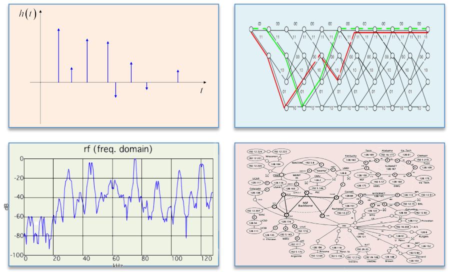


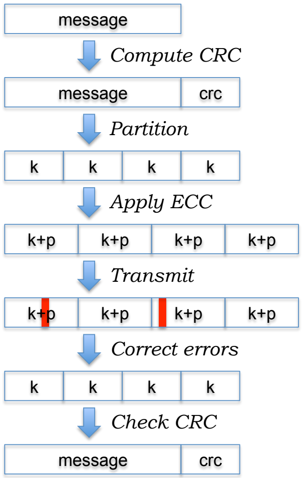
INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

6.02 Spring 2009
Lecture #8

- using SEC/CRC in digital transmissions
- impulse noise, burst errors, interleaving
- convolutional coding, state & trellis diagrams
- hidden Markov models

Digital Transmission using ECC


message

Compute CRC

message | crc

Partition

k | k | k | k

Apply ECC

k+p | k+p | k+p | k+p

Transmit

k+p | k+p | k+p | k+p

Correct errors

k | k | k | k

Check CRC

message | crc

- Start with original message
- Add CRC to enable verification of error-free transmission
- Apply ECC, adding parity bits to each k-bit block of the message. Our ECCs were designed for *single-bit error correction*. Number of parity bits (p) depends on code:
  - Replication: p grows as O(k)
  - Rectangular: p grows as O(√k)
  - Hamming: p grows as O(log k)
- After xmit, correct errors
- Verify CRC, fails if undetected/ uncorrectable error
- Deliver or discard message

Is Single-bit Error Correction Enough?

p(2 or more errors) = 1 – p(no errors) – p(exactly one error)

$$= 1 - (1 - \mathrm{BER})^k - k{*}\mathrm{BER}{*}(1-\mathrm{BER})^{k-1}$$

		BER				
p(≥2 errors)		10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}
k	8	2.8e-05	2.8e-07	2.8e-09	2.8e-11	2.8e-13
	32	4.9e-04	5.0e-06	5.0e-08	5.0e-10	5.0e-12
	256	2.8e-02	3.2e-04	3.3e-06	3.3e-08	3.3e-10
	1024	2.7e-01	4.9e-03	5.2e-05	5.2e-07	5.2e-09
	8192	1.0e+00	2.0e-01	3.2e-03	3.3e-05	3.4e-07

Conclusion: Yes, SEC is okay if BER isn’t too big and we keep k small. Some errors still get through but are caught by CRC check; deal with discarded messages at higher level of protocol.

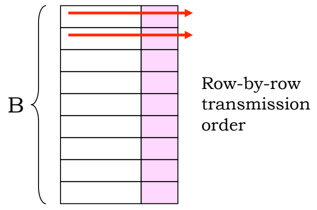
Noise models

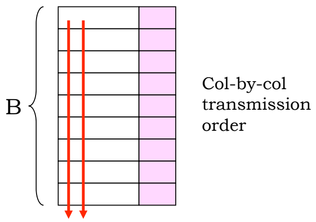
- Gaussian noise
  - Equal chance of noise at each sample
  - Gaussian PDF: low probability of large amplitude
  - Good for modeling total effect of many small, random noise sources
- Impulse noise
  - Infrequent bursts of high-amplitude noise, e.g., on a wireless channel
  - Some number of consecutive bits lost, bounded by some burst length B
  - Single-bit error correction seems like it’s useless for dealing with impulse noise…
    or is it???

Correcting single-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 error correction help with that?

Dealing with Burst Errors

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

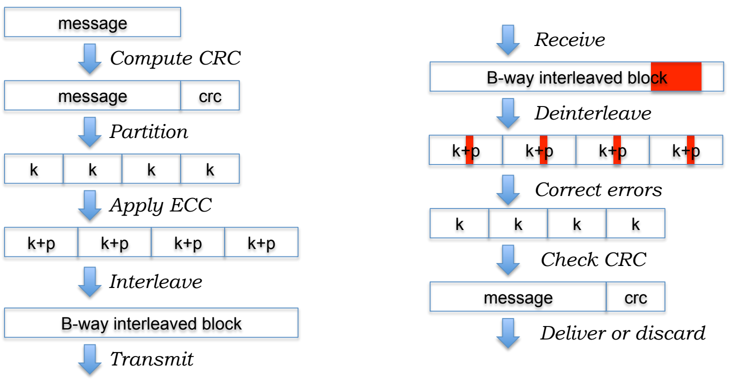




Problem: Bits from a particular codeword are transmitted sequentially, so a B-bit burst produces multi-bit errors.

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

Interleaving



Framing

- The receiver needs to know
  - the beginning of the B-way interleaved block in order to do deinterleaving
  - the beginning of each ECC block in order to do error correction.
  - Since the interleaved block is made up of B ECC blocks, knowing where the interleaved block begins automatically supplies the necessary start info for the ECC blocks
- Framing is accomplished by having the transmitter insert sync sequences to mark beginnings…
  - Data and parity bits must not have patterns that can be confused with the sync pattern
  - Syncs are themselves subject to error
  - Some channels have natural boundary indicators, e.g., the beginning of transmission.

Sync techniques

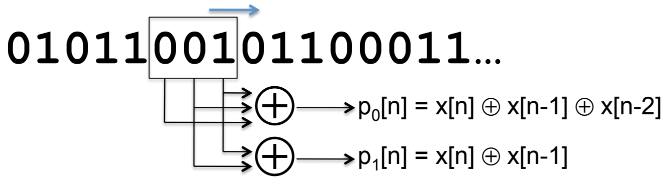
- Recode bit stream to ensure sync uniqueness
- 8b/10b recoding provides for several unique patterns useful for sync and other out-of-band information
  - Used on wired channels where BER is small, which means sync is seldom corrupted during transmission
- Choose sync pattern that has, say, 5 1's in a row
  - To prevent sync from appearing in message, "bit-stuff" 0's after any sequence of four 1's in the message.
  - This step is easily reversed at receiver (just remove 0 after any sequence of four consecutive 1's in the message).
  - Creates variable-length blocks, a slight pain
  - Less overhead than 8b/10b if you don't need 8b/10b's other benefits

Remaining agenda items

- With M ECC blocks per message, we can correct somewhere between 1 and M errors depending on where in the message they occur.
  - Can we make an ECC that corrects up to E errors without any constraints where errors occur?
  - Yes! Reed-Solomon codes, discussed next lecture
- Framing is necessary, but the sync itself can't be protected by an ECC scheme that requires framing.
  - This makes life hard for channels with higher BERs
  - Is there an error correction scheme that works on un-framed bit streams?
  - Yes! Convolutional codes: encoding discussed now and the clever decoding scheme will be discussed next week.

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, 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 be talking about rate 1/p 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.



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

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

Why "convolutional"?

- Each parity bit at bit n are computed using a formula of the form $\Sigma\, g[i]x[n-i] = G*X$
  - Looks just like convolution in LTI systems
  - G_{p0} = 1, 1, 1, 0, 0, … abbreviated as 111 for k=3 code
  - G_{p1} = 1, 1, 0, 0, 0, … abbreviated as 110 for k=3 code
- What are the "good" generator functions?

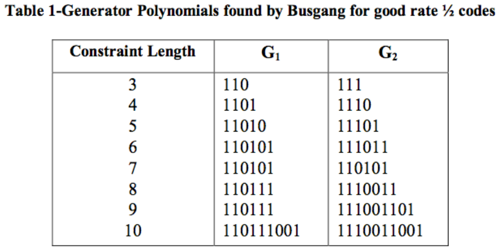
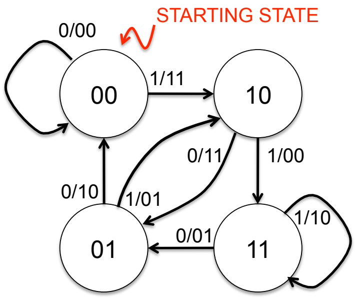


Table 1-Generator Polynomials found by Busgang for good rate ½ codes

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

State Machine View



- 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=1011; xmit = 11 11 01 00

Using Convolutional Codes

- Transmitter
  - Begining at starting state, processes message bit-by-bit
  - For each message bit: makes a state transition, sends p_i
  - Pad message with k 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
  - “most likely” is measured by the number of bit errors that had to have occurred to have produced the received p_i from the transmitted p_i – the fewer errors, the more likely that particular sequence of transmitter states.

Example

- Using k=3, rate ½ code from earlier slides
- Received: 11101100011000
- Some errors have occurred…
- What’s the 4-bit message?
- Look for message whose xmit bits are closest to rcvd bits

Msg	*Xmit*	*Rcvd*	*d*
0000	00000000000000	11101100011000	7
0001	00000011111000		8
0010	00001111100000		8
0011	00001101011000		4
0100	00111110000000		6
0101	00111101111000		5
0110	00110100100000		7
0111	00110010011000		6
1000	11111000000000		4
1001	11111011111000		5
1010	11110111100000		7
1011	11110100011000		2
1100	11000110000000		5
1101	11000101111000		4
1110	11001001100000		6
1111	11001010011000		3

Most likely: 1011