Lecture 18: Error Detection and Correction

John M Pauly

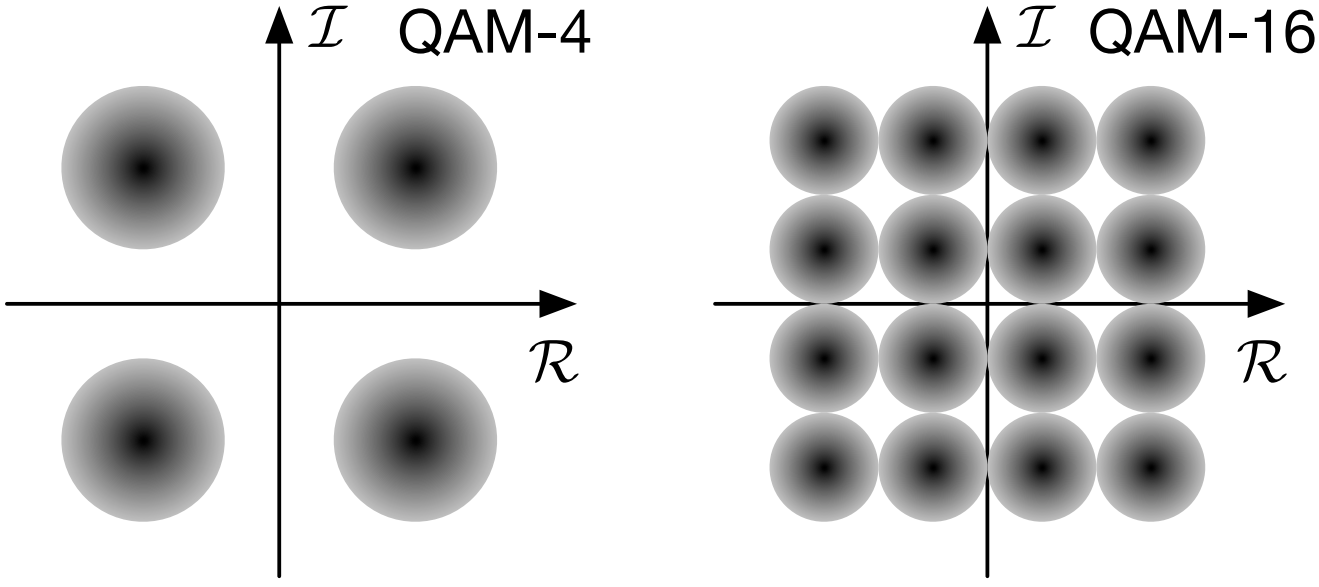
November 27, 2021

Error Protection: Detection and Correction

- Communication channels are subject to noise.
- Noise distorts analog signals.
- Noise can cause digital signals to be received as different values.
  - Bits can be flipped
  - Points in a signal constellation can be shifted.
- Changes in a digital signal are called *errors*.

Effects of Noise

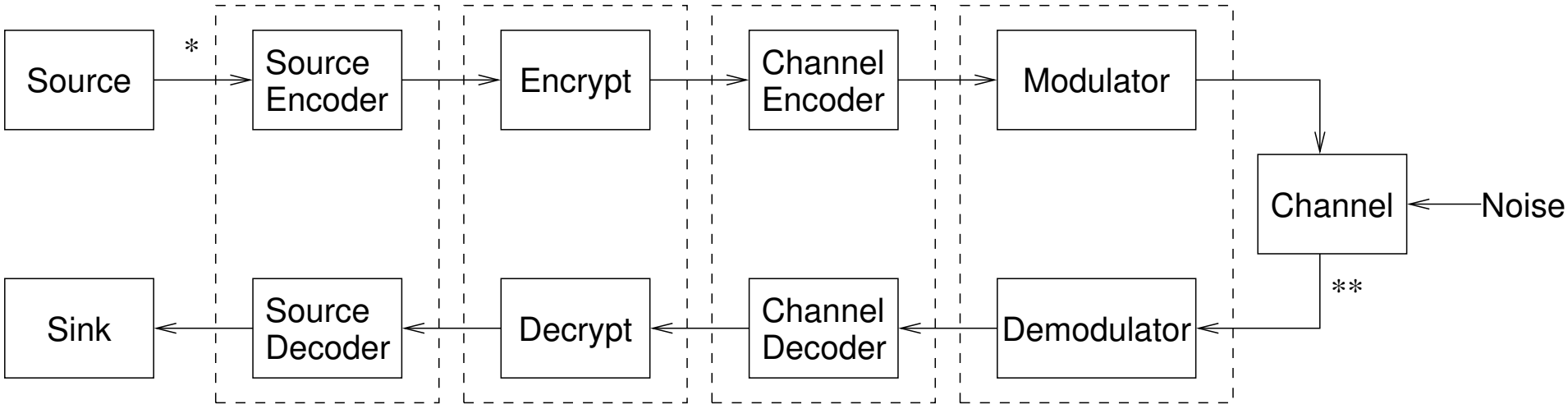
Noise causes the constellation to blur out,



- With enough noise we will get decoding errors
- We'd like to be able to detect, and perhaps correct, these errors
- To do this, we'll add some redundant information

Communication Systems

Recall the communication system block diagram:



We have concentrated on the modulator/demodulator blocks.

Error protection involves channel encoder and decoder.

Error Control Classification

The error control problem can be classified in several ways.

- Type of errors: how much clustering — random, burst, catastrophic
- Type of modulator output: digital ("hard") vs. analog ("soft")
- Type of error control coding: detection vs. correction
- Type of codes: block vs. convolutional
- Type of decoder: algebraic vs. probabilistic

The first two classifications are used to select a coding scheme according to the last three classifications.

Types of Error Potection

- Error detection

  Goal: avoid accepting faulty data.

  Lost data may be unfortunate; wrong data may be disastrous.

  Solution: *checksums* are included in messages (packets, frames, sectors).

  If any part of the message is altered, then the checksum is *not* valid (with high probability).

- (Forward) error correction (FEC or ECC).

  Use redundancy in encoded message to estimate from the received data (*senseword*) what message was actually sent.

  Optimal estimate is message that is most probable given what is received (MAP, *maximum a posteriori*). The best estimate is typically the message that is "closest" to the senseword.

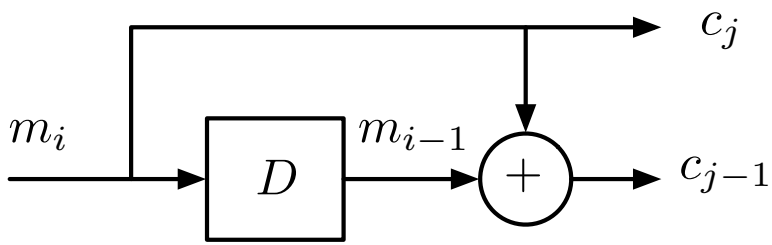
Block Codes

- k-digit blocks of information digits are encoded into n-digit codewords ($n \geq k$) by adding $p = n - k$ redundant check digits.

- There is no memory betwen blocks. The encoding of each data block is independent of past and future blocks.
- An encoding in which the information digits appear unchanged in the codewords is called *systematic*.
  - Adding parity bits is systematic
  - 8b/10b is not systematic (each 8 bit sequence is mapped onto one or two 10 bit sequences)

Convolutional Codes

- For convolutional codes, a filter continuously adds additional bits to the bit stream
- Each n-bit codeword block depends on the current codeword and on the previous m codewords. m is the memory order
- Here is the simplest convolutional code.



- This generates two output bits for every input bit, the current data bit, and the XOR with the previous data bit.
- For this rate $1/2$ convolutional code, $m = 1$ and $n = 2$. This is used in GSM.
- Convolutional codes are widely used in digital communications. They don't need a fixed block size. Examples are Viterbi codes and trellis codes. These are covered in EE279.

Block Codes: Simple Parity-Check Codes

Append one check bit to data bits so that all codewords have the same overall parity — either even or odd.

Even-parity codewords are defined by a single *parity-check equation*:

$$c_1 \oplus c_2 \oplus \cdots \oplus c_n \;=\; (c_1 + c_2 + \cdots + c_n) \bmod 2 \;=\; 0\,,$$

where $\oplus$ denotes the exclusive-or operation.

If we XOR c_n to both sides of the above equation, we obtain an *encoding equation*:

$$c_n \;=\; c_1 \oplus c_2 \oplus \cdots \oplus c_{n-1}\,.$$

This shows how to compute the *check bit* c_n from the *data bits* $c_1, \ldots, c_{n-1}$.

Any single bit error (or any odd number of errors) can be detected.

Any bit c_i can be considered to be the check bit because it can be computed from the other $n-1$ bits: $c_i = \sum_{j \neq i} c_j$.

Polynomial Division

- We can consider parity as the remainder after polynomial division.
- For data bits $c_1, \cdots, c_7$ the parity bit is the remainder after dividing

$$c_1x^6 + c_2x^5 + c_3x^4 + c_4x^3 + c_5x^2 + c_6^x + c_7$$

by $x + 1$.

- Example: $c_1, \cdots, c_7 = 1010001$ divided by 11

Encoding

```
          1 1 0 0 0 1 R 1
      ┌──────────────
1 1   │ 1 0 1 0 0 0 1
        1 1
        ─────
          1 1 0 0 0 1
          1 1
          ─────
          0 0 0 0 0 1
                  1 1
                  ─────
                    1   Remainder
```

Decoding

```
          1 1 0 0 0 1 0
      ┌──────────────
1 1   │ 1 0 1 0 0 0 1 [1]   Parity Bit
        1 1
        ─────
          1 1 0 0 0 1 1
          1 1
          ─────
          0 0 0 0 0 1 1
                  1 1
                  ─────
                    0   Remainder
```

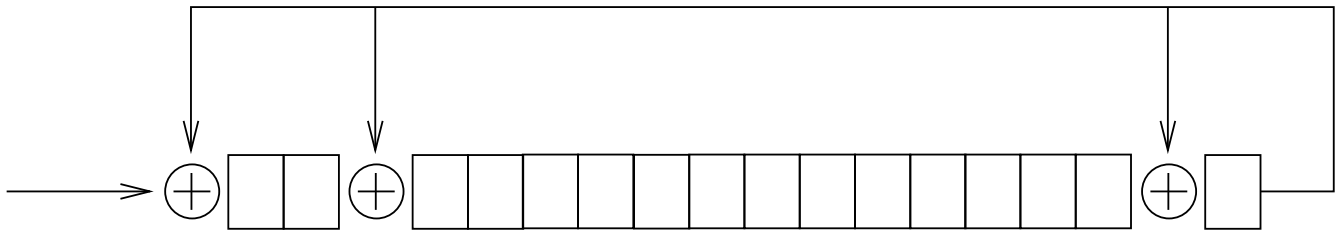
Addition and subtraction are without carry (XOR)

Cyclic Redundancy Check (CRC)

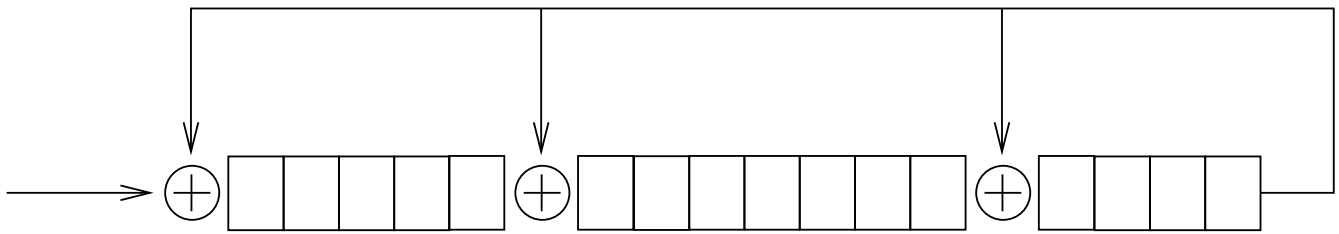
- If we use a longer divisor polynomial we can detect which bit is in error for a much longer bit stream
- For an optimal polynomial of order n we can
  - Detect any single or double bit error in $2^n - 1$ bits
  - With a factor $1 + x$, detect any error in an odd number of bits
- Lots of different polynomials are possible. Two very common ones are
  - CRC-16-IBM : $x^{16} + x^{15} + x^2 + 1$
  - CRC-CCITT : $x^{16} + x^{12} + x^5 + 1$
- Other CRC codes from CRC-3 to CRC-64
- Can be applied to any length block less than $2^n - 1$
- Used in Bluetooth, USB, GSM, X.25, ACARS (ACARS-16-ARINC)

CRC Implementation

- Polynomial division can be implemented using linear feedback shift registers. This does what we do with long division, clocking in one new bit each cycle.
  CRC-16:



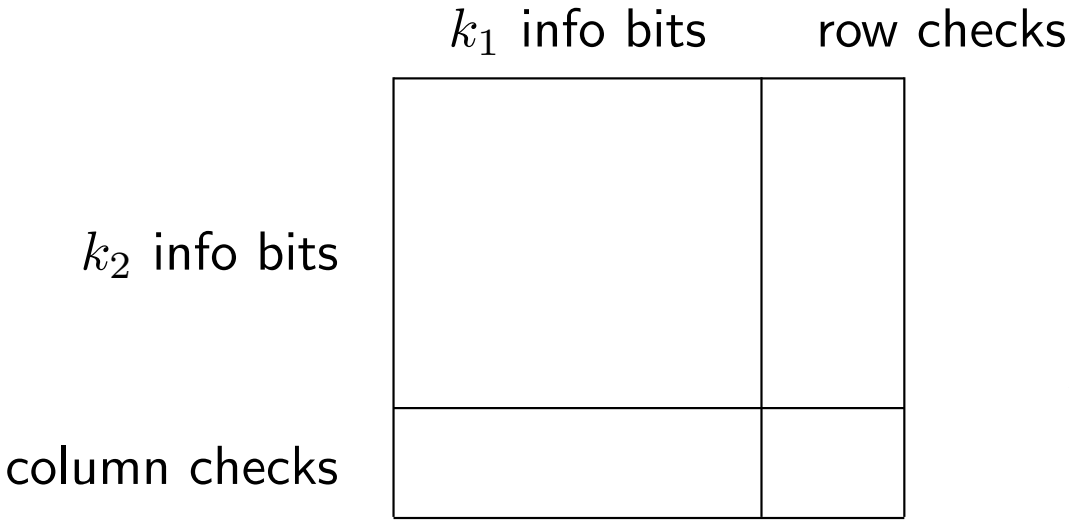
  CRC-CCITT:



- Remainder is left in the shift register at the end of the calculation, appended to the transmitted bits
- On reception, the division of the data with the appended CRC remainder will produce 0 if there are no errors
- If there is a single error, the CRC remainder identifies which bit is in error

Error Correction: Simple Product Codes

Arrange data bits in a two-dimensional array. Append parity check bits at the end of each row and column.



$$\begin{aligned} n &= (k_1+1)(k_2+1) \\ k &= k_1 k_2 \\ n-k &= k_1 + k_2 + 1 \end{aligned}$$

Single error causes failure of one row equation and one column equation. Incorrect bit is located at the intersection of the bad row and bad column.

Double errors can be detected — two rows or two columns (or both) have the wrong parity — but cannot be corrected.

Some triple errors cause miscorrection. Which?

Error Correction: Hamming Codes

Simple product codes are simple but inefficient:

- a failed parity-check equation locates row or column of error
- however, a satisfied equation gives little information

An "efficient" equation gives one bit of information about the error location. It "looks" at half the codeword bits and is "independent" of other equations.

Basic idea:

- We will send a block of size $n = 2^m - 1$
- We'll use m bits for parity, and the rest for data
- Each parity bit corresponds to half of the block
- A single error and its location can be identified by which parity equations fail. This pattern is called the *error syndrome*

Hamming Codes

To determine the parity equations,

- Index each output bit from one to $2^n - 1$, and represent it in binary
- Any index that has a single bit is used for parity (i.e. 00100)
- All other indexed outputs are sequentially filled with input bits
- The parity bit is computed for all indexes that have the same index bit set (i.e all the odd samples for the first parity bit)

Example: $2^n - 1 = 7$

- Parity bits are 001, 010, and 100
- Data bits are the rest, 011, 101, 110, and 111
- We'll have 3 parity bits, 4 data bits, and seven bits total
- This is called a (7,4) code

Hamming Codes

- The following table defines a $(7, 4)$ Hamming parity-check code.

	c_1	c_2	c_3	c_4	c_5	c_6	c_7
	p_1	p_2	d_1	p_4	d_2	d_3	d_4
p_1	1	0	1	0	1	0	1
p_2	0	1	1	0	0	1	1
p_4	0	0	0	1	1	1	1

} 3 parity-check equations

- Each column is the binary representation of that index
- Each row defines one parity bit equation
- The 1's indicate which codeword bits affect which parity-check equations.
- We can double the block size and only need one more parity bit, for a (15,11) code. This gets very efficient as n gets large.

Hamming Codes: Parity-Check Equations and Matrix

The following three equations are satisfied by all (and only) valid codewords:

$$c_1 \oplus c_3 \oplus c_5 \oplus c_7 = 0$$
$$c_2 \oplus c_3 \oplus c_6 \oplus c_7 = 0$$
$$c_4 \oplus c_5 \oplus c_6 \oplus c_7 = 0$$

The check equations can be described by a *parity-check* matrix:

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Valid codewords are characterized the equation:

$$H \begin{bmatrix} c_1 \\ \vdots \\ c_7 \end{bmatrix} = 0_{3\times 1}$$

In other words, a 7-tuple $\mathbf{c}$ is a codeword if and only if $H\mathbf{c}^T = 0$.

Hamming Codes: Encoding Equations

Each of the parity codeword bits c_1, c_2, c_4 appears in only one equation.

The parity bits c_1, c_2, c_4 are computed from the data bits, c_3, c_5, c_6, c_7.

$$
\begin{aligned}
c_1 &= c_3 \oplus c_5 \oplus c_7 \\
c_2 &= c_3 \oplus c_6 \oplus c_7 \\
c_4 &= c_5 \oplus c_6 \oplus c_7
\end{aligned}
$$

These *linear* encoder equations can be written as a vector-matrix product.

$$
[\, c_1\ c_2\ c_4 \,] = [\, c_3\ c_5\ c_6\ c_7 \,] P = [\, c_3\ c_5\ c_6\ c_7 \,] \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.
$$

Hamming Code Error Detection and Correction

The parity equations are

$$c_1 \oplus c_3 \oplus c_5 \oplus c_7 = 0$$
$$c_2 \oplus c_3 \oplus c_6 \oplus c_7 = 0$$
$$c_4 \oplus c_5 \oplus c_6 \oplus c_7 = 0$$

where $c_1 = p_1$, $c_2 = p_2$, and $c_4 = p_4$ are parity bits.

Parity bit error

- Each parity bit occurs in just one equation, so only that equation will have a parity error
- A single parity equation error is an error in that parity bit

Data bit error

- Each data bit occurs in multiple equations, each of which will have a parity error
- The data bit error is the bit shared by those multiple equations

Either way, a single bit error can be detected and corrected.

Hamming Code Error Detection and Correction

For larger blocks, we just add more parity bits.

For example, the parity-check matrix for (15,11) is

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

- There are now four parity equations, with parity bits p_1, p_2, p_4, and p_8
- The data bits appear in 2, 3, or 4 equations.
- A single parity equation error is an error in the parity bit
- Multiple parity equation errors is an error in the bit shared by these equations

All single bit errors can be detected and corrected.

Error Detection Conclusion

- A single overall parity-check equation detects single errors.
- Hamming codes use m equations to correct one error in $2^m - 1$ bits.
  - Most useful when the error rate is low. ECC RAM uses Hamming codes
  - More sophisticated algorithms needed when multiple errors likely in a block
- Extensions
  - We can use nonbinary equations if we create *symbols* from sequences of bits. E.g., four bits can represent $0, 1, \ldots, 15$.
  - Nonbinary check equations can use more advanced arithmetic, such as mod 16.
  - We can add other types of equations, $c_1 + 2c_2 + 3c_3 + \cdots$ and $c_1 + \alpha c_2 + \alpha^2 c_3 + \cdots$ to be able to detect and correct multiple bit errors.

Next Classes

Wednesday : Spread Spectrum, Radar, GPS, CDMA

Friday : Questions on projects

Following Friday : Projects due