary: Channel Coding

1. Take an input message stream: 011011101101
2. Break the message stream into k-bit blocks (e.g. k = 4).
   0110
   1110
   1101
3. Add (n-k) parity bits to form n-bit codeword (e.g. n = 8)
   01101111
   11100101
   11010110
4. Interleave bits from B codewords (e.g., B = 3).
   011111110001100111101110
5. Bit-stuff the interleaved block
   01111011100011001111001110
6. Add the sync pattern (e.g. [0111110]).
   01111100111101110001100111100 1110

Some content taken with permission from material for the course EECS6.02 by C. Sodini, M. Perrot and H. Balakrishnan at MIT

Step 4: sync = 0111110

Summary: Error Correction

0111110011110111000110011110 01110

1. Receive bit stream (with errors)

0111110011110111001000011110 01110

2. Search for and remove sync pattern (0111110)

01111011100100001111001110

3. Destuff the frame

011111110010000111101110

4. De-interleave the bits to form B n-bit codewords (B=3, n=8)

01100111
11110101
11000110

5. Perform error correction

010	110	110
101	111	001
11	01	10

6. Extract the k=4 message bits from each corrected codeword.

0110 1110 1101

Some content taken with permission from material for the course EECS6.02 by C. Sodini, M. Perrot and H. Balakrishnan at MIT

Summary

- The minimum Hamming distance between code words determines how many bit errors in the codeword we can detect or correct.
- We can increase the minimum Hamming distance in the (8,4,3) code by adding an overall parity bit, resulting in a (9,4,4) code.
  - This adds the capability of detecting 2 bit errors.
- We can handle B-bit burst errors by
  - Interleaving B codewords
  - Adding a sync pattern so receiver can find the start of each block
  - Bit stuff the interleaved block so that sync is unique.

Some content taken with permission from material for the course EECS6.02 by C. Sodini, M. Perrot and H. Balakrishnan at MIT