KUNGL TEKNISKA HÖGSKOLAN

International Master Program in System-on-Chip Design

Information redundancy

Information redundancy

- add information to date to tolerate faults
  - error detecting codes
  - error correcting codes
- data applications
  - communication
  - memory

Code

- Code of length n is a set of n-tuples satisfying some well-defined set of rules
- binary code uses only 0 and 1 symbols
  - binary coded decimal (BCD) code
    - uses 4 bits for each decimal digit

0000	0
0001	1
0010	2
...	
1001	9

Code word

- Codeword is an element of the code satisfying the rules of the code
- Word is an n-tuple not satisfying the rules of the code
- Codewords should be a subset of all possible 2^n binary tuples to make error detection/correction possible
  - BCD: 0110 valid; 1110 invalid
  - any binary code: 2013 invalid
- The number of codewords in a code C is called the size of C

Encoding/decoding

- encoding
  - transform data into code word

data → **encoding** → code word

- decoding
  - recover data from code word

code word → **decoding** → data

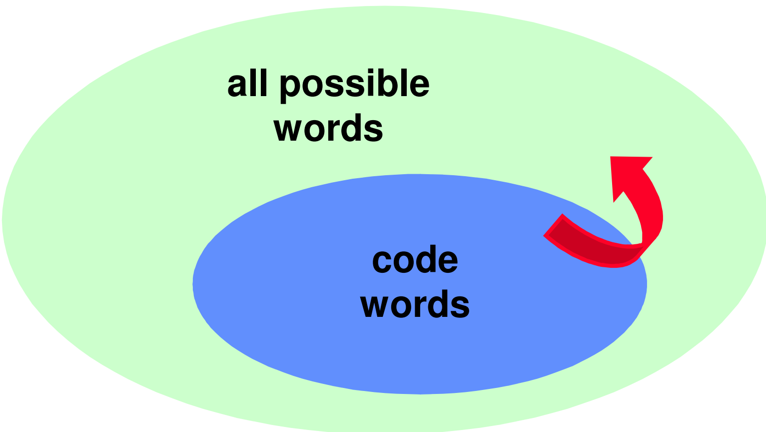
Encoding/decoding

- 2 scenario if errors affect codeword:
  - correct codeword → another codeword
  - correct codeword → word

Error detection

- We can define a code so that errors introduced in a codeword force it to lie outside the range of codewords
  - basic principle of error detection

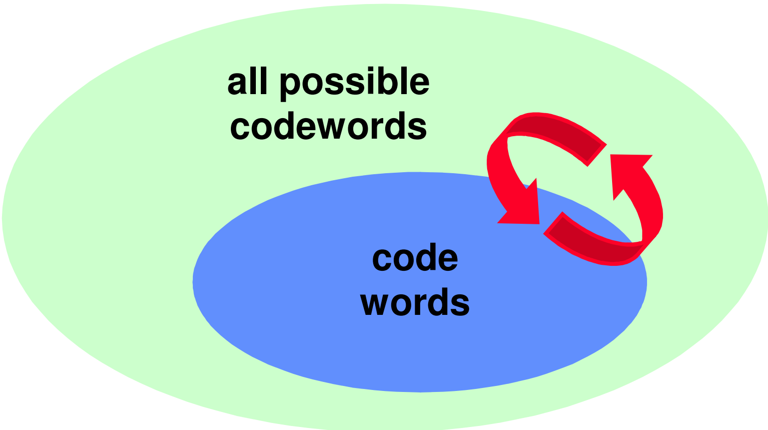
Error detection



Error correction

- We can define a code so that it is possible to determine the correct code word from the erroneous codeword
  - basic principle of error correction

Error correction



Error detecting/correcting code

- Characterized by the number of bits that can be corrected
  - double-bit detecting code can detect two single-bit errors
  - single-bit correcting code can correct one single-bit error
- Hamming distance gives a measure of error detecting/correcting capabilities of a code

Hamming distance

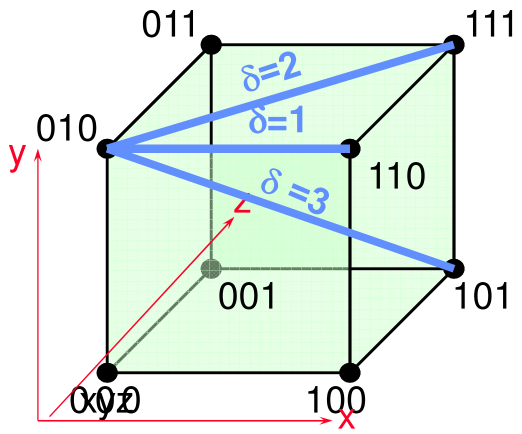
Hamming distance is the number of bit positions in which two n-tuples differ

x 0000

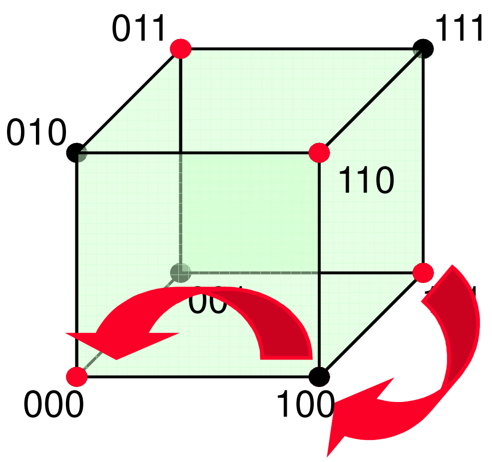
y 0101

$\delta(x,y) = 2$

3-dimensional space (3-bit words)

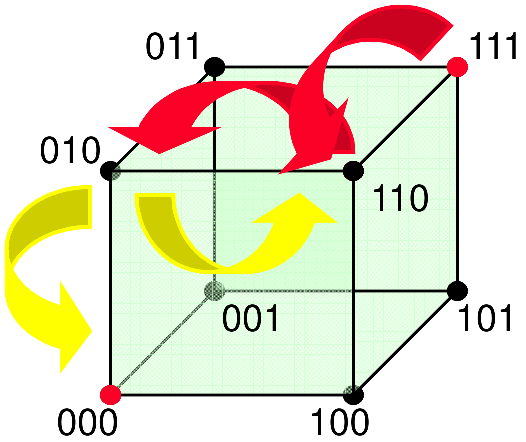


Error detection



If codewords are on distance ≥ 2, we can detect single-bit errors

Error correction



If codewords are on distance ≥ 3, we can correct single-bit errors

Code distance

code distance is the minimum Hamming distance between any two distinct codewords

C_d = 2 code detects all single-bit errors

code: 00, 11

invalid code words: 01 or 10

C_d = 3 code corrects all single-bit errors

code: 000, 111

invalid code words: 001, 010, 100, 101, 011, 110

Relation b/w code distance and capabilities of the code

A code can correct up to c bit errors and
detect up to d additional bit errors if and only if:

$$2c + d + 1 \leq C_d$$

Separable/non-separable code

- separable code
  - codeword = data + check bits
  - e.g. parity: 11011 = 1101 + 1
- non-separable code
  - codeword = data mixed with check bits
  - e.g. cyclic: 1010001 -> 1101
- decoding process is much easier for separable codes (remove check bits)

Information rate

- The ratio k/n, where
  - k is the number of data bits
  - n is the number of data + check bits

is called the information rate of the code

- Example: a code obtained by repeating data three times has the information rate 1/3

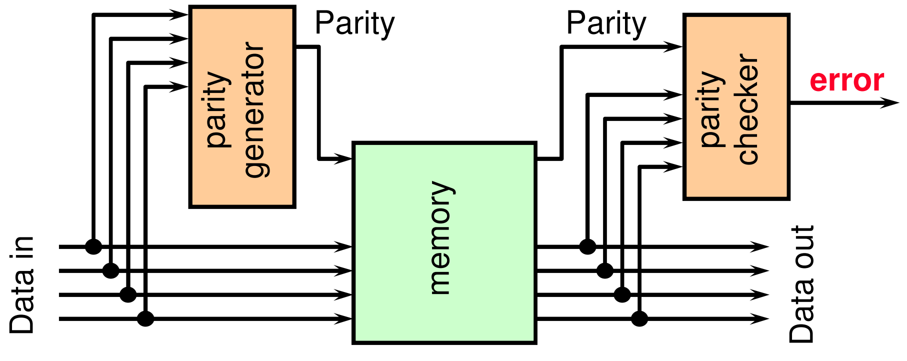
Next: Types of codes

- parity codes
- linear codes
  - Hamming codes
- cyclic codes
  - CRC codes
  - Reed-Solomon codes
- unordered codes
  - m-of-n codes
  - Berger codes
- arithmetic codes
  - AN-codes
  - residue codes

Single-bit parity code

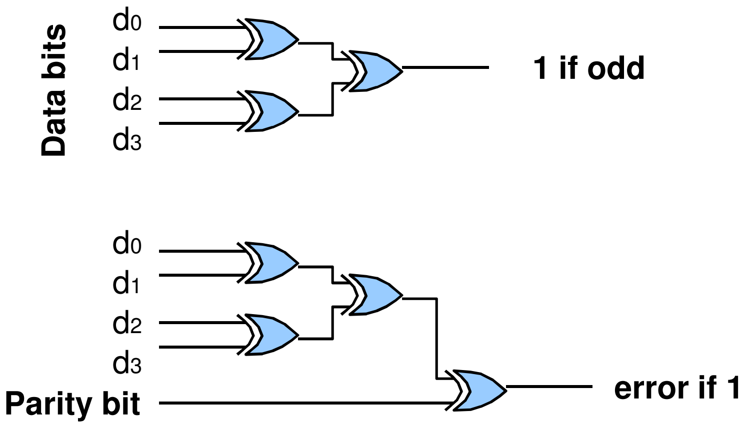
- Add an extra bit to binary word so that that resulting code word has either even or odd number of 1s
  - even parity: even # ‘1’
  - odd parity: odd # ‘1’
- single bit error detection: $C_d = 2$
- separable code
- use: bus, memory, transmission, …

Organization of memory with single-bit parity code



extra HW required (parity generator, checker, extra memory)

Parity generation and checking



Problem with single-bit parity code

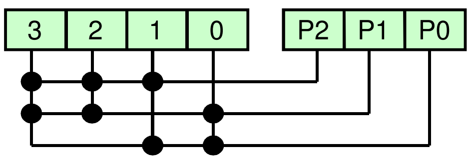
- Multiple-bit errors (even number of bits) cannot be detected
  - some of them are often very common
    - failure of the individual memory chip

Other parity codes

- The purpose is to provide additional error capability
  - bit-per-word
  - bit-per-byte
  - bit-per-multiple-chips
  - bit-per-chip
  - interlaced
  - overlapping

Overlapping parity code (Hamming code)

Overlapping parity for 4-bits of data - each data bit is assigned to multiple parity groups



Bit in error	Parity pattern
3	P2 P1 P0
2	P2 P1
1	P2 P0
0	P1 P0
P2	P2
P1	P1
P0	P0
no error	

Overlapping parity code (Hamming code)

- k data bits, c parity bits
- to have unique parity pattern per error:

$$2^c \geq k+c+1$$

k	c	redundancy
2	3	150%
4	3	75%
8	4	50%
16	5	31%
32	6	19%
64	7	11%

Background

- A field Z_2 is the set { 0,1} together with two operations of addition and multiplication (modulo 2) satisfying a given set of properties
- A vector space V_n over a field Z_2 is a subset of Z_2^n, with two operations of addition and multiplication (modulo 2) satisfying a given set of properties
- A subspace is a subset of a vector space which is itself a vector space
- A set of vectors $\{v_0,\ldots,v_{k-1}\}$ is linearly independent if $a_0v_0 + a_1v_1 + \ldots + a_{k-1}v_{k-1} = 0$ implies $a_0 = a_1 = \ldots = a_{k-1} = 0$

Linear code: Definition

- A (n,k) linear code over the field Z_2 is a k-dimensional subspace of V_n
  - spanned by k linearly independent vectors
  - any codeword c can be written as a linear combination of k basic vectors $(v_0,\ldots,v_{k-1})$ as follows

$$c = d_0 v_0 + d_1 v_1 + \ldots + d_{k-1} v_{k-1}$$

  - $(d_0,d_1,\ldots,d_{k-1})$ is the data to be encoded

Example: (4,2) linear code

- Data to be encoded are 2-bit words

[00], [01], [10], [11]

- Suppose we select for a basis the vectors

$$v_0 = [1000],\ v_1 = [0110]$$

- To find the codeword $c = [c_0 c_1 c_2 c_3]$ corresponding to the data $d = [d_0 d_2]$, we compute the linear combination of the basic vectors as

$$c = d_0 v_0 + d_1 v_1$$

- For example, data d = [11] is encoded to

$$c = 1 \cdot [1000] + 1 \cdot [0110] = [1110]$$

Example (cont.)

- d = [00] is encoded to

  $c = 0 \cdot [1000] + 0 \cdot [0110] = [0000]$
- d = [01] is encoded to

  $c = 0 \cdot [1000] + 1 \cdot [0110] = [0110]$
- d = [10] is encoded to

  $c = 1 \cdot [1000] + 0 \cdot [0110] = [1000]$

Generator matrix

- The rows of the generator matrix are the basis vectors $v_0,\ldots,v_{k-1}$
- For example, generator matrix for the previous example is

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

- Codeword c is obtained by multiplying G by d

$$c = d \cdot G$$

Example: (6,3) linear code

- Construct the code spanned by the basic vectors [100011], [010110] and [001101]
- The generator matrix for this code is

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

- For example, data d = [011] is encoded to

$$c=0\cdot[100011]+1\cdot[010110]+1\cdot[001101]=[011011]$$

Parity check matrix

- To check for errors in a (n,k) linear code, we use an $(n-k)\times n$ parity check matrix H of the code
- The parity check matrix is related to the generator matrix by the equation

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

where G^T denotes the transponse of G

- This implies that, for any codeword c, the product of the parity check matrix and the encoded message should be zero

$$H \cdot c^T = 0$$

Constructing parity check matrix

- If a generator matrix is of the form $G = [I_k \ A]$, then the parity check matrix is of the form

$$H = [A^T \ I_{n-k}]$$

Example: (6,3) linear code

- If G is of the form

$$G = \begin{pmatrix} 1 & 0 & 0 & | & 0 & 1 & 1 \\ 0 & 1 & 0 & | & 1 & 1 & 0 \\ 0 & 0 & 1 & | & 1 & 0 & 1 \end{pmatrix}$$

- Then H is of the form

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

Syndrome

- Encoded data is checked for errors by multiplying it by the parity check matrix

$$s = H \cdot c^T$$

- The resulting (n-k)-bit vector is called syndrome
  - If s = 0, no error has occurred
  - If s matches one of the columns of H, a single-bit error has occurred. The bit position corresponds to the position of the matching column of H
  - Otherwise, a multiple-bit error has occurred

Constructing linear codes

- To be able to correct e errors, a code should have a distance of at least 2e+1
- It is possible to ensure the code distance c by selecting the parity check matrix with c-1 linearly independent columns
  - To have a code with code distance 2 (single-error detecting), every column of H should be linearly independent, I.e. H shouldn't have a zero column

Example: (4,2) linear code

- Parity check matrix for (4,2) linear code we have constructed before is

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

- The first column is zero, therefore the columns of H are linearly dependent and code distance is 1
- Let us construct a code with distance 2

Example: (4,2) linear code, C_d=2

- Replace 1st column by a column containing all 1

$$H = \left(\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{array}\right) = [A^T \; I_2]$$

- Now G can be constructed as

$$G = [I_2 \, A] = \left(\begin{array}{cc|cc} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right)$$

Example: (4,2) linear code, C_d=2

- The resulting code generated by G is

data		codeword			
d_1	d_2	c_1	c_2	c_3	c_4
0	0	0	0	0	0
0	1	0	1	1	0
1	0	1	0	1	1
1	1	1	1	0	1

Hamming codes

- Hamming codes are a family of linear codes
- An (n,k) Hamming code satisfies the property that the columns of its parity check matrix represent all possible non-zero vectors of length n-k
- Example: (7,4) Hamming code

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

Parity check matrix

- If the columns of H are permuted, the resulting code remains a Hamming code
- Example: different (7,4) Hamming code

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

- Such H is called lexicographic parity check matrix
  - the corresponding code does not have a generator matrix in standard form $G = [I_3 \, A]$

Decoding

- If the parity check matrix H is lexicographic, a simple procedure for syndrome decoding exists
- To check a codeword c for errors, calculate

$$s = H \cdot c^T$$

  - If s = 0, no error has occurred
  - If $s \neq 0$, then it matches one of the columns of H, say i
  - c is decoded assuming that a single-bit error has occurred in the i^{th} bit of c

Example: (7,4) Hamming code

- Construct Hamming code corresponding to parity check matrix

$$H = \begin{pmatrix} 1\ 1\ 0\ 1 & 1\ 0\ 0 \\ 1\ 0\ 1\ 1 & 0\ 1\ 0 \\ 1\ 1\ 1\ 0 & 0\ 0\ 1 \end{pmatrix} = [A^T\ I_3]$$

- The corresponding G is

$$G = [I_4\ A] = \begin{pmatrix} 1\ 0\ 0\ 0 & 1\ 1\ 1 \\ 0\ 1\ 0\ 0 & 1\ 0\ 1 \\ 0\ 0\ 1\ 0 & 0\ 1\ 1 \\ 0\ 0\ 0\ 1 & 1\ 1\ 0 \end{pmatrix}$$

Example (cont.)

- Suppose the data to be encoded is d = [1110]
- We multiply d by G to get c = [1110001]
- Suppose an error has occurred in the last bit of c
  - c is transformed to [1110000]
- By multiplying [1110000] by H, we s = [001]
- s matches the last column of H
  - the error has occurred in the last bit of the codeword
- We correct [1110000] to [1110001] and decode it to d = [1110] by taking the first 4 bits of data

Example: (7,4) Hamming code, lexicographic

- Generator matrix corresponding to the lexicographic parity check matrix is

$$G = \begin{pmatrix} 0\,1\,0\,1\,0\,1\,0 \\ 0\,0\,1\,1\,0\,0\,1 \\ 0\,0\,0\,1\,1\,1\,1 \\ 1\,0\,0\,0\,0\,1\,1 \end{pmatrix}$$

- So, data d = $[d_0 d_1 d_2 d_3]$ is encoded as c = $[d_3 d_0 d_1 p_1 d_2 p_2 p_3]$ where p_1, p_2, p_3 are parity check bit defined by

$$p_1 = d_0 + d_1 + d_2$$
$$p_2 = d_0 + d_2 + d_3$$
$$p_3 = d_1 + d_2 + d_3$$

Error correction

- If parity check matrix is lexicographic, the error correction can be implemented using a decoder and XOR gates
- The first level of XOR gates compares stored check bits with re-computed ones
- The result of the comparison in the syndrome $[s_0 s_1 s_2]$, which is fed into the decoder
- For the syndrome s = i, $i \in \{0,1,\ldots,7\}$, i^{th} output of the decoder is high
- The second level of XOR gates complements the i^{th} bit of the word, thus correcting the error

Code distance of Hamming codes

- The code distance of a Hamming code is 3, so it can correct single-bit error
- Often extended Hamming code is used, which can correct single-bit error and detect double-bit errors
  - obtained by adding a parity check bit to every codeword of a Hamming code
  - if $c = [c_1c_2\ldots c_n]$ is a codeword of a Hamming code, $c' = [c_0c_1c_2\ldots c_n]$ is the corresponding extended codeword, where c_0 is the parity bit

Extended Hamming code

- The parity check matrix of an extended (n,k) Hamming code is obtained as follows
  - add a zero column in front of a lexicographic parity check matrix of an (n,k) Hamming code
  - attach a row consisting of all 1's as n-k+1th row of the resulting matrix
- Example: Extended (1,1) Hamming code

$$H = \left(\begin{array}{c|c} 0 & 1 \\ \hline 1 & 1 \end{array}\right)$$

Cyclic codes

- Cyclic codes are special class of linear codes
- Used in applications where burst errors can occur
  - a group of adjacent bits is affected
  - digital communication, storage devices (disks, CDs)
- Important classes of cyclic codes:
  - Cyclic redundancy check (CRC)
    - used in modems and network protocols
  - Reed-Solomon code
    - used in CD and DVD players

Cyclic code: Definition

- A linear code is called cyclic if any end-around shift of codeword produces another codeword
  - if $[c_0 c_1 c_2 \ldots c_{n-2} c_{n-1}]$ is a codeword, then $[c_{n-1} c_0 c_1 c_2 \ldots c_{n-2}]$, is a codeword, too
- it is convenient to think of words as polynomials rather than vectors
  - for example, a codeword $[c_0 c_1 \ldots c_{n-1}]$ is represented as a polynomial

$$c_0 \cdot x^0 + c_1 \cdot x^1 + \ldots + c_{n-1} \cdot x^{n-1}$$

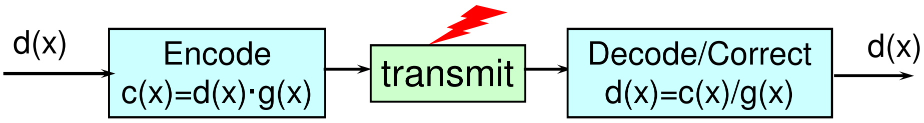
Polynomials

- Since the code is binary, the coefficients are 0 and 1
- For example, $d(x) = 1 \cdot x^0 + 0 \cdot x^1 + 1 \cdot x^2 + 1 \cdot x^3$ represents the data (1011)
- We always write least significant digit on the left

Polynomials

- The degree of a polynomial equals to its highest exponent
  - e.g. the degree of $1 + x^1 + x^3$ is 3
- a cyclic code with the generator polynomial of degree (n-k) detects all burst errors affecting (n-k) bits or less
  - n is the number of bits in codeword
  - k is the number of bits in data word

Encoding/decoding



Encoding

Multiply data polynomial by generator polynomial:

$$c(x) = d(x).g(x)$$

Calculations are performed in Galois Field GF(2):

- multiplication modulo 2 = AND operation
- addition modulo 2 = XOR operation
- in GF(2), subtraction = addition

Properties of generator polynomial

- g(x) is the generator polynomial for a linear cyclic code of length n if and only if g(x) divides $1+x^n$ without a reminder

Example of polymomial multiplication (1)

$d(x) = (1011) = x^3 + x^2 + 1$

$g(x) = x^3 + x + 1$

k = 4

$c(x) = d(x).g(x)$

$= (x^3 + x^2 + 1).(x^3 + x + 1)$

$= x^6 + x^4 + x^3 + x^5 + x^3 + x^2 + x^3 + x + 1$

$= x^6 + x^5 + x^4 + x^3 + x^2 + x^1 + 1$

$= (1111111)$

n = 7

Example of polynomial multiplication (2)

$d(x) = (1010) = x^2 + 1$

$g(x) = x^3 + x + 1$

k = 4

$c(x) = d(x).g(x)$

$= (x^3 + x + 1).(x^2 + 1)$

$= x^5 + x^3 + x^3 + x + x^2 + 1$

$= x^5 + x^2 + x + 1$

$= (1110010)$

n = 7

Example: (7,4) cyclic code

- Find a generator polynomial for a code of length n=7 for encoding of data of length k=4
- g(x) should be of a degree 7-4=3 and should divide $1+x^7$ without a reminder
- $1+x^7$ can be factored as

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

- so, we can choose for g(x) either $1+x+x^3$ or $1+x^2+x^3$

Parity check polynomial

- For a cyclic code with the generator polynomial g(x), the check polynomial h(x) is determined by

$$g(x) \cdot h(x) = 1+x^n$$

- Since codewords are multiples of g(x), for every codeword c(x), it is hold that

$$c(x) \cdot h(x) = 0 \bmod 1+x^n$$

Decoding

Divide received polynomial c(x) by the generator polynomial g(x):

$$d(x) = c(x)/g(x)$$

- Division is the polynomial division in GF(2)
- The reminder from the division is syndrome s(x)
  - if s(x) is zero, no error has occurred

Example of polynomial division (1)

$$
\begin{array}{l|l}
x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 & x^3 + x + 1 \\
\underline{x^6 + x^4 + x^3} & \overline{x^3 + x^2 + 1} \\
\quad x^5 + x^2 + x + 1 & \\
\quad \underline{x^5 + x^3 + x^2} & \\
\qquad x^3 + x + 1 & \\
\qquad \underline{x^3 + x + 1} & \\
\qquad\qquad 0 &
\end{array}
$$

Example of polynomial division (2)

$$
\begin{array}{l|l}
x^8 + x^5 + x^4 + x^2 + 1 & x^3 + x + 1 \\
\underline{x^8 + x^6 + x^5} & \overline{x^5 + x^3 + 1} \\
\quad x^6 + x^4 + x^2 + 1 & \\
\quad \underline{x^6 + x^4 + x^3} & \\
\qquad x^3 + x^2 + 1 & \\
\qquad \underline{x^3 + x + 1} & \\
\qquad\qquad x^2 + x &
\end{array}
$$

Decoding (no error)

- if no error occurred, then the received codeword is the correct codeword c(x)
- therefore, d(x) = c(x)/g(x)

Decoding in presence of error

- Suppose an error has occurred, then

  $c^{received}(x) = c(x) + e(x)$, e(x) - error polynomial
  $d^{received}(x) = (c(x) + e(x))/g(x)$

- Unless e(x) is a multiple of g(x), the received codeword will not be evenly dividable by g(x)

Decoding/detecting process

- We detect errors by checking whether $c^{received}(x)$ is evenly dividable by g(x)
- If yes, we assume that there is no error and $d^{received}(x) = d(x)$
- If there is a reminder, we assume that there is an error

Undetectable errors

- However, if e(x) is a multiple of g(x), the reminder of e(x)/g(x) is 0 and the error will not be detected

Example of error detection

$d(x) = (1011) = x^3 + x^2 + 1$

$g(x) = x^3 + x + 1$

$c(x) = d(x).g(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$

Let $e(x) = x^3+1$, then $c^{received} = x^6 + x^5 + x^4 + x^2 + x$

$c^{received(x)}/g(x) = (x^3 + x^2) + x/(x^3 + x + 1)$

Reminder is not 0, so the error is detected

HW for encoding/decoding of cyclic codes

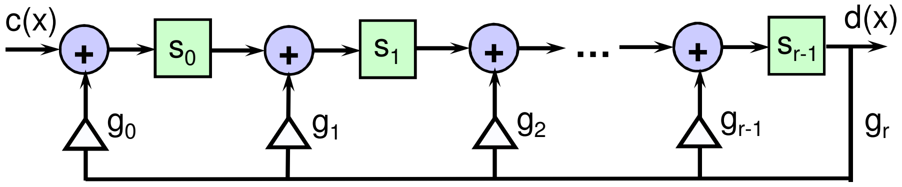
- Encoding and decoding is done using linear feedback shift registers (LFSRs)
- LFSR implements polynomial division by generator polynomial g(x)

LSFR

- Linear feedback shift register consists of:
  - register cells $s_o, s_1, \ldots, s_{r-1}$, where r = n-k is the degree of g(x)
  - XOR-gates between the cells
  - feedback connections to XOR, with weights
  - clock

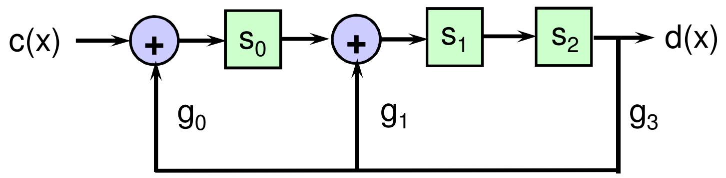
LFSR

- Weights g_i are the coefficients of the generator polynomial $g(x)=g_0+g_1x^1+\ldots+ g_r x^r$
  - g_i=0 means 'no connection'
  - g_i=1 means 'connection'
  - g_r is always connected



Example

- LFSR for $g(x)=1+x+x^3$



$s_0^+=s_2+c(x)$
$s_1^+=s_0+s_2$
$s_2^+=s_1$

Example: decoding, no error

Suppose the word to decode is [1010001], i.e $c(x) = 1 + x^2 + x^6$. Most significant bit is fed first.

	c(x)	s_0	s_1	s_2	d(x)
t_0		0	0	0	
t_1	1	1	0	0	0
t_2	0	0	1	0	0
t_3	0	0	0	1	0
t_4	0	1	1	0	1
t_5	1	1	1	1	0
t_6	0	1	0	1	1
t_7	1	0	0	0	1

$s_0^+= s_2+c(x)$
$s_1^+= s_0+ s_2$
$s_2^+= s_1$

$d(x) = 1 + x + x^3$. Most significant bit comes out first.

Example: decoding, with error

Suppose an error has occurred in the 4th bit, i.e. we received [1011001] instead of [1010001].

	c(x)	s_0	s_1	s_2	d(x)
t_0		0	0	0	
t_1	1	1	0	0	0
t_2	0	0	1	0	0
t_3	0	0	0	1	0
t_4	1	0	1	0	1
t_5	1	1	0	1	0
t_6	0	1	0	0	1
t_7	1	1	1	0	0

$s_0^+ = s_2 + c(x)$
$s_1^+ = s_0 + s_2$
$s_2^+ = s_1$

The syndrome [110] matches the 4th column of the check matrix H.

Encoding for separable cyclic codes

- Division can be used for encoding of a separable (n,k) cyclic code
  - shift data by n-k positions, i.e. multiply d(x) by x^{n-k}
  - use LFSR to divide d(x) x^{n-k} by g(x). The reminder r(x) is contained in the register
  - append the check bits r(x) to the data by adding r(x) to d(x) x^{n-k}:

$$c(x) = d(x)\, x^{n-k} + r(x)$$

Example: encoding for (7,4) code

- Let $d(x) = x+x^3$ and $g(x)=1+x+x^3$
  - n=7, k=4
  - shift data by n-k=3 positions:

$$d(x) \cdot x^3 = (x+x^3)x^3 = x^4+x^6$$

  - divide $d(x)\, x^{n-k}$ by $g(x)$ to compute the $r(x)$

$$x^4+x^6 = (1+x^3)(1+x+x^3)+(1+x)$$

  - $c(x) = d(x)\, x^{n-k} + r(x)$

$$c(x) = 1+x+x^4+x^6$$

CRC codes

- Cyclic Redundancy Check (CRC) codes are separable codes with specific generator polynomials, chose to provide high error detection capability for data transmission and storage
- Common generator polynomials are:

CRC-16: $1 + x^2 + x^{15} + x^{16}$

CRC-CCITT: $1 + x^5 + x^{12} + x^{16}$

CRC-32: $1 + x + x^2 + x^4 + x^7 + x^8 + x^{10} + x^{11} + x^{12} + x^{16} + x^{22} + x^{23} + x^{26} + x^{32}$

CRC codes

- CRC-16 and CRC-CCITT are widely used in modems and network protocols in the USA and Europe, respectively, and give adequate protection for most applications
  - the number of non-zero terms in their polynomials is small (just four)
  - LFSR required to implement encoding and decoding is simpler
- Applications that need extra protection, e.g. DD, use CRC-32

Encoding/decoding

- The encoding and decoding is done either in software, or in hardware using the usual procedure for separable cyclic codes
- To encode:
  - shift data polynomial right by deg(g(x)) bit position
  - divided it by the generator polynomial
  - the coefficients of the remainder form the check bits of the CRC codeword

Encoding/decoding

- The number check bits equals to the degree of the generator polynomial
  - an CRC detects all burst error of length less or equal than deg(g(x))
- CRC also detects many errors which are larger than deg(g(x))
  - apart from detecting all burst errors of length 16 or less, CRC-16 and CRC-CCITT are also capable to detect 99.997% of burst errors of length 17 and 99.985% burst errors of length 18

Reed-Solomon codes

- Reed-Solomon (RS) codes are a class of separable cyclic codes used to correct errors in a wide range of applications including
  - storage devices (tapes, compact disks, DVDs, bar-codes), wireless
  - communication (cellular telephones, microwave links), satellite
  - communication, digital television, high-speed modems (ADSL, xDSL).

Reed-Solomon codes

- The encoding for Reed-Solomon code is done the using the usual procedure
  - codeword is computed by shifting the data right n-k positions, dividing it by the generator polynomial and then adding the obtained reminder to the shifted data
- A key difference is that groups of m bits rather than individual bits are used as symbols of the code.
  - usually m = 8, i.e. a byte.
  - the theory behind is a field $Z^m{}_2$ of degree m over {0,1}

Encoding

- An encoder for an RS code takes k data symbols of s bits each and computes a codeword containing n symbols of m bits each
- A Reed-Solomon code can correct up to $\lfloor n-k \rfloor/2$ symbols that contain errors

Example: RS(255,223) code

- A popular Reed-Solomon code is RS(255,223)
  - symbols are a byte (8-bit) long
  - each codeword contains 255 bytes, of which 223 bytes are data and 32 bytes are check symbols
  - n = 255, k = 223, this code can correct up to 16 bytes containing errors
  - each of these 16 bytes can have multiple bit errors.

Decoding

- Decoding of Reed-Solomon codes is performed using an algorithm designed by Berlekamp
  - popularity of RS codes is due to efficiency this algorithm to a large extent.
- This algorithm was used by Voyager II for transmitting pictures of the outer space back to Earth
- Basis for decoding CD in players

Summary of cyclic codes

- Any end-around shift of a codeword produce another codeword
- code is characterized by its generator polynomial g(x), with a degree (n-k), n = bits in codeword, k = bits in data word
- detect all single errors and all multiple adjacent error affecting (n-k) bits or less

Unordered codes

- Designed to detect unidirectional errors
- An error is unidirectional if all affected bits are changed to either 0 $\rightarrow$ 1 or 1 $\rightarrow$ 0, but not both
- Example:
  - correct codeword: 010101
  - same codeword with unidirectional errors:

110101	000101
111101	000001
111111	000000

Unidirectional error detection

- Theorem: A code C detects all unidirectional errors if and only if every pair of codewords in C is unordered
- two binary n-tuples x and y are ordered if either $x_i \leq y_i$ or $x_i \geq y_i$ for all $i \in \{1,2,...,n\}$
- Examples of ordered codewords:

$$0110 < 0111 < 1111$$

$$0110 > 0100 > 0000$$

Unidirectional error detection

- A unidirectional error always changes a word x to a word y which is either smaller or greater than x
- A unidirectional error cannot change x to a word which is not ordered with x
- Therefore, if any pair of codewords are unordered, a unidirectional error will never transform a codeword to another codeword

m-of-n code

- Code words are *n* bits in length and contain exactly *m* 1's
  - C_d = 2, detect single-bit errors
  - detects all unidirectional errors
- (+) simple to understand
- (-) if non-separable, encoding and decoding is difficult to organize

k-of-*2k* code

- Take the original *k* bits of information and append *k* bits so that the resulting 2*k*-bit word has exactly *k* 1s
  - (-) 100% redundancy
  - (+) separable code, so encoding and decoding are easy to organize

data	3-of-6 code
000	000 111
001	001 110
010	010 101
…	…
111	111 000

Berger code

- Append c check bits to k data bits

$$c = \lceil log_2(k+1) \rceil$$

- separable code
- how to create code word:
  - count number of 1's in k data bits
  - complement resulting binary number and append it to the data bits

Example of Berger codeword

data = (0111010), k = 7

$c = \lceil log_2(7+1) \rceil = 3$

number of 1's in (0111010) is 4 = (100)

complement of (100) is (011)

resulting codeword is (0111010011)

Berger code capability

- Berger code detects all unidirectional errors
- for the error detection capability it provides, the Berger code uses the fewest number of check bits of the available separable unordered codes

Arithmetic codes

- For checking arithmetic operations
  - before the operation, data is encoded
  - after the operation, code words are checked
- Arithmetic code is the invariant to “*” if:

$$A(b*c) = A(b) * A(c)$$

b, c - operands
*A(b), A(c), A(b*c)* - codes for b, c and b*c

Examples of arithmetic codes

- Two common types of arithmetic codes are
  - AN codes
  - residue codes

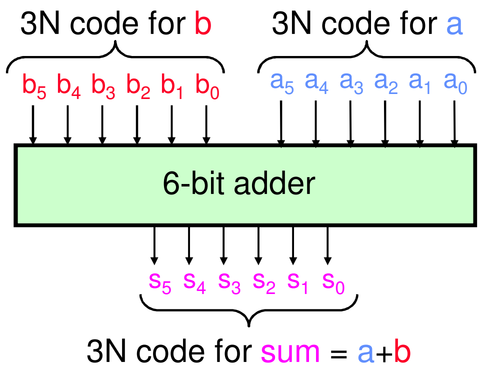
AN code

- *AN* code is formed by multiplying each data word *N* by some constant *A*
- *AN* codes are invariant to addition (and subtraction):

$$A(b + c) = A(b) + A(c)$$

- If no error occurred, *A(b+c)* is evenly divisible by *A*

Adder protected by *3N* code



Original data are 4-bit long. By multiplying them by 3, we obtain code words (6-bit long)

data	code word
0000	000000
0001	000011
0010	000110
...	...
1111	101101

Addition using *3N* code - fault-free

Normal operation:

a = 0 1 0 0 1 0 (3N code of 6)

\+ b = 0 0 0 0 1 1 (3N code of 1)

s = 0 1 0 1 0 1 (3N code of 7)

Addition using *3N* code - with faults

a = 0 1 0 0 1 0 (3N code of 6)

\+ b = 0 0 0 0 1 1 (3N code of 1)

s = 0 1 0 1 1 1 (23 is not evenly divisible by 3 i.e. not a valid code word)

If s_1 is stuck to “1”:

Selecting the value of *A*

- For binary codes, the constant A shouldn’t be a power of two
  - otherwise multiplication by A results a left shift of the original data
  - error in a single bit yields a codeword evenly divisible by A (valid), so it will not be detected
- 3N code is easy to encode using n+1 bit adder: create 2N by shift and add N to it

Residue codes

- Residue codes are created by computing a residue for data and appending it to the data
- The residue is generate by dividing a data by a integer, called modulus.
- Decoding is done by simply removing the residue

Residue codes

- Residue codes are invariants with respect to addition, since

$$(b + c) \bmod m = b \bmod m + c \bmod m$$

where b and c are data words and m is modulus

- This allows us to handle residues separately from data during addition process.
- Value of the modulus determines the information rate and the error detection capability of the code

Next lecture

- Time redundancy

Read chapter 6

of the text book