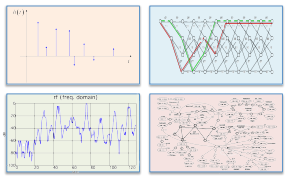


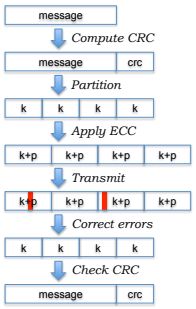
INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

6.02 Fall 2010
Lecture #8

- Error detection (cont.)
- Convolutional codes: state & trellis diagrams
- What "most likely message" means…

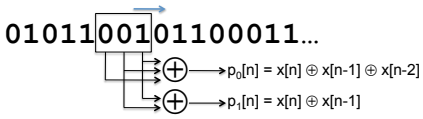
Coded Transmissions So Far in 6.02



- Start with original message.
- Add CRC to enable verification of error-free transmission.
- Apply ECC using block code, adding parity bits to each k-bit block of the message. Our ECCs were designed for *single-bit error correction*.
- After xmit, correct errors.
- Verify CRC, fails if undetected/ uncorrectable error.
- Deliver or discard message.

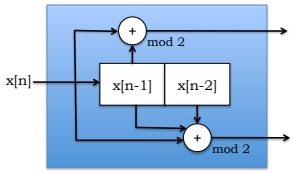
Convolutional Codes

- Like the block codes discussed earlier, send parity bits computed from blocks of message bits
  - Unlike block codes, send **only** the parity bits!
  - The code rate of a convolutional code tells you how many parity bits (r) are sent for each message bit. We'll be talking about rate 1/r codes for the most part.
  - Use a sliding window to select which message bits are participating in the parity calculations. The width of the window (in bits) is called the code's constraint length, K.



Block diagram view

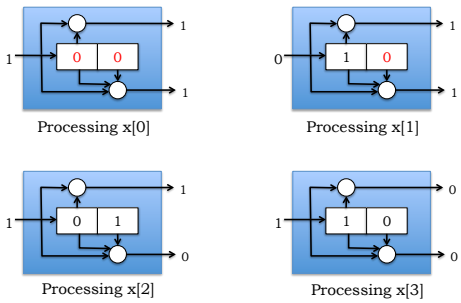
- One often sees convolutional encoders described with a block diagram like the following:



The x[n-i] values are referred to as the "state" of the encoder.

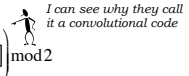
- Think of this a "black box": message in, parity out
  - Input bits arrive one-at-a-time on the wire on the left
  - The box computes the parity bits using the incoming bit and the K-1 previous message bits
  - At the end of the bit time, all the saved message bits are shifted right one location and the incoming bit moves into the left locn.

Example: xmit 1011



Parity Bit Equations

- A convolutional code generates sequences of parity bits from sequences of message bits:



$$p_i[n] = \left(\sum_{j=0}^{k-1} g_i[j]\, x[n-j] \right) \bmod 2$$

- K is the constraint length of the code
  - The larger K is, the more times a particular message bit is used when calculating parity bits
    → greater redundancy
    → better error correction possibilities
    → but longer decoding time (exponential in K for the decoder we'll study)
- g_i is the K-element generator polynomial for parity bit p_i.
  - Each element $g_i[n]$ is either 0 or 1
  - More than one parity sequence can be generated from the same message

Convolutional Codes (cont'd.)

- We'll transmit the parity sequences, not the message itself
  - Can recover the message sequences from the parity sequences
  - Each message bit is "spread across" K elements of each parity sequence, so the parity sequences are better protection against bit errors than the message bits themselves
- If we're using multiple generators, construct the transmit sequence by interleaving the bits of the parity sequences:

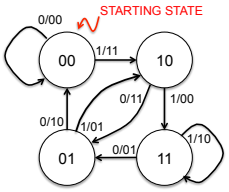
$$xmit = p_0[0], p_1[0], p_0[1], p_1[1], p_0[2], p_1[2], \ldots$$

- Code rate is 1/number_of_generators
  - 2 generator polynomials → rate = ½
  - Engineering tradeoff: using more generator polynomials improves bit-error correction but decreases the number of message bits/sec that can be transmitted

Example

- Using two generator polynomials:
  - g_0 = 1, 1, 1, 0, 0, … abbreviated as 111 (or "7") for K=3 code
  - g_1 = 1, 1, 0, 0, 0, … abbreviated as 110 (or "6") for K=3 code
- Writing out the equations for the parity sequences (in GF(2)):
  - $p_0[n] = (x[n] + x[n-1] + x[n-2])$
  - $p_1[n] = (x[n] + x[n-1])$
- Let x[n] = [1,0,1,1,…]; as usual x[n]=0 when n<0:
  - $p_0[0] = (1 + 0 + 0) = 1$, $p_1[0] = (1 + 0) = 1$
  - $p_0[1] = (0 + 1 + 0) = 1$, $p_1[1] = (0 + 1) = 1$
  - $p_0[2] = (1 + 0 + 1) = 0$, $p_1[2] = (1 + 0) = 1$
  - $p_0[3] = (1 + 1 + 0) = 0$, $p_1[3] = (1 + 1) = 0$
- Transmit: 1, 1, 1, 1, 0, 1, 0, 0, …

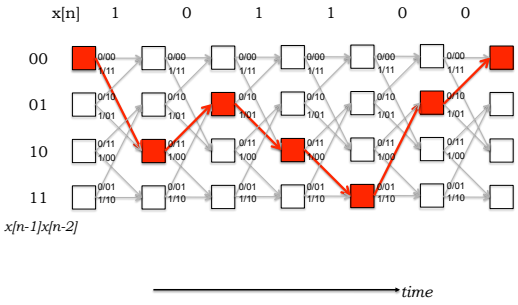
State Machine View



The state machine is the same for all K=3 codes. Only the p_i labels change depending on number and values for the generator polynomials.

- Example: K=3, rate ½ convolutional code
- States labeled with x[n-1] x[n-2]
- Arcs labeled with x[n]/p_0p_1
- msg=101100; xmit = 11 11 01 00 01 10

Trellis View @ Transmitter



Using Convolutional Codes

- Transmitter
  - Beginning at starting state, processes message bit-by-bit
  - For each message bit: makes a state transition, sends p_i
  - Pad message with K-1 zeros to ensure return to starting state
- Receiver
  - Doesn't have direct knowledge of transmitter's state transitions; only knows (possibly corrupted) received p_i
  - Must find most likely sequence of transmitter states that could have generated the received p_i
  - If BER is small, prob(more errors) < prob(fewer errors)
    - Most likely message sequence is the one that generated the sequence of parity bits with the smallest Hamming distance from the actual received p_i, i.e., where we minimize the number of bit errors that explains how the transmit sequence was corrupted to produce the received p_i

Example

- Using K=3, rate ½ code from earlier slides
- Received: 111011000110
- Some errors have occurred…
- What's the 4-bit message?
- Look for message whose xmit bits are closest to rcvd bits

Most likely: 1011

Msg	Xmit*	Rcvd	d
0000	000000000000	111011000110	7
0001	000000111110		8
0010	000011111000		8
0011	000011010110		4
0100	001111100000		6
0101	001111011110		5
0110	001101001000		7
0111	001100100110		6
1000	111110000000		4
1001	111110111110		5
1010	111101111000		7
1011	111101000110		2
1100	110001100000		5
1101	110001011110		4
1110	110010011000		6
1111	110010100110		3

*Msg padded with 2 zeroes before xmit