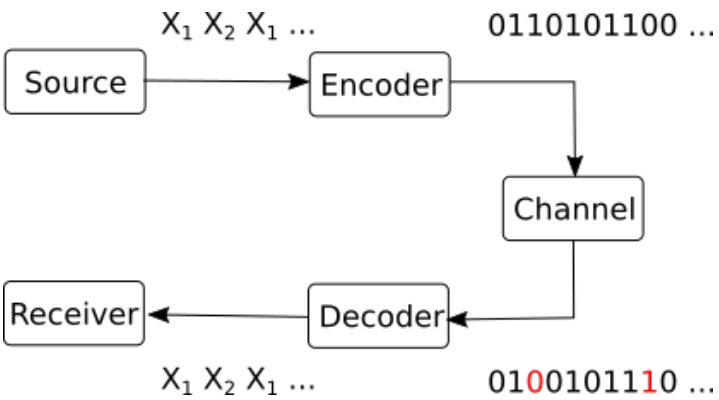
01_ErrorControlCoding

October 20, 2016

0.0.1 What is error control coding?



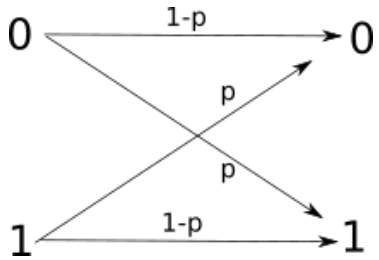
Communication system

- The second main task of coding: **error control**
- Protect information against channel errors

0.0.2 Mutual information and error control

- Mutual information $I(X,Y)$ = the information transmitted on the channel
- Even though there is information transmitted on the channel, when the channel is noisy the information is **not enough**

Example: consider the following BSC channel (p = 0.01, $p(x_1) = 0.5$, $p(x_2) = 0.5$):



Binary symmetric channel (BSC)

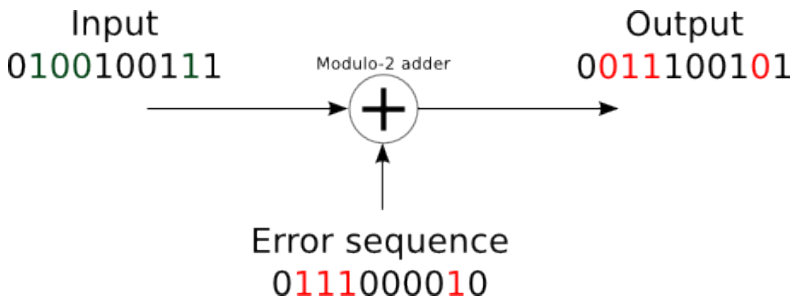
- The receiver would like to know the source messages
  - In absence of communication, the uncertainty is $H(X) = 1$ bit/msg
  - With communication, the uncertainty is $H(X|Y) \approx 0.081$ bit/msg
  - $I(X,Y) = H(X) - H(X|Y) = \approx 0.919$ bit/msg
- Even though we have large I(X,Y), about 1% of all bits are erroneuos
  - Imagine downloading a file, but having 1% wrong bits

0.0.3 Why is error control needed?

- In most communications it is required that *all* bits are received correctly
  - Not 1% errors, not 0.1%, not 0.0001%. **None!**
- But that is not possible unless the channel is ideal.
- So what do to? **Error control coding**

0.0.4 Modelling the errors on the channel

- We consider only binary channels (symbols = $\{0, 1\}$)
- An error = a bit is changed from 0 to 1 or viceversa
- Changing the value of a bit = modulo-2 sum with 1
- Value of a bit remains the same = modulo-2 sum with 0



Channel error model

- Channel model we use (simple):
  - The transmitted sequence is summed modulo-2 with an **error sequence**
  - Where the error sequence is 1, there is a bit error
  - Where the error sequence is 0, there is no error

$$\mathbf{r} = \mathbf{c} \oplus \mathbf{e}$$

0.0.5 Error detection and error correction

Binary error correction:

- For binary channels, know the location of error => fix error by inverting bit
- Locating error = correcting error

Two possibilities in practice:

- **Error detection**: find out if there is any error in the received sequence
  - don't know exactly where, so cannot correct the bits, but can discard whole sequence
  - perhaps ask the sender to retransmit (examples: TCP/IP, internet communication etc)
  - easier to do
- **Error correction**: find out exactly which bits have errors, if any
  - can correct all errored bits by inverting them
  - useful when can't retransmit (data is stored: on HDD, AudioCD etc.)
  - harder to do than mere detection

0.0.6 What is error control coding?

The process of error control:

1. Want to send a sequence of k bits = **information word**

$$\mathbf{i} = i_1 i_2 ... i_k$$

2. For each possible information word, the coder assigns a **codeword** of length $n > k$:

$$\mathbf{c} = c_1 c_2 ... c_n$$

3. The codeword is sent on the channel instead of the original information word
4. The receiver receives a sequence $\mathbf{r} = \mathbf{c} + \mathbf{e}$, with possible errors:

$$\mathbf{r} = r_1 r_2 ... r_n$$

5. The decoding algorithm detects/corrects the errors in $\mathbf{r}$

0.0.7 Definitions

- An **error correcting code** is an association between the set of all possible information words to a set of codewords
  - Each possible information word $\mathbf{i}$ has a certain codeword $\mathbf{c}$
- The association can be done:
  - randomly: codewords are selected and associated randomly to the information words

  - based on a certain rule: the codeword is computed with some algorithm from the information word

- A code is a **block code** if it operates with words of *fixed size*

  - Size of information word $\mathbf{i} = k$, size of codeword $\mathbf{c} = n$, $n > k$
  - Otherwise it is a *non-block code*

- A code is **linear** if any linear combination of codewords is also a codeword

- The **coding rate** of a code is:

$$R = \frac{k}{n}$$

0.0.8 Definitions

- A code C is an t**-error-detecting** code if it is able to *detect* t errors

- A code C is an t**-error-correcting** code if it is able to *correct* t errors

0.0.9 Hamming distance

- The **Hamming distance** of two binary sequences a, b of length n = the total number of bit differences between them

$$d_H(a,b) = \sum_{i=1}^{N} a_i \bigoplus b_i$$

- We need at least $d_H(a,b)$ bit changes to convert one sequence into another

- It satisfies the 3 properties of a metric function:

  1. $d(a,b) \geq 0 \forall a,b$, with $d(a,b) = 0 \Leftrightarrow a = b$
  2. $d(a,b) = d(b,a), \forall a,b$
  3. $d(a,c) \leq d(a,b) + d(b,c), \forall a,b,c$

- The **minimum Hamming distance of a code**, $d_{H\,min}$ = the minimum Hamming distance between any two codewords $\mathbf{c_1}$ and $\mathbf{c_2}$

- Example at blackboard

0.0.10 Linear block codes

- A code is a **block code** if it operates with words of *fixed size*

  - Size of information word $\mathbf{i} = k$, size of codeword $\mathbf{c} = n$, $n > k$
  - Otherwise it is a *non-block code*

- A code is **linear** if any linear combination of codewords is also a codeword

- A code is called **systematic** if the codeword contains all the information bits explicitly, unaltered

  - coding merely adds supplementary bits besides the information bits
  - codeword has two parts: the information bits and the parity bits
  - example: parity bit added after the information bits

- Otherwise the code is called **non-systematic**

  - the information bits are not explicitly visible in the codeword

- Example: at blackboard

0.0.11 Generator matrix

- All codewords for a linear block code can be generated via a matrix multiplication:

$$\mathbf{i} \cdot [G] = \mathbf{c}$$

Codeword construction with generator matrix

- $[G]$ = **generator matrix** of size $k \times n$
- All operations are done in modulo-2 arithmetic:

  - $0 \oplus 0 = 0, 0 \oplus 1 = 1, 1 \oplus 0 = 1, 1 \oplus 1 = 0$
  - multiplications as usual

- Row-wise interpretation:

  - $\mathbf{c}$ = a linear combination of rows in $[G]$
  - The rows of $[G]$ = a *basis* for the linear code

0.0.12 Parity check matrix

- How to check if a binary word is a codeword or not
- Every $k \times n$ generator matrix $[G]$ has complementary matrix $[H]$ such that

$$0 = [H] \cdot [G]^T$$

- For every codeword $\mathbf{c}$ generated with $[G]$:

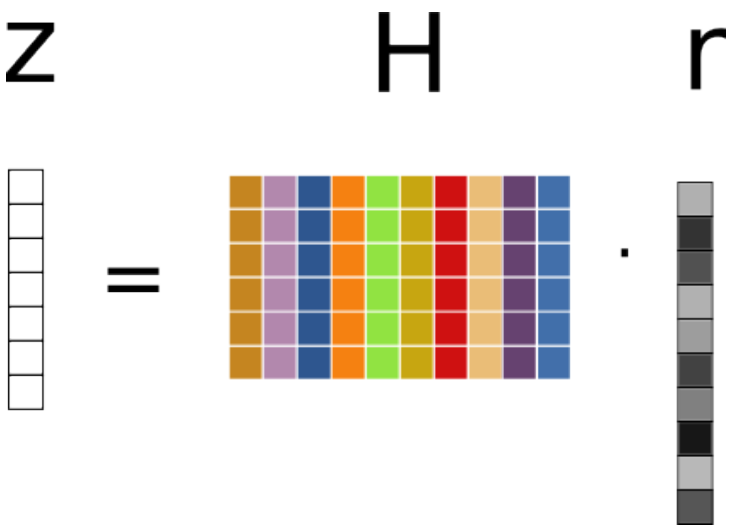
$$\boxed{0 = [H] \cdot \mathbf{c}^T}$$

- because:

$$\mathbf{i} \cdot [G] = \mathbf{c}$$
$$[G]^T \cdot \mathbf{i}^T = \mathbf{c}^T$$
$$[H] \cdot \mathbf{c}^T = [H] \cdot [G]^T \cdot \mathbf{i}^T = 0$$

0.0.13 Parity check matrix

- $[H]$ is the **parity-check matrix**, size = $(n-k) \times n$
- $[G]$ and $[H]$ are related, one can be deduced from the other
- The resulting vector $z = [H] \cdot [c]^T$ is the **syndrome**
- All codewords generated with $[G]$ will produce 0 when multiplied with $[H]$
- All binary sequences that are not codewords will produce $\neq 0$ when multiplied with $[H]$
- Column-wise interpretation of multiplication:



Codeword checking with parity-check matrix

0.0.14 [G] and [H] for systematic codes

- For systematic codes, [G] and [H] have special forms
- Generator matrix
  - first part = some matrix Q
  - second part = identity matrix

$$[G]_{k \times n} = [Q_{k \times (n-k)} \;\; I_{k \times k}]$$

- Parity-check matrix

  - first part = identity matrix
  - second part = same Q, transposed

$$[H]_{(n-k)\times n} = [I_{(n-k)\times(n-k)} \;\; Q^T_{(n-k)\times k}]$$

- Can easily compute one from the other
- Example at blackboard

0.0.15 Syndrome-based error detection

Syndrome-based error *detection* for linear block codes:

1. generate codewords with generator matrix:
$$\mathbf{i} \cdot [G] = \mathbf{c}$$
2. send codeword $\mathbf{c}$ on the channel
3. random error word $\mathbf{e}$ is applied on the channel
4. receive word $\mathbf{r} = \mathbf{c} \oplus \mathbf{e}$
5. compute **syndrome** of $\mathbf{r}$:
$$\mathbf{z} = [H] \cdot \mathbf{r}^T$$
6. Decide:
   - If $\mathbf{z} = 0$ => $\mathbf{r}$ has no errors
   - If $\mathbf{z} \neq 0$ => $\mathbf{r}$ has errors

0.0.16 Syndrome-based error correction

Syndrome-based error *correction* for linear block codes:

- $\mathbf{z} \neq 0$ => $\mathbf{r}$ has errors, we need to locate them
- The syndrome is the effect only of the error word:
$$\mathbf{z} = [H] \cdot \mathbf{r}^T = [H] \cdot (\mathbf{c}^T \oplus \mathbf{e}^T) = [H] \cdot \mathbf{e}^T$$

7. Create a **syndrome lookup table**:
   - for every possible error word $\mathbf{e}$, compute the syndrome $\mathbf{z} = [H] \cdot \mathbf{e}^T$
   - start with error words with 1 error (most likely), then with 2 errors (less likely), and so on
8. Locate the syndrome $\mathbf{z}$ in the table, read the corresponding error word $\hat{\mathbf{e}}$
9. Find the correct word:
   - adding the error word again will invert the errored bits back to the originals
$$\hat{\mathbf{c}} = \mathbf{r} \oplus \hat{\mathbf{e}}$$

0.0.17 Summing up

Summing up until now:

- Linear block codes use a generator matrix G for encoding, and a parity-check matrix H for checking the received word.

0.0.18 Hamming codes

- A particular class of linear error-correcting codes
- Definition: a **Hamming code** is a linear block code where the columns of $[H]$ are *the binary representation of all numbers from 1 to* $2^r - 1, \forall r \geq 2$
- Example (blackboard): (7,4) Hamming code
- Systematic: arrange the bits in the codeword, such that the control bits correspond to the columns having a single 1
  - no big difference from the usual systematic case, just a rearrangement of bits
  - makes implementation easier
- Example codeword for Hamming(7,4):

$$c_1 c_2 i_3 c_4 i_5 i_6 i_7$$

0.0.19 Properties of Hamming codes

- From definition of $[H]$ it follows:
  1. Codeword has length $n = 2^r - 1$
  2. r bits are parity bits (also known as ***control bits***)
  3. $k = 2^r - r - 1$ bits are information bits
- Notation: **(n,k) Hamming code**
  - n = codeword length = $2^r - 1$,
  - k = number of information bits = $2^r - r - 1$
  - Example: (7,4) Hamming code, (15,11) Hamming code, $(127, 120)$ Hamming code

0.0.20 Properties of Hamming codes

- Can detect two errors
  - All columns are different => can detect 2 errors
  - Sum of two columns equal to a third => cannot correct 3

OR

- Can correct one error
  - All columns are different => can correct 1 error
  - Sum of two columns equal to a third => cannot correct 2
  - Non-systematic: syndrome = error position

BUT * Not simultaneously! * same non-zero syndrome can be obtained with 1 or 2 errors, can't distinguish

0.0.21 Coding rate of Hamming codes

Coding rate of a Hamming code:

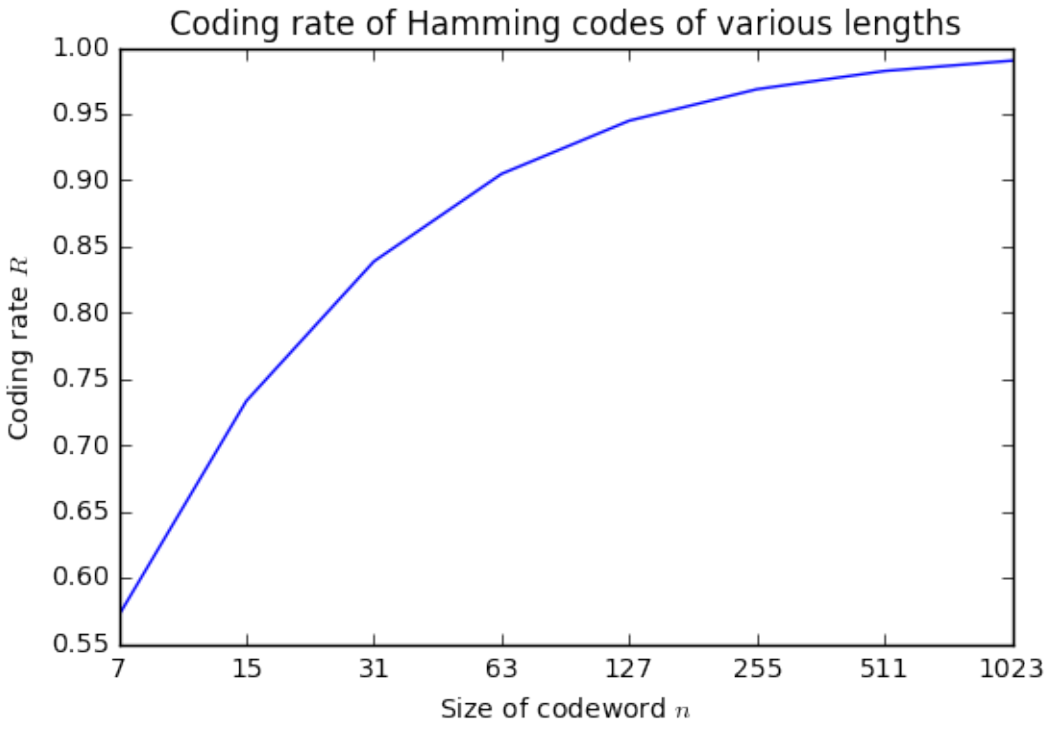
$$R = \frac{k}{n} = \frac{2^r - r - 1}{2^r - 1}$$

The Hamming codes can correct 1 OR detect 2 errors in a codeword of size n * (7,4) Hamming code: $n = 7$ * (15,11) Hamming code: $n = 15$ * (31,26) Hamming code: $n = 31$

Longer Hamming codes are progressively weaker: * weaker error correction capability * better efficiency (higher coding rate) * more appropriate for smaller error probabilities

```
In [2]: %matplotlib inline
        import numpy as np, matplotlib.pyplot as plt
        r = np.array([3., 4., 5., 6., 7., 8., 9., 10.])
        k = 2**r - r - 1
        n = 2**r-1
        R = k/n
        plt.plot(r, R)
        plt.xticks(r, [str(int(i)) for i in n])
        plt.title('Coding rate of Hamming codes of various lengths')
        plt.xlabel('Size of codeword $n$')
        plt.ylabel('Coding rate $R$')
```

```
Out[2]: <matplotlib.text.Text at 0xab8e646c>
```



0.0.22 Encoding & decoding example for Hamming(7,4)

See whiteboard.

In this example, encoding is done without the generator matrix G, directly with the matrix H, by finding the values of the parity bits c_1, c_2, c_4 such that

$$\begin{bmatrix}0\\0\\0\end{bmatrix} = [H] \begin{bmatrix}c_1\\c_2\\i_3\\c_4\\i_5\\i_6\\i_7\end{bmatrix}$$

For a single error, the syndrome **is the binary representation of the location of the error**.

0.0.23 Circuits for encoding and decoding Hamming(7,4)

At whiteboard, using **shift registers**.

0.0.24 SECDED Hamming codes

Hamming codes can correct 1 error OR can detect 2 errors, but we cannot differentiate the two cases:

Example: * the syndrome $\mathbf{z} = [H] \cdot \mathbf{r}^T = \begin{bmatrix}0\\1\\1\end{bmatrix}$ can be caused by: * a single error in location 3 (bit i_3) * two errors in location 1 and 2 (bits c_1, bits c_2)

- if we know it is a single error, we can go ahead and correct it, then use the corrected data
- if we know there are two errors, we should NOT attempt to correct them, because we cannot locate the errors correctly

Unfortunately, it is **not possible to differentiate** between the two cases.
Solution? Add additional parity bit $\rightarrow$ SECDED Hamming codes

0.0.25 SECDED Hamming codes

- Add an additional parity bit to differentiate the two cases
  - c_0 = sum of all n bits of the codeword
- For (7,4) Hamming codes:

$$\mathbf{c_0} c_1 c_2 i_3 c_4 i_5 i_6 i_7$$

- The pairty check matrix is extended by 1 row and 1 column

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

- Known as SECDED Hamming codes
  - **S**ingle **E**rror **C**orrection - **D**ouble **E**rror **D**etection

0.0.26 Encoding and decoding of SECDED Hamming codes

Encoding

- compute codeword using $\tilde{H}$

Decoding

- Compute syndrome of the received word using $\tilde{H}$

$$\tilde{\mathbf{z}} = \begin{bmatrix} z_0 \\ \mathbf{z} \end{bmatrix} = [\tilde{H}] \cdot \mathbf{r}^T$$

- z_0 is an additional bit in the syndrome corresponding to c_0
- z_0 tells us whether the received c_0 matches the paroty of the received word
  - $z_0 = 0$: the additional parity bit matches the parity of the received word
  - $z_0 = 1$: the additional parity bit does not match the parity of the received word

Decide which of the following cases happened

- If no error happened: $z_1 = z_2 = z_3 = 0, z_0 = \forall$
- If 1 error happened: syndrome is non-zero, $z_0 = 1$ (does not match)
- If 2 errors happened: syndrome is non-zero, $z_0 = 0$ (does match, because the two errors cancel each other out)
- If 3 errors happened: same as 1, can't differentiate
- Now can simultaneously differentiate between:
  - 1 error: $\rightarrow$ perform correction
  - 2 errors: $\rightarrow$ detect, but do not perform correction
- Also, if correction is never attempted, can detect up to 3 errors
  - minimum Hamming distance = 4 (no proof given)
  - don't know if 1 error, 2 errors or 3 errors, so can't try correction

0.0.27 Circuits for encoding and decoding SECDED Hamming codes

At whiteboard. Slight modification of the previous schematic to accommodate the extra parity bit.

0.0.28 Summary until now

- Systematic codes: information bits + parity bits
- Generator matrix: use to generate codeword

$$\mathbf{i} \cdot [G] = \mathbf{c}$$

- Parity-check matrix: use to check if a codeword

$$0 = [H] \cdot \mathbf{c}^T$$

- Syndrome:
$$\mathbf{z} = [H] \cdot \mathbf{r}^T$$
- Syndrome-based error detection: syndrome non-zero
- Syndrome-based error correction: lookup table
- Hamming codes: $[H]$ contains all numbers $1...2^r - 1$
- SECDED Hamming codes: add an extra parity bit

0.1 Cyclic codes

Definition:

Cyclic codes are a particular class of linear block codes for which *every cyclic shift of a codeword is also a codeword*

- Cyclic shift: cyclic rotation of a sequence of bits (any direction)
- Are a particular class of linear block codes, so all the theory up to now still applies
  - they have a generator matrix, parity check matrix etc.
- But they can be implemented more efficient than general linear block codes (e.g. Hamming)
- Used **everywhere** under the common name **CRC** (**C**yclic **R**edundancy **C**heck)
  - Network communications (Ethernet), data storage in Flash memory

0.1.1 Binary polynomials

Every binary sequence $\mathbf{a}$ corresponds to a polynomial $\mathbf{a}(\mathbf{x})$ with binary coefficients

$$a_0 a_1 ... a_{n-1} \rightarrow \mathbf{a}(\mathbf{x}) = a_0 \oplus a_1 x \oplus ... \oplus a_{n-1} x^{n-1}$$

Example:

$$10010111 \rightarrow 1 \oplus x^3 \oplus x^5 \oplus x^6 \oplus x^7$$

From now on, by "codeword" we also mean the corresponding polynomial.

Can perform all mathematical operations with these polynomials: * addition, multiplication, division etc. (examples)

There are efficient circuits for performing multiplications and divisions.

0.1.2 Generator polynomial

Theorem:

All the codewords of a cyclic code are multiples of a certain polynomial $g(x)$, known as **generator polynomial**.

Properties of generator polynomial $g(x)$:

- The generator polynomial has first and last coefficient equal to 1.
- The generator polynomial is a factor of $X^n \oplus 1$
- The *degree* of $g(x)$ is $n - k$, where:
  - The codeword = polynomial of degree $n - 1$ (n coefficients)

- The information polynomial = polynomial of degree $k-1$ (k coefficients)

$$(k-1)+(n-k)=n-1$$

- **The degree of $g(x)$ is the number of parity bits of the code.**

0.1.3 Finding a generator polynomial

Theorem:

If $g(x)$ is a polynomial of degree $(n-k)$ and is a factor of $X^n \oplus 1$, then $g(x)$ generates a (n,k) cyclic code.

Example:

$$1 \oplus x^7 = (1 \oplus x)(1 \oplus x + \oplus x^3)(1 \oplus x^2 \oplus x^3)$$

Each factor generates a code:

- $1 \oplus x$ generates a (7,6) cyclic code
- $1 \oplus x \oplus x^3$ generates a (7,4) cyclic code
- $1 \oplus x^2 \oplus x^3$ generates a (7,4) cyclic code

0.1.4 Computing the codewords

Start from **information polynomial** with k bits

$$i(x) = i_0 \oplus i_1 x \oplus ... \oplus i_{k-1}x^{k-1}$$

Non-systematic codeword generation:

- Codeword = $i(x) \cdot g(x)$

$$\boxed{c(x) = i(x) \cdot g(x)}$$

Systematic codeword generation:

$$\boxed{c(x) = b(x) \oplus x^{n-k} i(x)}$$

where $b(x)$ is the remainder of dividing $x^{n-k}i(x)$ to $g(x)$:

$$x^{n-k}i(x) = a(x)g(x) \oplus b(x)$$

- (Proof: at blackboard)

0.1.5 Proving the cyclic property

We prove that any cyclic shift of a codeword is also a codeword.

Proof: at whiteboard

- Original codeword

$$c_0 c_1 c_2 ... c_{n-1} \rightarrow \mathbf{c(x)} = c_0 \oplus c_1 x \oplus ... \oplus c_{n-1}x^{n-1}$$

- Cyclic shift to the right by 1 position

$$c_{n-1}c_0c_1...c_{n-2} \rightarrow \mathbf{c}'(\mathbf{x}) = c_{n-1} \oplus c_0 x \oplus ... \oplus c_{n-2}x^{n-1}$$

- Note that

$$\begin{aligned}\mathbf{c}'(\mathbf{x}) =& x \cdot \mathbf{c}(\mathbf{x}) \oplus c_{n-1}x^n \oplus c_{n-1} \\ =& x \cdot \mathbf{c}(\mathbf{x}) \oplus c_{n-1}(x^n \oplus 1)\end{aligned}$$

Since $\mathbf{c}(\mathbf{x})$ is a multiple of $g(x)$, so is $x \cdot \mathbf{c}(\mathbf{x})$. Also $(x^n \oplus 1)$ is always a multiple of $g(x)$. It follows that their sum $\mathbf{c}'(\mathbf{x})$ is a also a multiple of $g(x)$, which means it is a codeword.

0.1.6 Cyclic code encoder circuits

- Coding = based on polynomial multiplications and divisions
- Efficient circuits for multiplication / division exist, that can be used for systematic or non-systematic codeword generation (draw on blackboard)

0.1.7 Circuits for multiplication of binary polynomials

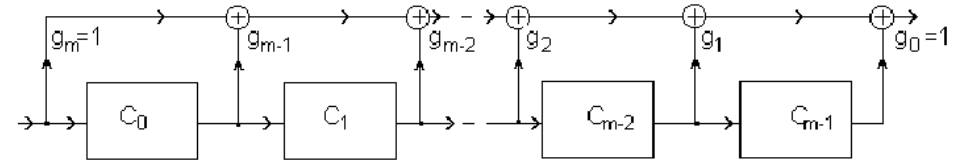


Figura 2

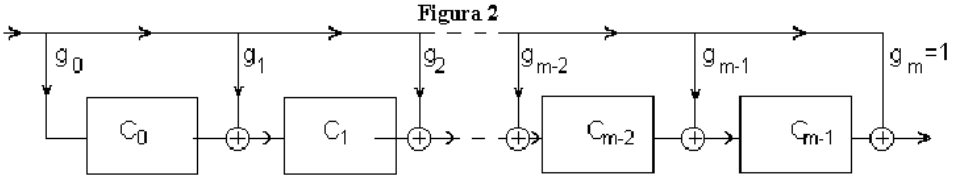


Figura 3

Circuits for polynomial multiplication

0.1.8 Operation of multiplication circuits

- The input polynomial is applied at the input, 1 bit at a time, starting from highest degree
- The output polynomial is obtained at the output, 1 bit at a time, starting from highest degree
- Because output polynomial has larger degree, the circuit needs to operate a few more samples until the final result is obtained. During this time the input is 0.
- Examples: at the whiteboard

0.1.9 Linear analysis of multiplication circuits

- These circuits are **linear time-invariant systems** (remember Digital Signal Processing class?), because they are composed only of summations, multiplication by scalars, and delay blocks.
- Therefore, using the Z transform approach (to come soon in Digital Signal Processing class), the output can be computed based on the graph of the system:

- Draw the graph of the system: cells become z^{-1} blocks, everything else is the same
- Every z^{-1} block means a delay of one, which is what a cell does
- Call the input polynomial is $\mathbf{X(z)}$
- Call the output polynomial is $\mathbf{Y(z)}$
- Every z^{-1} block means multiplying with z^{-1}
- Compute the output $\mathbf{Y(z)}$ based on $X(z)$, from the graph

We get:

$$Y(z) = X(z) \cdot G(z) \cdot z^{-m},$$

meaning that the **output polynomial = input polynomial * g(x) polynomial, with a delay of** m **bits (time samples)**.

The delay of m time samples is caused by the fact that the input polynomial has degree $(k-1)$, but the resulting polynomial has larger degree $(k-1)+m$, therefore we need to wait m more time samples until we get the full result.

0.1.10 Circuits for division binary polynomials

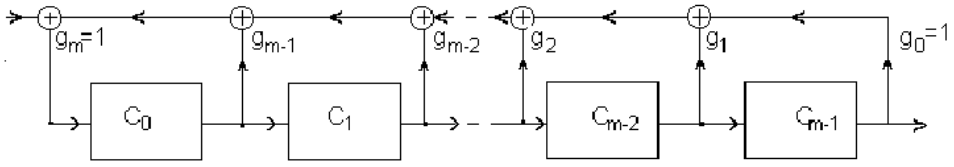


Figura 4

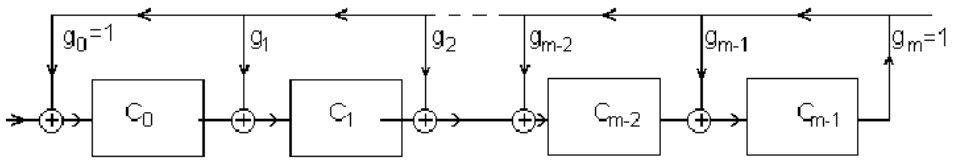


Figura 5

Circuits for polynomial division

0.1.11 Operation of division circuits

- The input polynomial is applied at the input, 1 bit at a time, starting from highest degree
- The output polynomial is obtained at the output, 1 bit at a time, starting from highest degree
- Because output polynomial has smaller degree, the circuit first outputs some zero values, until starting to output the result.
- Examples: at the whiteboard

0.1.12 Linear analysis of division circuits

- These circuits are also **linear time-invariant systems**, because they are composed only of summations, multiplication by scalars, and delay blocks.
- Therefore, using the Z transform approach, the output can be computed based on the graph of the system:

- Draw the graph of the system: cells become z^{-1} blocks, everything else is the same
- Every z^{-1} block means a delay of one, which is what a cell does
- Call the input polynomial is $\mathbf{X}(\mathbf{z})$
- Call the output polynomial is $\mathbf{Y}(\mathbf{z})$
- Every z^{-1} block means multiplying with z^{-1}
- Compute the output $\mathbf{Y}(\mathbf{z})$ based on $X(z)$, from the graph

We get:

$$Y(z) = \frac{X(z)}{G(z)}$$

meaning that the **output polynomial = input polynomial / g(x) polynomial**.

0.1.13 Linear-Feedback Shift Registers (LFSR)

0.1.14 Error detection with cyclic codes

- Like usual for linear codes: check if received word is codeword or not
- Every codeword is multiple of $g(x)$
- Check if received word is actually dividing with $g(x)$
  - Use a circuit for division of polynomials
- If remainder is 0 => it is a codeword, no error
- If remainder is non-0 => error detected!
- Cyclic codes have very good error detection capabilities

0.1.15 Error correction capability

Theorem:
Any (n,k) cyclic codes is capable of detecting any error **burst** of length $n - k$ or less.

- A large fraction of longer bursts can also be detected (but not all)
- For non-burst errors (random): more difficult to analyze

0.1.16 Error correction with cyclic codes

- Like usual for linear codes: lookup table based on remainder
- Remainder of division = the effect of the error polynomial
- Create lookup table: for every error word, compute remainder
- Search the table for the remainder of the received word => find error word

0.1.17 Summary of cyclic codes

- Generated using a generator polynomial $g(x)$
- Non-systematic:
$$c(x) = i(x) \cdot g(x)$$
- Systematic:
$$c(x) = b(x) \oplus X^{n-k} i(x)$$
  - $b(x)$ is the remainder of dividing $X^{n-k} i(x)$ to $g(x)$
- Syndrome = remainder of division $r(x)$ to $g(x)$
- Error detection: remainder (syndrome) non-zero
- Error correction: lookup table