ECE454/544: Fault-Tolerant Computing & Reliability Engineering

Lecture #5 –

Information Redundancy Techniques (II)

Instructor: Dr. Liudong Xing

Fall 2022

Administrative Issues (9/21, Wednesday)

- Homework#1 due today
- Homework#2 assigned today
  - Please download the problems from the course website: https://xingteaching.sites.umassd.edu/
  - Due **Sept. 28, Wednesday**
- Project Proposal
  - Due **Oct. 5, Wednesday**
  - Refer to Proposal Guideline on the course website

Review of Lecture #4

- Basic concepts:
  - Code, code word, binary code, error detecting /correcting code, encoding / decoding process
  - Error models for code development: bit / symmetric / asymmetric / unidirectional /byte errors
  - Hamming distance, code distance, error detection/correction capabilities (3 theorems)
- Parity codes
  - Single-bit and multiple-bit parity codes
  - Hamming single error correcting (SEC) codes
    - Calculate number of check bits
    - Arrange bit positions
    - Generate the check bits
    - Correct the erroneous bit according to the syndrome word
  - Horizontal and Vertical parity code: can correct any single-bit errors in groups of data words

Agenda

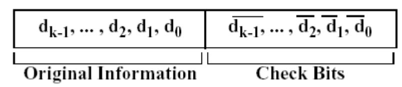
L#4
- ✓ Basic concepts
- Example codes
  - √ Parity codes: Hamming SEC code, Horizontal and Vertical Parity code
  - **m-of-n**
  - Berger
  - Checksums
  - Cyclic
  - Arithmetic
- Code selection issue

m-of-n Codes

- Each code word has exactly *m* 1s out of a total of *n* bits.
  - each code word has a weight of *m*.
- CD = ?   2
- The m-of-n code can detect all single-bit errors and all multiple, unidirectional errors
  - Any single-bit error forces the resulting erroneous word to have either (m+1) or (m-1) ones.
  - Unidirectional: errors are either a change of a 1 to a 0 or a change of a 0 to a 1

Separable m-of-n Codes

- It is possible to construct separable m-of-n codes as:

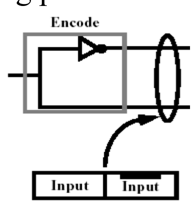


k-of-2k code

Example k-of-2k Code

3-of-6 code	
000	111
001	110
010	101
011	100
100	011
101	010
110	001
111	000
Original Information	Appended Check Bits

- Disadvantage: 100% redundancy
- Advantage: separable code →both encoding and decoding processes are simple



Property of m-of-2m Code

- Suppose we have two code words A and B

$$A = (a_{m-1}, \ldots, a_0, \overline{a_{m-1}}, \ldots, \overline{a_0}) \qquad B = (b_{m-1}, \ldots, b_0, \overline{b_{m-1}}, \ldots, \overline{b_0})$$

$$A + B = (a_{m-1} + b_{m-1}, \ldots, a_0 + b_0, \overline{a_{m-1}} + \overline{b_{m-1}}, \ldots, \overline{a_0} + \overline{b_0})$$

- In modulo-2 addition

$$\bar{x} + \bar{y} = x + y$$

$$A + B = (a_{m-1} + b_{m-1}, \ldots, a_0 + b_0, a_{m-1} + b_{m-1}, \ldots, a_0 + b_0)$$

data portion | check portion

- The modulo-2 addition of two m-of-2m code words generates a code word where the check portion is the exact replica of the functional portion.

Non-separable m-of-n Code

- The encoding and decoding can be performed by look-up tables
- Can detect any single-bit errors

Non-separable 2-of-5 code for BCD data		
Dec. digit	BCD data	2-of-5 code
0	0000	00011
1	0001	11000
2	0010	10100
3	0011	01100
4	0100	10010
5	0101	01010
6	0110	00110
7	0111	10001
8	1000	01001
9	1001	00101

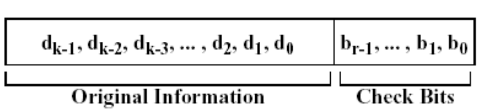
Have the same error detection capabilities as the single-bit parity code

Agenda

✓ Basic concepts
- Example codes
  - √ Parity codes: Hamming SEC code, Horizontal and Vertical Parity code
  - √ m-of-n
  - **Berger**
  - Checksums
  - Cyclic
  - Arithmetic
- Code selection issue

Berger Codes

- A **separable** code formed by appending a check symbol which is simply the count of the number of 0s (N_0) in the original information.



- The check bits $b_{r-1}, \dots, b_1, b_0$ are the binary representation of N_0

Example

- Berger code words for 4-bit information words

Original Information	Berger Code
0000	0000|100
0001	0001|011
0010	0010|011
0011	0011|010
0100	0100|011
0101	0101|010
0110	0110|010
0111	0111|001
1000	1000|011
1001	1001|010
1010	1010|010
1011	1011|001
1100	1100|010
1101	1101|001
1110	1110|001
1111	1111|000

Characteristics of Berger Codes

- Berger codes detect all unidirectional errors in the information bits

  since any number of 0 to 1 errors will decrease the number of 0s in the information, and any number of 1 to 0 errors will increase the number of 0s in the information

- Berger codes detect all unidirectional errors in the check bits

  since any number of 1 to 0 errors will decrease the check value, and any number of 0 to 1 errors will increase the check value

Characteristics of Berger Codes (Cont'd)

- If 1 to 0 errors occur in both the information and the check symbol, the number of 0s in the information will increase while the check value will decrease

- If 0 to 1 errors occur in both the information and the check symbol, the number of 0s (zeros) in the information will decrease while the check value will increase

Berger Codes: How many check bits?

- For k information bits, the number of check bits N_c in the Berger code is given by

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

# of information bits	# of check bits	Percentage redundancy
4	3	75%
8	4	50%
16	5	31.25%
32	6	18.75%
64	7	10.94%

- The redundancy is high when the number of information bits is small
- As number of information bits increases, the efficiency improves substantially

Invariant of Berger Codes

- It is possible to manipulate Berger code words such that they are invariant to the following operations
  - Arithmetic operations: addition, subtraction, multiplication and division
  - Logical operations: AND, OR, XOR, NOT, ROTATE, and SHIFT

Addition Example

- Suppose we want to ADD two n+1-bit binary numbers and a carry-in bit c_{in}

$$X = (x_n, x_{n-1}, \ldots, x_1, x_0) \quad Y = (y_n, y_{n-1}, \ldots, y_1, y_0)$$

- Let the Berger check symbol of X be Xc, of Y be Yc, of the sum S be Sc and the check symbol of the internal carries C as Cc, thus:

$$S_c = X_c + Y_c - C_c - c_{in} + c_{out}$$

- Example:

$$X = (10110011) \quad Y = (11001011) \quad c_{in} = 1$$

$$X_c = 3 \quad Y_c = 3$$

$$S = (01111111) \quad c_{out} = 1 \quad C = (10000011)$$

Note that the most significant bit of C is c_{out}

$$S_c = X_c + Y_c - c_{in} - C_c + c_{out} = 3 + 3 - 1 - 5 + 1 = 1$$

AND Example

- Suppose we want to AND two n+1-bit binary numbers

$$X = (x_n, x_{n-1}, \ldots, x_1, x_0)$$

$$Y = (y_n, y_{n-1}, \ldots, y_1, y_0)$$

- Let the Berger check symbol of X be Xc, of Y be Yc, and check symbol of X AND Y be $(X \wedge Y)_c$, thus:

$$(X \wedge Y)_c = X_c + Y_c - (X \vee Y)_c$$

- Example:

$$X = (10110011) \quad Y = (11001011)$$

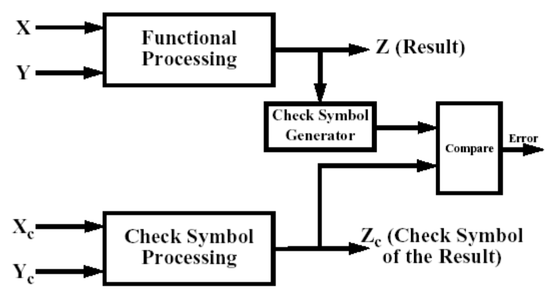
$$X_c = 3 \quad Y_c = 3$$

$$X \wedge Y = (10000011) \quad X \vee Y = (11111011)$$

$$(X \wedge Y) = X_c + Y_c - (X \vee Y)_c = 3 + 3 - 1 = 5$$

Error Detection Using Berger Codes

- Berger code processor

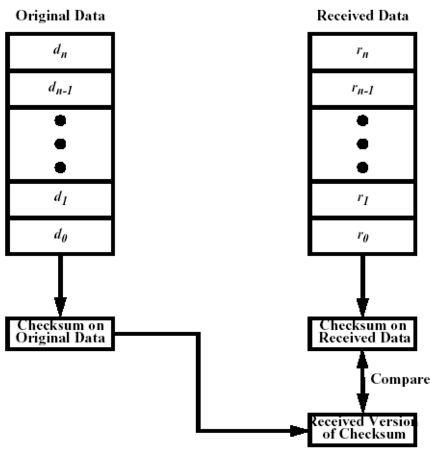


Agenda

✓ Basic concepts
- Example codes
  - √ Parity codes: Hamming SEC code, Horizontal and Vertical Parity code
  - √ m-of-n
  - √ Berger
  - – Checksums
  - – Cyclic
  - – Arithmetic
- Code selection issue

Checksum Codes

- A form of separable codes
- Checksum is a quantity of information added to the block of data to help detect errors



Dr. Xing 21

The Checksum

- Basically the sum of original data
- Difference between various forms of checksum is the way in which the summation is generated
  - Single-precision
  - Double-precision
  - Honeywell
  - Residue

All checksums can detect errors but not locate them!

Dr. Xing 22

Single-Precision Checksum (SPC)

- Formed by performing binary addition of the data to be protected by the checksum and ignoring any overflow that occurs;
  - An overflow occurs if the binary sum of the n-bit data exceeds 2^n-1
- Formed by adding the n-bit data in a modulo-2^n style
- Example:

	0	1	1	1	Original Data
	0	0	0	1	
	0	1	1	0	
(addition) +	1	0	0	0	
(carry ignored) 1	0	1	1	0	Checksum

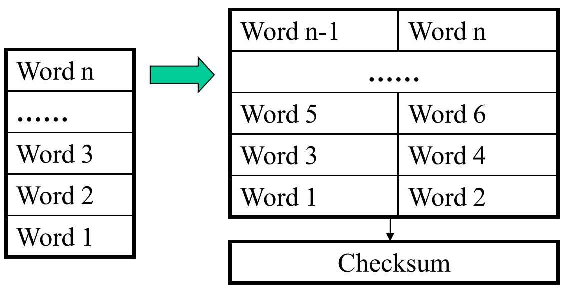
- Difficulty: information, thus the ability to detect errors can be lost in the ignored overflow → an example (extra notes on board)

Double-Precision Checksum (DPC)

- Compute the checksum in double precision
- Compute a $2n$-bit checksum for a block of n-bit words using modulo-2^{2n} arithmetic
- Can overcome the difficulty of SPC
- An example (extra notes)

Honeywell Checksum

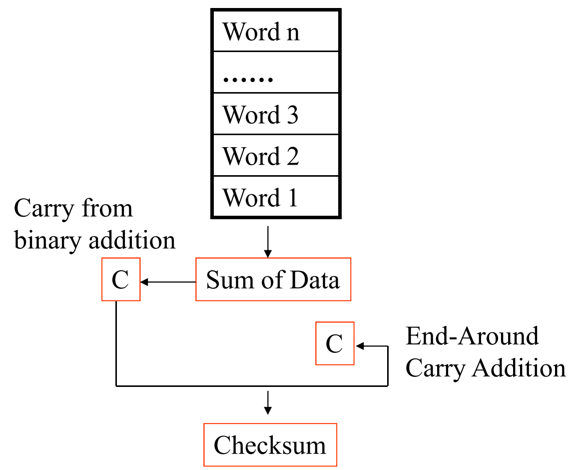
- Concatenate consecutive words to form a collection of double length words
- Assume there are n k-bit words, a set of $n/2$ $2k$-bit words is formed, and a checksum is formed over the newly double-length words → a bit error appearing in the same bit positions of all words will affect at least two bit positions of the checksum!



- An example (extra notes)

Residue Checksum

- Same basic concept as in SPC except that the carry bit out of the MSB position is not ignored but is added back to the checksum in an end-around carry fashion



- An example (extra notes)

Agenda

✓ Basic concepts

- Example codes
  - √ Parity codes: Hamming SEC code, Horizontal and Vertical Parity code
  - √ m-of-n
  - √ Berger
  - √ Checksums
  - – **Cyclic**
  - – Arithmetic
- Code selection issue

Cyclic Codes (1)

- Every cyclic/end-around shift of a code word is also a code word
- Often used in sequential devices (disks, tapes)
- Cyclic codes are best represented and analyzed through the use of **polynomial algebra**
- Each bit of the code word *v* can be represented as a coefficient of the polynomial *V(X)*

$$v=(v_0, v_1, \ldots, v_{n-1})$$

$$V(X) = v_0 + v_1X + v_2X^2 + \ldots + v_{n-1}X^{n-1}$$

Cyclic Codes (2)

- Generator Polynomial **-** The set of code words for an (n , k) cyclic code is uniquely generated by its generator polynomial G(X)
- Example: **nonseparable cyclic code**

$$\text{Code Polynomial} = V(X) = D(X)G(X)$$

$$D(X) = \text{Data Polynomial} = d_{k-1}X^{k-1} + \ldots + d_1X + d_0$$

$$V(X) = \text{Code Polynomial} = v_{n-1}X^{n-1} + \ldots + v_1X + v_0$$

$$n = \text{Number of bits in the code word}$$

$$k = \text{Number of bits in the original data word}$$

$$G(X) = g_{n-k}X^{n-k} + g_{n-k-1}X^{n-k-1} + \ldots + g_2X^2 + g_1X + g_0$$

Modulo-2 operation!

G(X)	(n , k)
$1 + X + X^3$	(3 + K, K)
$1 + X + X^2 + X^3 + X^{11} + X^{12}$	(12 + K, K)
$1 + X^2 + X^{13} + X^{16}$	(16 + K, K)
$1 + X^5 + X^{12} + X^{16}$	(16 + K, K)

Cyclic Codes (3)

- The generator polynomial G(X) has the following properties
  - The degree of G(X) is n-k
  - Each code word V(X) is a multiple of G(X) and is computed as V(X) = D(X)G(X)
  - G(X) must be a factor of $X^n - 1$
- The (n,k) cyclic codes can detect all **single-bit errors** and all **multiple, adjacent errors affecting fewer than (n-k) bits**
  - Communication applications with burst errors
  - A burst error is the result of a transient fault and usually introduces a number of adjacent errors into a given data item
  - (n,k) cyclic codes can detect adjacent errors as long as number of adjacent bits affected <= (n-k)
- Non-separable and separable

Non-separable Cyclic Codes

- Encoding process: simply multiplying the data polynomial D(X) by the generator polynomial G(X)

- Implementation using **combinatorial circuits**
  - For binary codes the coefficients of each polynomial (generator, data, and code) are either 0 (zero) or 1 (one)
  - Consequently, the coefficients of the code polynomial are simply summations of the appropriate data bits
  - The summations are modulo 2 summations which are equivalent to EXCLUSIVE-OR gates

Non-separable Cyclic Coding Using Combinatorial Circuits

- The encoding process for a particular generator polynomial can be performed using a combinational circuit containing only EXCLUSIVE-OR gates

Example: (7, 4) Cyclic Code

- The generator polynomial is

$$G(X) = 1 + X + X^3$$ ➡ $$g_0 = 1, \quad g_1 = 1, \quad g_2 = 0, \quad g_3 = 1$$

- Also, assume that the data polynomial is given by

$$D(X) = d_0 + d_1X + d_2X^2 + d_3X^3$$

- The resulting code polynomial is given by

$$V(X) = d_0 + (d_0 + d_1)X + (d_1 + d_2)X^2 + (d_0 + d_2 + d_3)X^3 + (d_1 + d_3)X^4 + d_2X^5 + d_3X^6$$

- The code word is given by

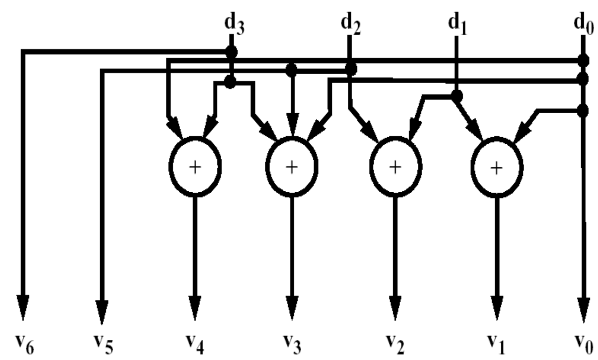
$$v = (v_0, v_1, v_2, v_3, v_4, v_5, v_6)$$

$$v = (d_0, (d_0 + d_1), (d_1 + d_2), (d_0 + d_2 + d_3), (d_1 + d_3), d_2, d_3)$$

Combinatorial Circuits for the Example

- Combinatorial circuit composed of modulo-2 adders (exclusive-OR gates) for the example (7, 4) cyclic code with

$$G(X) = 1 + X + X^3$$



Checking of Non-separable Cyclic Codes

- Checking of cyclic codes:

$$(r_0, r_1, r_2, \ldots, r_{n-1})$$

- It can be represented by the polynomial

$$R(X) = r_0 + r_1X + r_2X^2 + \ldots + r_{n-1}X^{n-1}$$

- If it is valid, then ***R(X)=D(X)*G(X)***

- In general, we write

$$R(X) = D(X)G(X) + S(X)$$

  - *S(X):* the remainder of the division *R(X)/G(X),* called *syndrome polynomial*
  - *R(X)* is valid if *S(X)=0*

Exercise (1)

- Check if the code word (1111111) is a valid (7,4) cyclic code word or not? Assume the generator polynomial is $G(X)=1+X^2+X^3$

- If valid, what is the corresponding original data word?

Exercise (2)

- Assume the generator polynomial is $G(X)=1+X+X^3$.
  - Generate non-separable (7,4) cyclic code word for data word (d0d1d2d3)=(1 1 1 1).
  - Check if the code word (v0v1v2v3v4v5v6)=(1001111) is a valid (7,4) cyclic code word or not?

Separable Cyclic Codes

- To generate a separable (n,k) code, the original data polynomial D(X) is first multiplied by X^{n-k} and result is divided by G(X) to obtain a remainder R(X):

$$R(X) = r_{n-k-1}X^{n-k-1} + r_{n-k-2}X^{n-k-2} + \ldots + r_1X + r_0$$

$$X^{n-k}D(X) = Q(X)G(X) + R(X)$$

$$Q(X) = \text{Quotient Polynomial}$$

$$R(X) = \text{Remainder Polynomial}$$

$$G(X) = \text{Generator Polynomial}$$

$$D(X) = \text{Data Polynomial}$$

$$V(X) = X^{n-k}D(X) + R(X) = Q(X)G(X)$$

$$= d_{k-1}X^{n-1} + d_{k-2}X^{n-2} + \ldots + d_1X^{n-k+1} + d_0X^{n-k}$$

$$+ r_{n-k-1}X^{n-k-1} + \ldots + r_1X + r_0$$

- The code word v is given by the coefficients of the code polynomial which are

$$v = (r_0, r_1, \ldots, r_{n-k-1}, d_0, d_1, \ldots, d_{k-2}, d_{k-1})$$

Exercise (3)

- Construct a separate (7,4) cycle code with data d0d1d2d3=1001 and $G(X)=1+X+X^3$.

Agenda

✓ Basic concepts
- Example codes
  - √ Parity codes: Hamming SEC code, Horizontal and Vertical Parity code
  - √ m-of-n
  - √ Berger
  - √ Checksums
  - √ Cyclic
  - – **Arithmetic**
- Code selection issue

Arithmetic Codes

- Useful in checking arithmetic operations
- **Basic concept:** data presented to arithmetic operation is encoded before operations; resulting code words are checked after completing arithmetic operations. If not valid, an error condition is detected
- Must be **invariant** to a set of arithmetic operations

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

- Examples
  - AN codes
  - Residue codes
  - Inverse residue codes

AN Codes (1)

- Formed by multiplying data word **N** by some constant **A**
- Invariant to addition and subtraction, but not multiplication and division
- The magnitude of **A** determines
  - Number of extra bits for code word
  - The error detection capability
- An example: **3**N code → all words encoded by multiplying by 3 (Table 3.11)
  - (n+2) bits are required for 3N code of n-bit data words

AN Codes (2)

- For binary codes, **A** must not be a power of 2 because an AN code with $A=2^a$ cannot detect any single-bit errors.
  - Multiplication by 2^a is equivalent to a left arithmetic shift of binary data word
  - Changing any data bit still yields a result that is evenly divisible by 2^a → a valid AN code → the error remains undetected!

Residue Codes

- Formed by appending the **residue** (r) of a data word to the original data word (N): N|r
- A separable code
- **Residue** of a number is simply the remainder generated when the number is divided by an integer *m*:

  $N = Im + r$ or $\frac{N}{m} = I + \frac{r}{m}$

  - m: check base or modulus
  - I: quotient
  - r: remainder or residue

- An example: separable residue code words for 4-bit data words using a modulus of 3 (Table 3.12)

Inverse-Residue Codes

- Formed by appending the **inverse-residue** q of a data word N to that data word: N|q
- **Inverse residue** q of a number is simply *m-r*, where *r* is the remainder generated when the number is divided by an integer *m*:
- A separable code

- An example: separable inverse-residue code words for 4-bit data words using a modulus of 3 (Table 3.13)

Review of Codes

- Parity codes
  - Single-bit parity codes
  - Multiple-bit parity codes (Hamming single error correcting codes)
  - Horizontal and vertical parity codes
- m-of-n codes (separable/non-separable)
- Berger codes (separable)
- Checksum (separable)
  - SPC, DPC, Honeywell, Residue
- Cyclic codes (separable/non-separable)
- Arithmetic codes
  - AN, residue, inverse-residue

Code Selection Issue

- The key: select a code that fulfills the desired error detection/correction capability while maintaining costs at an acceptable level
  - Information redundancy involves other forms of redundancy (time: encoding/decoding process; hardware redundancy: additional storage for extra bits)
- Three major factors / decisions
  - Whether or not the code needs to be separable
  - Whether error detection, error correction, or both are required
  - Number of bit errors needs to be detected or corrected

Summary of Lecture #6

- m-of-n codes (separable/non-separable) can detect all single-bit errors and all multiple, unidirectional errors
- Berger codes are separable unidirectional error detecting codes; which can be manipulated so that they are invariant to the arithmetic/logical operations
- Checksum (SPC/DPC/Honeywell/Residue) codes are separable codes and can only detect errors but not locate/correct errors
- Cyclic codes are invariant to the end-around shift operation; are best represented and analyzed using polynomial algebra; can be separable and non-separable
- AN codes are invariant to addition and subtraction, but not multiplication and division
- Both residue and inverse-residue codes are separable codes

Things to Do

- Homework

- Class Project
  - Proposal due **Wednesday Oct. 5**

Next topic:

Time redundancy & Software redundancy!