Xi'an Jiaotong-Liverpool University
西交利物浦大学

EEE220 Instrumentation and Control System

2018-19 Semester 2

Dr. Qing Liu

Email: qing.liu@xjtlu.edu.cn

Office: EE516

Department of Electrical and Electronic Engineering

14 March, 2019

Lecture 8

Outline

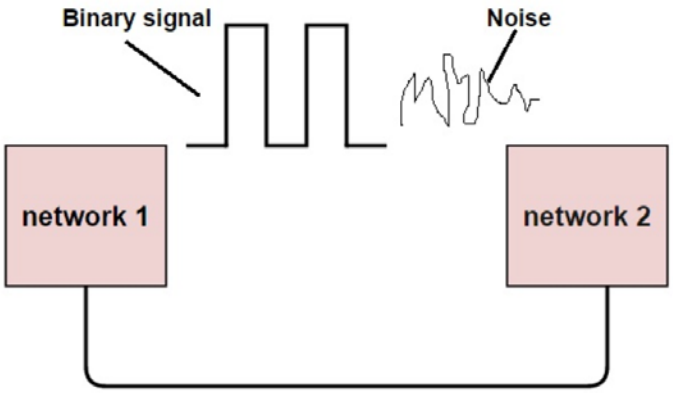
Communication System (cont'd)

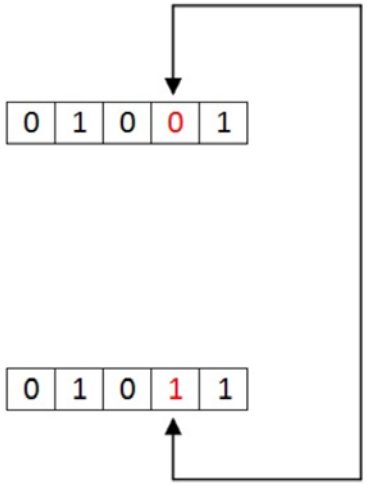
- **Error Control Methods**
  - Error Detecting Code: Parity Check Code
  - Error Correcting Code: Hamming Code, Chain Code

Data Transmission Error in Digital Systems

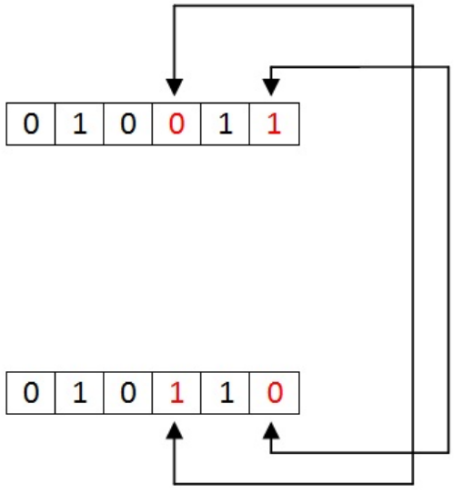
- In digital instrumentation systems, the analog signals will be changed into digital sequence (in the form of bits);
- The transmission of **digitised** information requires an acceptable bit error rate that depends on the accuracy required. For example, with speech an error rate of 1 in 10^5 bits is acceptable;
- The data errors will cause loss of important / secured data. In instrumentation systems, it is usually the case that any uncorrected error at all is a disaster - even one bit of change in data may affect the whole system's performance.

In many cases, methods of error control is essential.





Single bit error



Multiple bit error

Error Control Methods

Generally two approaches:

1. Feedback Error Control

- only contains sufficient information to detect when errors are present and then to request a re-transmission so that the information is sent again (e.g. **parity check**, block sum check etc.).

2. Forward Error Control

- each transmitted character or frame contains additional (redundant) information so that the receiver can not only detect when errors are present but also correct these errors (e.g. **Hamming codes**, **chain codes** etc.)

Both require an overhead that increases the number of bits sent and therefore increases the bandwidth of the transmission path required to achieve a given data rate.

Parity Check Code

An extra bit is attached to the word and is made either 0 or 1 so that the total number of 1 in that word is always even (even parity).

If a single error occurs then the number of 1's becomes odd and a signal is sent back to the transmitter to request re-transmission of the word.

- Example:

Message (data word)	1010	
Code word (even parity)	10100	(parity bit: 0)
Code word (odd parity)	10101	(parity bit: 1)

The bandwidth requirement is increased only marginally provided the signal-to-noise ratio is high (so only a very few words need to be re-transmitted) :

$$BW = f_{max} \cdot (N + 1)$$

where f_{max} is the maximum frequency of signal, N is the number of bits required to represent the signal sample.

------ However it has limited error detection capabilities and no error correction is possible.

Even vs. Odd Parity

Even Parity

- If data has even number of 1's, the parity bit is 0. E.g. data is 10000001 -> parity bit 0
- Odd number of 1's, the parity bit is 1. E.g. data is 10010001 -> parity bit 1

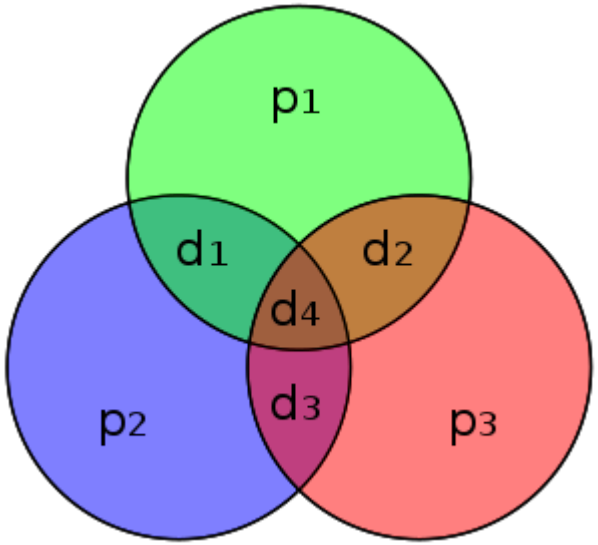
Odd Parity

- If data has odd number of 1's, the parity bit is 0. E.g. data is 10011101 -> parity bit 0
- Even number of 1's, the parity bit is 1. E.g. data is 10010101 -> parity bit 1

3 bit data			Message with even parity		Message with odd parity	
A	B	C	Message	Parity	Message	Parity
0	0	0	000	0	000	1
0	0	1	001	1	001	0
0	1	0	010	1	010	0
0	1	1	011	0	011	1
1	0	0	100	1	100	0
1	0	1	101	0	101	1
1	1	0	110	0	110	1
1	1	1	111	1	111	0

Hamming Code

- This error correcting code technique is developed by R. W. Hamming. It's a linear error correcting code, and allows correction of single-bit error.
- In this scheme, further (redundant) parity bits are added to the original data word. In such a way that when an error occurs, they indicate in which bit the error has occurred. This allows correction of the message (If an error is detected, the bit is flipped).
- The number of parity bits depends upon the number of massage bits.

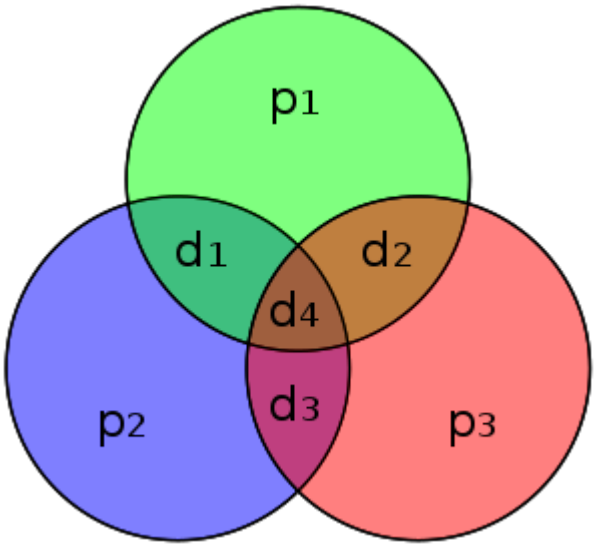


The Hamming(7,4) code

E.g. message (d_1 d_2 d_3 d_4): 1001
Parity bits? (if consider even parity)

- p_1: 0
- p_2: 0
- p_3: 1

How to Correct Errors?



The Hamming(7,4) code

- When the message is received, the error detection circuits within the receiver carry out each of these even parity checks.
- If none of them fail then no error has occurred.
- If an error in any data bit occurs, then at least two parity checks will fail and which two allows the error to be identified.
- If only one parity check fails then the error is in the corresponding parity check bit itself.

E.g. received codeword (d_1 d_2 d_3 d_4 p_1 p_2 p_3) is 1010010:

- Errors occur at which bits?
- Correct data message should be?

Number of Parity Bits Required

Hamming code can be applied to data of any length.

If the original data word contains n data bits then 'p' extra parity bits are required where

$$2^p \geq n + p$$

e.g. 4-bit data word, it will require 3 extra parity bits (n = 4, p = 3) since

$$2^3 \geq 4 + 3$$

We now have a total of 7 bits in codeword.

Parity bits	Total bits	Data bits	Name	Rate
2	3	1	Hamming(3,1) (Triple repetition code)	1/3 ≈ 0.333
3	7	4	Hamming(7,4)	4/7 ≈ 0.571
4	15	11	Hamming(15,11)	11/15 ≈ 0.733
5	31	26	Hamming(31,26)	26/31 ≈ 0.839
6	63	57	Hamming(63,57)	57/63 ≈ 0.905
7	127	120	Hamming(127,120)	120/127 ≈ 0.945
8	255	247	Hamming(255,247)	247/255 ≈ 0.969

What's the length of the codeword if assume an 8-bit data word?

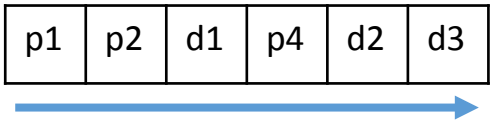
Hamming Distance

Hamming distance: the number of bits which differ between two binary strings. More formally, the distance between two strings A and B is $\sum | A_i - B_i |$.

Hamming Code example for 3-bit data.

Data word (message)

000
001
010
011
100
101
110
111



P1 looks at bit d1, d2;
P2 looks at bit d1, d3;
P4 looks at bit d2, d3.

Code word

00_00_{00}
01_01_{01}
10_01_{10}
11_00_{11}
11_10_{00}
10_11_{01}
01_11_{10}
00_10_{11}

Distance from pattern:		0	1	2	3	4	5	6	7
Pattern:									
0	000000	–	3	3	4	3	4	4	3
1	010101	3	–	4	3	4	3	3	4
2	100110	3	4	–	3	4	3	3	4
3	110011	4	3	3	–	3	4	4	3
4	111000	3	4	4	3	–	3	3	4
5	101101	4	3	3	4	3	–	4	3
6	011110	4	3	3	4	3	4	–	3
7	001011	3	4	4	3	4	3	3	–

--- Hamming distance for this code: 3
(minimum distance between each two patterns.)

- **If assume only 1 bit error, can always tell the correct data word.**

  --- List all patterns and find the nearest one.

Problem: computationally expensive. Especially with longer strings (much more patterns). - More efficient way?

How It Works

With Hamming code, single-bit error can be found quickly by just looking at one pattern:

p1	p2	d1	p4	d2	d3

Example 1:

Let's say error in a data bit:
100 sent
111000
became: 111001
data extracted: 101
Do parity checks:
Check p1 d1d2 -1 1 0 - OK
Check p2 d1d3 -1 1 1 - WRONG
Check p4 d2d3 -0 0 1 - WRONG

The bad bit is bit 2 + 4 = bit 6. Data was corrupted. Data should be 100.

Example 2:

Let's say error in a check bit:
100 sent
111000
became: 011000
data extracted: 100
Do parity checks:
Check p1 d1d2 -0 1 0 - WRONG
Check p2 d1d3 -1 1 0 - OK
Check p4 d2d3 -0 0 0 - OK

The bad bit is bit 1. Check bit was corrupted. Data is fine.

General Rules for Encoding Data

If consider even parity:

Message: $d_1\ d_2\ d_3\ d_4 d_5\ d_6\ d_7$

Code word:

bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7	bit 8	bit 9	bit 10	bit 11	
p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	.....

1. Mark all bit positions that are powers of two as parity bits. (positions 1, 2, 4, 8, 16 etc.)
2. All other bit positions are for the data to be encoded. (positions 3, 5, 6, 7, 9, 10 etc.)
3. Each parity bit calculates the parity for some of the bits in the code word. The position of the parity bit determines the sequence of bits that it alternately checks and skips.
   - p1: check 1 bit, skip 1 bit, check 1 bit, skip 1 bit, etc. (bit 1,3,5,7,9,11,13,15,...)
   - p2: check 2 bits, skip 2 bits, check 2 bits, skip 2 bits, etc. (bit 2,3,6,7,10,11,14,15,...)
   - p4: check 4 bits, skip 4 bits, check 4 bits, skip 4 bits, etc. (bit 4,5,6,7,12,13,14,15,20,21,22,23,...)
   - p8: check 8 bits, skip 8 bits, check 8 bits, skip 8 bits, etc.
4. Set a parity bit to 1 if the total number of ones in the positions it checks is odd. Set a parity bit to 0 if the total number of ones in the positions it checks is even.

Syndrome to Correct Data

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity bit coverage	p1	X		X		X		X		X		X		X		X		X		X		...
	p2		X	X			X	X			X	X			X	X			X	X		
	p4				X	X	X	X					X	X	X	X					X	
	p8								X	X	X	X	X	X	X	X						
	p16																X	X	X	X	X	

- When receive a code word, do parity check for each parity bit.
- The result of the parity checks is known as the syndrome and is an p bit word pointing directly to the error bit.
- Note that the position of the parity bits in the transmitted word are chosen where it is the only designated bit in a column. This ensures that:

1) An error in a single parity bit only affects the outcome of only one parity check.
2) Each parity check group contains one and only one parity bit that can be chosen to produce odd (or even) parity for the group.

e.g. Use hamming (7,4) code to transmit 4-bit data. In received code word, if parity checks for p1 and p4 fail, but p2 pass, then the syndrome would be 1 0 1 - Pointing directly to bit 5 (4+1) as the error bit.

Example 8.1

A byte of data: 10011010
Create the data word, leaving spaces for the parity bits: _ _ 1 _ 0 0 1 _ 1 0 1 0
Calculate the parity for each parity bit (a ? represents the bit position being set):

- Position 1 checks bits 1,3,5,7,9,11:
  ? _ **1** _ **0** 0 **1** _ **1** 0 **1** 0. Even parity so set position 1 to a 0: **0** _ **1** _ **0** 0 **1** _ **1** 0 **1** 0
- Position 2 checks bits 2,3,6,7,10,11:
  0 **?** **1** _ 0 **0** **1** _ 1 **0** **1** 0. Odd parity so set position 2 to a 1: 0 **1** **1** _ 0 **0** **1** _ 1 **0** **1** 0
- Position 4 checks bits 4,5,6,7,12:
  0 1 1 **?** **0** **0** **1** _ 1 0 1 **0**. Odd parity so set position 4 to a 1: 0 1 1 **1** **0** **0** **1** _ 1 0 1 **0**
- Position 8 checks bits 8,9,10,11,12:
  0 1 1 1 0 0 1 **?** **1** **0** **1** **0**. Even parity so set position 8 to a 0: 0 1 1 1 0 0 1 **0** **1** **0** **1** **0**
- Code word: 011100101010.

- Suppose the codeword that was received was 011100101110 instead.
Please calculate which bit was wrong and correct it.

Method: verify each parity bit. Write down all the incorrect parity bits. Doing so, you will discover that parity bits 2 and 8 are incorrect. It is not an accident that 2 + 8 = 10, and that bit position 10 is the location of the bad bit. In general, check each parity bit, and add the positions that are wrong, this will give you the location of the bad bit.

Example 8.2

Receiver takes information 110101000, even parity was considered. Is received data 110101000 correct ? If it was wrong, what is the correct data ?

Step 1. find the number of data bits and parity bits;

- 5 data bits and 4 parity bits.

Step 2. do parity check for each parity bit;

Parity bit should place at 2^n, ***n*** = 0, 1, 2, 3, … So 1, 2, 4, 8, ….

110**1**010**0**0, bold represent parity bits.

p1 check: 10000, odd
P2 check: 1010, even
P4 check: 1010, even
P8 check: 00, even

Step 3. determine bad bit position according to the check results.

Bad bit at bit 1. Therefore the first parity bit was wrong, data transmitted should be 00100, correct code word should be 010101000.

Chain Code

-------- Another error correcting code (cycling code).

Method of construction of chain code for a 4-bit data word is as follows:-

Use modulo-2 addition (exclusive-OR) on the first two bits and add the result to the end of the word. Eventually, the original data word will re-appear.

Message: 1101

1010
0101
1011
0111
1111
1110
1100
1000
0001
0010
0100
1001
0011
0110 - stop here!
1101 - original data word re-appears on next sequence

The chain code is then the sequence of bits obtained by taking the first bit from each row.

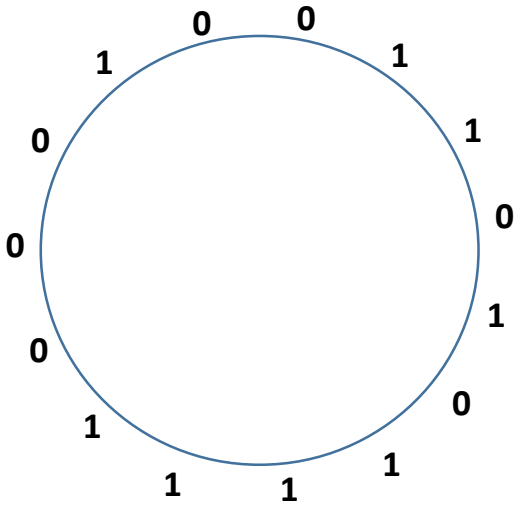
i.e. **110101111000100**

- note that this has the original date word as its first four bits and the remaining bits can be regarded as extra (redundant) bits for error correction.

If a different four bit data word is chosen and the corresponding chain code produced, the original data word will re-appear. It turns out that all the possible chain codes are cyclic and *all follow the same cycle, but with different starting points*.

Hence the name 'chain codes'.

Four bit data words give rise to a 15 bit chain code.

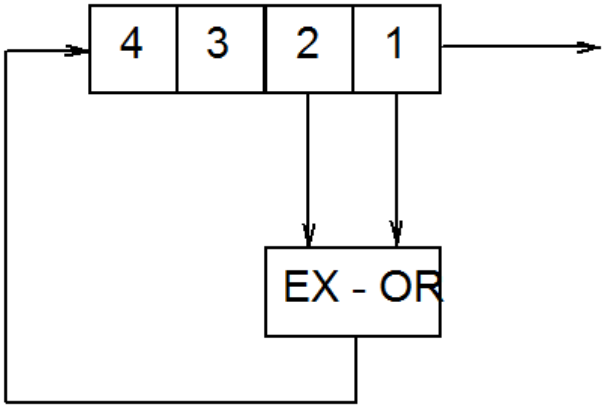


e.g. 111100010011010 is the chain code for 1111.

What's the chain code for 1000?

For the chain code, each allowed data word is stored in the receiver.

Chain Code Generator



Other sized data words use EX-OR on different bits.

Message Size	Bits used	Word size
3	1,3	7
4	1,2	15
5	1,3	31
6	1,2	63
7	1,5	127
N	---	$2^N - 1$

Properties of A Chain Code

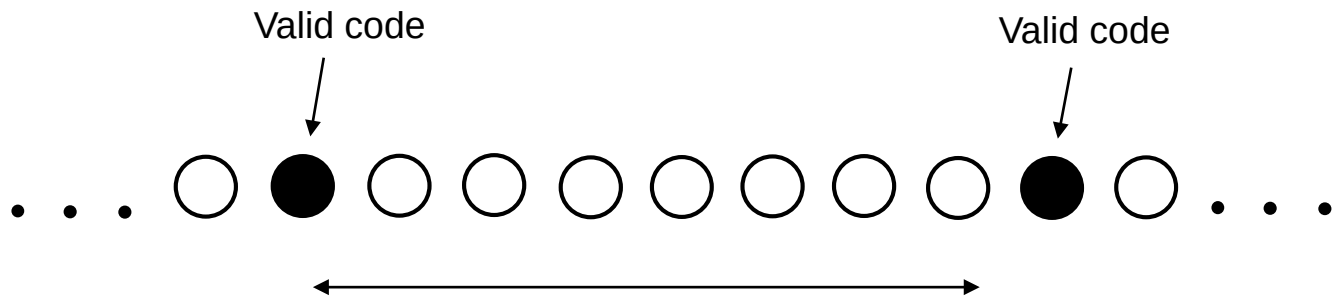
1. If number of data bit is N, chain code length = $2^N - 1$;
2. All chain code words are all the same cyclic sequence. The feature which distinguishes one code from another is the **starting point** in the chain.
3. In any chain code there is one more 1 than 0's.
4. There are 2^{N-1} differences between any chain code and every other one.

e.g. 1101011111000100
1111000100111010

i.e. 8 differences ($2^{4-1} = 8$)

This fact is the basis of the ability of a chain code to identify the correct code in the presence of errors.

Hamming Distance for Chain Codes



Hamming distance is the number of differences between each two valid codes.

(Hamming distance for chain code of N-bit word is 2^{N-1}).

A code with Hamming distance of 3 can detect up to 2 errors but correct only 1

….Hamming distance of 4 can detect up to 3 errors but correct only 1

….Hamming distance of 5 can detect up to 4 errors and correct up to 2

….Hamming distance of 6 can detect up to 5 errors and correct up to 2

A code with Hamming distance of d can detect up to $d - 1 = 2^{N-1} - 1$ errors and correct up to $\text{int}\left(\frac{d-1}{2}\right) = 2^{N-2} - 1$.

Working Rules

- If an error in any bit occurs, there will then be at least 2^{N-1}-1 differences between any other code and only one difference between the received code and the correct one. So the correct code can be identified.
- If two errors occur, there will be at least 2^{N-1}-2 differences between any other code and two differences between the received code and the correct one. So the correct code can still be identified.
- This logic can be continued. Since the number of differences between any two codes is 2^{N-1}, then there can be half this many errors before the number of differences with the correct code becomes greater than that with any other (incorrect) code, i.e. 2^{N-2} errors. Thus the number of errors that can be tolerated before confusion with an incorrect code occurs (i.e. the number that can be 'corrected') is one less than this, 2^{N-2} -1.

e.g. a data word with 8 bits will produce a chain code of length $\mathbf{2^N-1}$= 255 bits. It contains 128 1's and 127 0's. It has $\mathbf{2^{N-1}}$ = 128 differences between any two chain codes produced from an eight bit data word and can therefore tolerate 2^{N-2} -1 = 63 errors and still allow the correct code to be determined.

Example 8.3

Suppose the previous signal **110101111000100** was received with three errors, becoming:

110001101000110

The receiver compares the received code with each of the possible four-bit chain codes:

Possible codes	**number of differences with received code**
0000	NOT ALLOWED
000100110101111	9
001001101011110	5
001101011110001	11
010011010111100	9
010111100010011	7
011010111100010	7
011110001001101	9
100010011010111	7
100110101111000	9
101011110001001	9
101111000100110	7
110001001101011	5
110101111000100	3
111000100110101	7
111100010011010	9

For a four bit code, the maximum number of errors that can be corrected is $=2^{N-2}-1=3$. Thus the example above is at the maximum limit (also known as the 'Hamming distance').

Thank You !