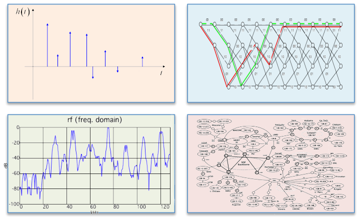


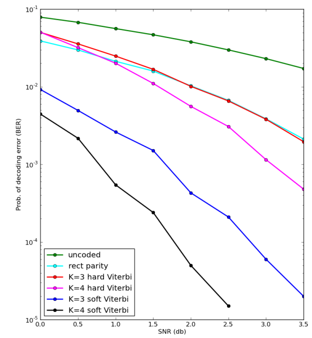
INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

6.02 Fall 2011
Lecture #7

- Convolutional codes
- State-machine view & trellis
- *Most likely* message to have been transmitted

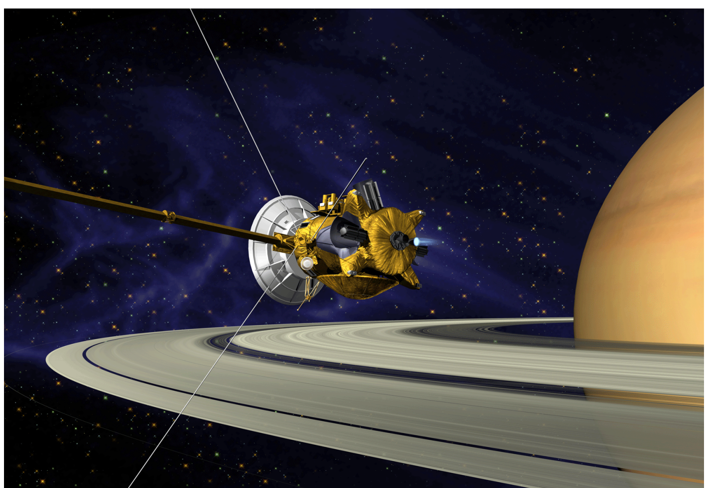
Do We Need Better Channel Coding?



The graph shows how a rate ½ "rectangular" block code experimentally improves over using no coding at all, especially at higher SNRs (lower overall BER).

But there's considerable room for improvement.

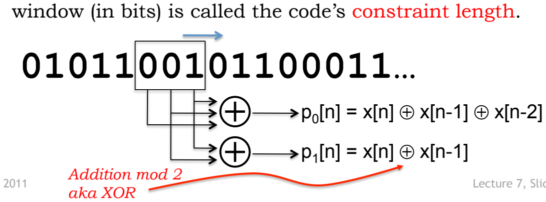
Can we find more effective rate-½ codes, for instance?



Phoning home using a k=15, rate=1/6 convolutional code

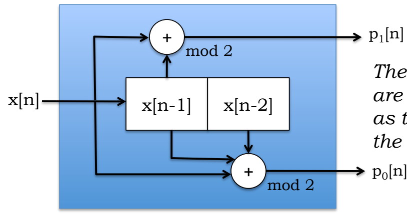
Convolutional Codes

- Like the block codes discussed earlier, send parity bits computed from blocks of message bits
  - Unlike block codes, don't send message bits, send only the parity bits!
  - The code rate of a convolutional code tells you how many parity bits are sent for each message bit. We'll mostly be talking about rate $1/r$ codes.
  - 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.



Shift-register View

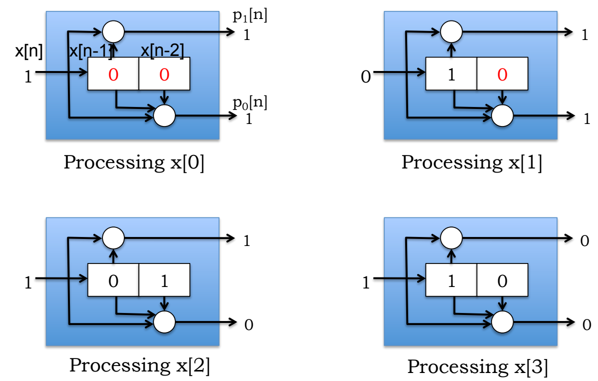
- One often sees convolutional encoders described with a block diagram like the following:



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 from 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, the saved message bits are *shifted right* by one, and the incoming bit moves into the left position.

Example: Transmit message 1011



$p_0[n] = x[n] \oplus x[n-1] \oplus x[n-2]$

$p_1[n] = x[n] \oplus x[n-1]$

Parity Bit Equations

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

I can see why they call it a convolutional code

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

- *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 (in general)*

- 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; the simplest choice is to use 2 generator polynomials

Convolutional Codes (cont'd.)

- We'll transmit the parity sequences, not the message itself
  - As we'll see, we 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 sequence itself

- 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 rate of the code (the number of message bits/s that can be transmitted)

Example

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

Example Generator Polynomials

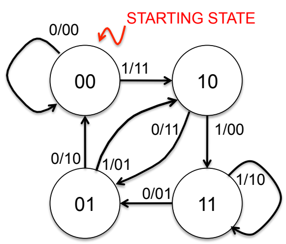
Table 1-Generator Polynomials found by Busgang for good rate ½ codes

Constraint Length	G_1	G_2
3	110	111
4	1101	1110
5	11010	11101
6	110101	111011
7	110101	110101
8	110111	1110011
9	110111	111001101
10	110111001	1110011001

www.complextoreal.com

Next lecture: Concept of *free distance* of a convolutional code, as a measure of its error correction power

State-Machine View



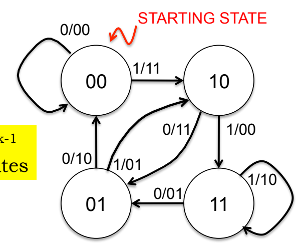
$p_0[n] = x[n] \oplus x[n-1] \oplus x[n-2]$

$p_1[n] = x[n] \oplus x[n-1]$

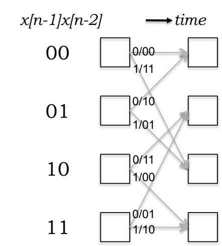
The state machine is the *same* for all K=3 codes. Only the p_i labels change depending on number and values for the generator polynomials.

- 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

From a State Machine to a *Trellis*

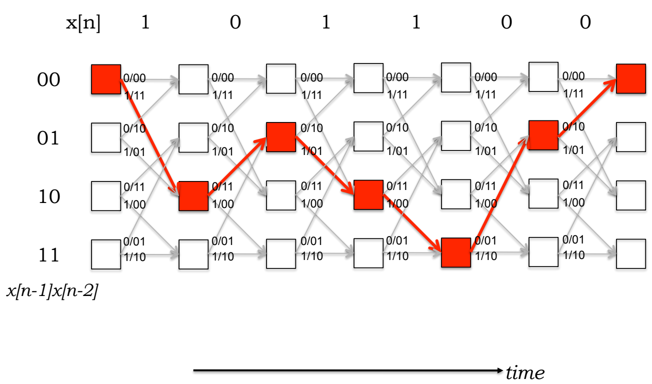


2^{k-1} states



- Example: K=3, rate-½ convolutional code
  - $G_0 = 111$: $p_0 = 1{*}x[n] \oplus 1{*}x[n-1] \oplus 1{*}x[n-2]$
  - $G_1 = 110$: $p_1 = 1{*}x[n] \oplus 1{*}x[n-1] \oplus 0{*}x[n-2]$
- States labeled with *x[n-1] x[n-2]*
- Arcs labeled with $x[n]/p_0p_1$

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 parity bits, p_i
  - Must find most likely sequence of transmitter states that could have generated the received parity bits, p_i
  - If BER < ½, P(more errors) < P(fewer errors)
  - Theorem: When BER < ½, maximum-likelihood message sequence is the one that generated the codeword (here, sequence of parity bits) with the smallest Hamming distance from the received codeword (here, parity bits)
  - I.e., find nearest valid codeword *closest* to the received codeword

Example

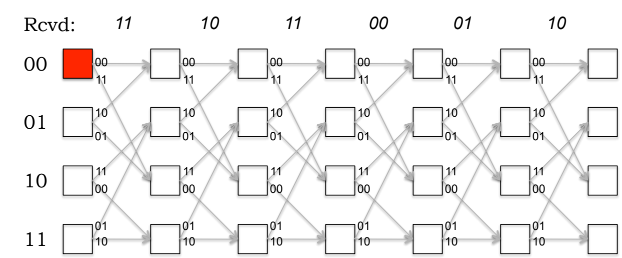
- Using K=3, rate-½ code from earlier slides
- Received: 111011000110
- Some errors have occurred…
- What's the 4-bit message?
- Look for message whose xmit bits are closest to rcvd bits

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

Decoding: Finding the Maximum-Likelihood Path



Given the received parity bits, the receiver must find the most-likely sequence of transmitter states, i.e., the path through the trellis that minimizes the Hamming distance between the received parity bits and the parity bits the transmitter would have sent had it followed that state sequence.

One solution: Viterbi decoding – come to the next lecture!