

UNIVERSITY OF MASSACHUSETTS
Dept. of Electrical & Computer Engineering

Fault Tolerant Computing
ECE 655

Part 5
Coding - II

Israel Koren
Fall 2006

ECE655/Koren Part.5 .1

Copyright 2006 Koren & Krishna

Cyclic Codes

- ◆ Cyclic codes are often **non-separable** although separable cyclic codes exist
- ◆ Encoding consists of multiplying (**modulo-2**) the data word by a constant number
- ◆ The coded word is the product
- ◆ Decoding is dividing by the same constant - if the remainder is non-zero, an error has occurred
- ◆ Cyclic codes are widely used in data storage and communication
- ◆ Called cyclic since - if $a_{n-1}, a_{n-2}, \dots, a_0$ is a codeword, so is its cyclic shift $a_0, a_{n-1}, a_{n-2}, \dots, a_1$
- ◆ Example: A 5-bit cyclic code

{00000, 00011, 00110, 01100, 11000, 10001, 00101, 01010,
10100, 01001, 10010, 01111, 11110, 11101, 11011, 10111}

ECE655/Koren Part.5 .2

Copyright 2006 Koren & Krishna

Cyclic Codes - Theory

- ◆ k - number of bits of data that are encoded
- ◆ Encoded word of length n bits - obtained by multiplying the given k data bits by a number that is $n-k+1$ bits long
- ◆ The multiplier is represented as a polynomial - **the generator polynomial**
- ◆ 1s and 0s in the $n-k+1$ -bit multiplier are treated as coefficients of an $(n-k)$ -degree polynomial
- ◆ **Example:** multiplier is **11001** - generator polynomial is

$$\begin{aligned}
 \text{◆ } G(x) &= 1 X^0 + 0 X^1 + 0 X^2 + 1 X^3 + 1 X^4 \\
 &= 1 + X^3 + X^4
 \end{aligned}$$

ECE655/Koren Part.5 .3

Copyright 2006 Koren & Krishna

An (n,k) Cyclic Code

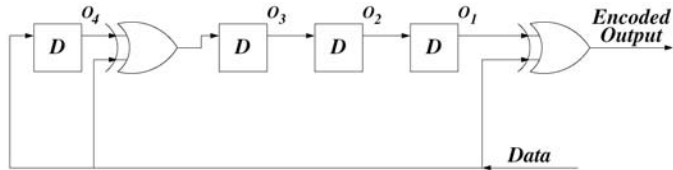
- ◆ Using a generator polynomial of degree $n-k$ and total number of encoded bits n
 - ◆ An (n,k) cyclic code can detect all single errors and all runs of adjacent bit errors shorter than $n-k$
 - ◆ Useful in applications like wireless - channels are frequently noisy and have bursts of interference resulting in runs of adjacent bit errors
 - ◆ For a polynomial to be a generator polynomial for an (n,k) cyclic code it must be a factor of $X^n - 1$
 - ◆ $1 + X^3 + X^4$ is a factor of $X^{15} - 1 \Rightarrow (15,11)$ code
- $$X^{15} - 1 = (X + 1)(X^2 + X + 1)(X^4 + X + 1)(X^4 + X^3 + 1)(X^4 + X^3 + X^2 + X + 1)$$
- ◆ For the 5-bit code, $(X+1)$ - generator polynomial
- $$X^5 - 1 = (X + 1)(X^4 + X^3 + X^2 + X + 1)$$
- ◆ Multiply {0000, ..., 1111} by $(X+1)$ to obtain all codewords of the $(5,4)$ code

ECE655/Koren Part.5 .4

Copyright 2006 Koren & Krishna

Hardware Implementation

- ◆ Multiplication can be implemented by **shift registers** and **exclusive-or gates**
- ◆ **Example:** generator polynomial $1 + X^3 + X^4$
(corresponding to the multiplier **11001**)
- ◆ **Encoding circuit:**



- ◆ Square boxes are delay elements which hold their input for one clock cycle

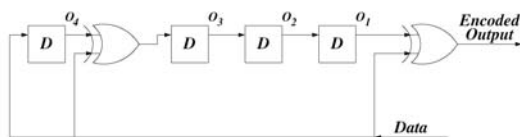
ECE655/Koren Part.5 .5

Copyright 2006 Koren & Krishna

Modulo-2 Multiplication

- ◆ The **5th** bit of the product is the **modulo-2** sum of the corresponding bits of the multiplicand shifted **0** times, **3** times, and **4** times
- ◆ If multiplicand is fed in serially - we add the shifted multiplicand
- ◆ Shifting done by delay elements
- ◆ This cyclic code is **not separable** - data and check bits within **110000100011101** are not separable

$$\begin{array}{r}
 10001100101 \\
 \times \quad 11001 \\
 \hline
 10001100101 \\
 00000000000 \\
 00000000000 \\
 10001100101 \\
 10001100101 \\
 \hline
 110000100011101
 \end{array}$$



ECE655/Koren Part.5 .6

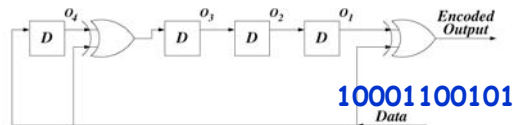
Copyright 2006 Koren & Krishna

Operation of Encoding Circuit

shift clock	input data	O ₄	i ₃	O ₃ , O ₂ , O ₁	encoded output
1	1	0	1	000	1
2	0	1	1	100	0
3	1	0	1	110	1
4	0	1	1	111	1
5	0	0	0	111	1
6	1	0	1	011	0
7	1	1	0	101	0
8	0	1	1	010	0
9	0	0	0	101	1
10	0	0	0	010	0
11	1	0	1	001	0
12	0	1	1	100	0
13	0	0	0	110	0
14	0	0	0	011	1
15	0	0	0	001	1

1100001000111101

◆ i₃ is the input to the O₃ delay element



10001100101
Data

ECE655/Koren Part.5 .7

Copyright 2006 Koren & Krishna

Decoding Through Division by Generator Polynomial

1100001000111101 : 11001 = 10001100101 110000100111101 : 11001 = 10001100110

```

11001
 $\overline{10100}$ 
  11001
   $\overline{11010}$ 
    11001
     $\overline{11111}$ 
      11001
       $\overline{11001}$ 
        11001
         $\overline{11001}$ 
          00000
    
```

```

11001
 $\overline{10100}$ 
  11001
   $\overline{11011}$ 
    11001
     $\overline{10111}$ 
      11001
       $\overline{11100}$ 
        11001
         $\overline{11001}$ 
          01011
    
```

- ◆ Division by 11001
- ◆ Subtraction modulo-2 - identical to addition
- ◆ Zero remainder indicates no error detected
- ◆ If a single error occurs - 110000100111101 - non-zero remainder

ECE655/Koren Part.5 .8

Copyright 2006 Koren & Krishna

Three Bit Errors

$ \begin{array}{r} 110000111010101 : 11001 = 10001101101 \\ \underline{11001} \\ 10111 \\ \underline{11001} \\ 11100 \\ \underline{11001} \\ 10110 \\ \underline{11001} \\ 11111 \\ \underline{11001} \\ 11001 \\ \underline{11001} \\ 00000 \end{array} $	$ \begin{array}{r} 110000011011101 : 11001 = 10001110011 \\ \underline{11001} \\ 10011 \\ \underline{11001} \\ 10100 \\ \underline{11001} \\ 11011 \\ \underline{11001} \\ 10110 \\ \underline{11001} \\ 11111 \\ \underline{11001} \\ 00110 \end{array} $
--	--

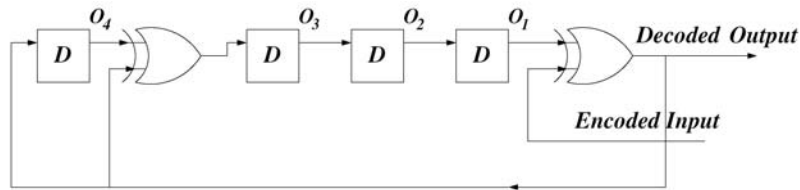
- ◆ **110000111010101** instead of **1100000100011101**
 - zero remainder - errors not detected (erroneously)
- ◆ If errors are adjacent - **110000011011101** -
 non-zero remainder indicates an error

Implementing a Divider Circuit

- ◆ Division can be done by multiplication in the feedback loop
- ◆ **Example:** Encoded word - polynomial $E(X)$, generator polynomial - $G(X)$, original data word - polynomial $D(X)$
- ◆ If no bit errors exist - we will receive $E(X)$ and calculate $D(X)$ by $D(X)=E(X)/G(X)$ - remainder will be zero
- ◆ **Example:** $E(X)=D(X)G(X)=D(X)(1+X^3+X^4)$

$$=D(X)+D(X)(X^3+X^4)$$
- ◆ $D(X)=-E(X)+D(X)(X^3+X^4)=E(X)+D(X)(X^3+X^4)$
 * (since addition = subtraction in mod-2 arithmetic)

Divider



- ◆ Feedback circuit for division $D(X) = E(X) + D(X)(X^3 + X^4)$
- ◆ Start with all delay elements holding 0 and produce first the **seven** quotient bits (data bits), and then the **four** remainder bits
- ◆ If remainder bits are **nonzero**, an error has occurred

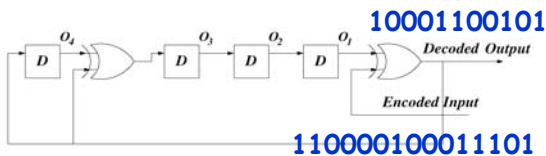
ECE655/Koren Part.5 .11

Copyright 2006 Koren & Krishna

Operation of Divider

- ◆ i_3 - input to the O_3 delay element = $i_4 \oplus O_4$

shift clock	encoded input	i_4	O_4	i_3	O_3, O_2, O_1	decoded output
1	1	1	0	1	000	1
2	0	0	1	1	100	0
3	1	1	0	1	110	1
4	1	0	1	1	111	0
5	1	0	0	0	111	0
6	0	1	0	1	011	1
7	0	1	1	0	101	1
8	0	0	1	1	010	0
9	1	0	0	0	101	0
10	0	0	0	0	010	0
11	0	1	0	1	001	1
12	0	0	1	1	100	0
13	0	0	0	0	110	0
14	1	0	0	0	011	0
15	1	0	0	0	001	0



- ◆ Any error in received sequence $E(X)$ will result in a **non-zero** remainder

ECE655/Koren Part.5 .12

Copyright 2006 Koren & Krishna

Detecting Bursty Errors

- ◆ Many applications need to make sure that all burst errors of length **16 bits** or less will be detected
- ◆ Cyclic codes of the type **(16+k,k)** are used
- ◆ The generating polynomial should be selected to allow a large number of data bits (use same circuit for different sizes of data blocks)
- ◆ Most commonly used :
 - * **CRC-16** (16-bit Cyclic Redundancy Code)

$$G(X) = X^{16} + X^{15} + X^2 + 1$$

- * **CRC-CCITT**

$$G(X) = X^{16} + X^{12} + X^5 + 1$$

Both divide $X^n - 1$ for $n = 2^{15} - 1 = 32,767$ bits

Separable Cyclic Codes

- ◆ Allow use of data before encoding completed
- ◆ Data word $D(X) = d_{k-1}X^{k-1} + d_{k-2}X^{k-2} + \dots + d_0$
- ◆ Append **(n-k)** zeroes to $D(X)$ to obtain $\bar{D}(X) = d_{k-1}X^{n-1} + d_{k-2}X^{n-2} + \dots + d_0X^{n-k}$
- ◆ Divide by $G(X)$: $\bar{D}(X) = Q(X)G(X) + R(X)$, degree of $R(X) < n-k$
- ◆ Codeword $C(X) = \bar{D}(X) - R(X)$ has $G(X)$ as a factor
- ◆ Divide $C(X)$ by $G(X)$ - if non-zero \Rightarrow error
- ◆ In $C(X)$: 1st k bits data, last $n-k$ check bits
- ◆ Example: **(5,4)** code with $G(X) = X+1$: for data 0110 we get $\bar{D}(X) = X^3 + X^2 = (X+1)X^2 + 0$
 - * Codeword 01100
 - * Same codewords generated but different correspondence

Arithmetic Codes

- ◆ Codes that are preserved under a set of arithmetic operations
- ◆ Enabling detection of errors which may occur during execution of arithmetic operations
- ◆ Such concurrent error detection can always be attained by duplicating the arithmetic processor
 - * This however, is too costly
- ◆ An error code is preserved under an arithmetic operation * if for any two operands X and Y and the corresponding encoded entities X' and Y' there is an operation \otimes satisfying
$$X' \otimes Y' = (X * Y)'$$
 - * The result of \otimes when applied to the encoded X' and Y' will yield the same as encoding the outcome of the original operation * to the original operands X and Y

Error Detection

- ◆ We expect the arithmetic codes to be able to detect all **single-bit** faults
- ◆ A single bit error in an operand or an intermediate result may cause a multiple-bit error in the final result
- ◆ **Example** - when adding two binary numbers, if stage i of the adder is faulty, all the remaining $n-i$ higher order digits may be erroneous

Non-Separable Arithmetic Codes

- ◆ Simplest - AN -codes - formed by multiplying the operands by a constant A
- ◆ $X' = A \cdot X$ and the operations $*$ and \otimes are identical for add/subtract
- ◆ Example - $A=3$
 - * Each operand is multiplied by 3 (obtained as $2X+X$)
 - * The result of the operation is checked to see whether it is an integer multiple of 3
- ◆ All error magnitudes that are multiples of A will not be detected

AN Codes

- ◆ A should not be a power of the radix 2
- ◆ An odd A is best - it will detect every single bit fault - such an error has a magnitude of 2^i
- ◆ $A=3$ - least expensive AN -code that enables detection of all single bit errors
- ◆ Example - the number $0110_2 = 6_{10}$
- ◆ Representation in the AN -code with $A=3$ is $010010_2 = 18_{10}$
- ◆ A fault in bit position 2^3 may give the erroneous result $011010_2 = 26_{10}$
- ◆ The error is easily detectable - 26 is not a multiple of 3

Separable Arithmetic Codes

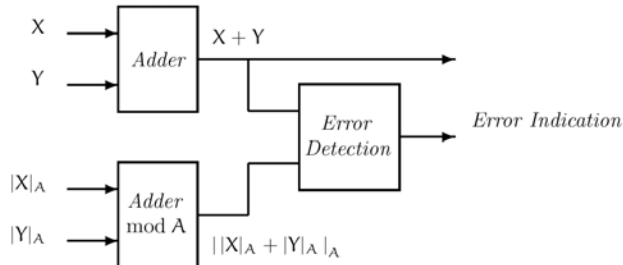
- ◆ Simplest - residue code and inverse residue code
- ◆ In each of these we attach a separate check symbol $C(X)$ to every operand X
- ◆ For the residue code, $C(X)=X \bmod A = |X|_A$
- A is called the check modulus
- ◆ For the inverse residue code, $C(X)=A-(X \bmod A)$
- ◆ For both codes $C(X) \otimes C(Y) = C(X * Y)$ where $*$ equals \otimes and they are either addition or multiplication
- ◆ $|X+Y|_A = ||X|_A + |Y|_A|_A$
- ◆ $|X \cdot Y|_A = ||X|_A \cdot |Y|_A|_A$
- ◆ Division: $X-S=Q \cdot D$ - X is the dividend, D is the divisor, Q is the quotient, S is the remainder
- ◆ The check is: $||X|_A - |S|_A|_A = ||Q|_A \cdot |D|_A|_A$

Examples

- ◆ $A=3, X=7, Y=5$ - the residues are:
 $|X|_3=1$ and $|Y|_3=2$; $|7+5|_3=0 = ||7|_3+|5|_3|_3$
 $=|1+2|_3=0$
- ◆ $|7 \cdot 5|_3=2=||7|_3 \cdot |5|_3|_3=|1 \cdot 2|_3=2$
- ◆ $A=3, X=7$ and $D=5$ - $Q=1$ and $S=2$ - the residue check is:
 $||7|_3 - |2|_3|_3 = ||5|_3 \cdot |1|_3|_3=2$
- ◆ Subtraction is done by adding the complement to the modulus:
 $|1-2|_3 = |1+|3-2|_3|_3 = |1+1|_3=2$

Residue mod A vs. AN Code

- ◆ Both have the same undetectable errors
 - * Example: $A=3$ - only errors that modify the result by a multiple of 3 will not be detected
 - * single-bit errors are always detectable
- ◆ Same checking algorithm for both
 - * Compute the residue modulo A of the result
- ◆ Same increase in word length - $\lceil \log_2 A \rceil$
- ◆ Most important difference - separability
 - * The unit for $C(X)$ in the residue code is separable
 - * Single unit for the AN code



ECE655/Koren Part.5 .21

Low Cost Arithmetic Codes

- ◆ The AN and residue codes with $A=3$ are the simplest examples of a class of arithmetic codes which use an A of the form $A=2^a-1$ (a integer)
- ◆ This simplifies the calculation of remainder when dividing by A (checking algorithm)
- ◆ Calculating the remainder is simple since
- ◆ $|z_i r^i|_{r-1} = |z_i|_{r-1} ; r=2^a$
- ◆ Allows the use of modulo- (2^a-1) summation of the groups of size a bits that compose the number (each group has a value $0 \leq z_i \leq 2^a-1$)

ECE655/Koren Part.5 .22

Copyright 2006 Koren & Krishna

Example

◆ Remainder when dividing $X=11110101011$ by $A=7=2^3-1$

- * Partition X into groups of size 3, starting with the least significant bit
- * This yields $X=(Z_3, Z_2, Z_1, Z_0)=(11, 110, 101, 011)$
- * Add these groups modulo 7: "cast out" 7's and add the end-around-carry when necessary
- * Weight of carry-out is 8 & $|8|_7=1$:
- * Add end-around-carry
- * Residue mod 7 of X is 3
- * Correct remainder of $X=1963_{10}$ divided by 7

$$\begin{array}{r}
 11 \quad z_3 \\
 + 110 \quad z_2 \\
 \hline
 1 \quad 001 \\
 + 1 \quad \text{end-around carry} \\
 \hline
 010 \\
 + 101 \quad z_1 \\
 \hline
 111 \\
 + 011 \quad z_0 \\
 \hline
 1 \quad 010 \\
 + 1 \quad \text{end-around carry} \\
 \hline
 + 011
 \end{array}$$

$$|z_i r^i|_{r-1} = |z_i|_{r-1} ; r=2^a$$

ECE655/Koren Part.5 .23

Copyright 2006 Koren & Krishna

Arithmetic Codes with Signed Operands

- ◆ Code must be complementable with respect to R
 - * $R = 2^n$ (two's complement)
 - * or $R = 2^n - 1$ (one's complement)
 - * n - number of bits in the encoded operand
- ◆ For the AN code, $R-AX$ must be divisible by A - A must be a factor of R
- ◆ If we insist on an odd A - $R=2^n$ is excluded
- ◆ Odd A - only one's complement can be used - A must be a factor of $2^n - 1$

ECE655/Koren Part.5 .24

Copyright 2006 Koren & Krishna

Example - AN code

- ◆ $n=4$, $R=2^n-1=15$ for **one's complement** - divisible by A for the AN code with $A=3$
- ◆ $X=0110$ is represented by $3X=010010$ - **one's complement** is $101101 = 45_{10}$ - divisible by 3
- ◆ The **two's complement** of $3X$ is $101110=46_{10}$ - not divisible by 3
- ◆ If $n=5$ - **one's complement** - $R=31$ - not divisible by $A=3$
- ◆ $X=00110$ - represented by $3X=0010010$ - **one's complement** is $1101101=109_{10}$ - not divisible by 3

Residue Code with Signed Operands

- ◆ $A - |X|_A = |R - X|_A$ must be satisfied
- ◆ R must be an integer multiple of A - again allowing only **one's complement** arithmetic
- ◆ Modifying the procedure so that **two's complement** (with $R=2^n$) can also be used -
 - * $|2^n - X|_A = |2^n - 1 - X + 1|_A = |2^n - 1 - X|_A + |1|_A$
 - * Need to add a correction term $|1|_A$ to the residue code when forming the **two's complement**
 - * A must still be a factor of $2^n - 1$

Example - Residue Code

- ◆ $A=7, n=6, R=2^6=64$ for two's complement - $R-1=63$ is divisible by 7
- ◆ $001010_2=10_{10}$ has the residue 3 mod 7
- ◆ The two's complement of 001010 is 110110
- ◆ The complement of $|3|_7$ is $|4|_7$ and adding the correction term $|1|_7$ yields 5 - the correct residue mod 7 of $110110=54_{10}$
- ◆ Similar correction needed when adding two's complement operands and a carry-out (of weight 2^n) is generated and discarded
- ◆ To compensate, subtract $|2^n|_A$ from the residue check
- ◆ Since A is a factor of $2^n-1, |2^n|_A = |1|_A$

Interdependence Between Main and Check Units

- ◆ In this two's complement addition - a carry-out is generated and discarded

$$\begin{array}{r}
 110110=X \\
 + \quad 001101=Y \\
 \hline
 1 \quad 000011
 \end{array}
 \quad
 \begin{array}{r}
 101=|X|_7 \\
 + \quad 110=|Y|_7 \\
 \hline
 1 \quad 011 \\
 \quad \quad \quad 1 \text{ end-around carry} \\
 \quad \quad \quad \hline
 \quad \quad \quad 100 \\
 \quad \quad \quad - \quad 1 \text{ correction term} \\
 \quad \quad \quad \hline
 \quad \quad \quad 011
 \end{array}$$

- ◆ This results in an interdependence between the main and check units
- ◆ An error in the main unit may propagate to the check unit and the effect of the fault is masked
- ◆ A single-bit error is always detectable

Bi-Residue Code

- ◆ Error correction can be achieved by using two or more residue checks
- ◆ Simplest case - **bi-residue code**
- ◆ Consists of two residue checks A_1 and A_2
- ◆ $A_1=2^{a-1}$ and $A_2=2^{b-1}$ are two low-cost residue checks with $n=l.c.m.(a,b)$
 - * n is the number of bits in the operands
- ◆ Any single-bit error can be **corrected**