**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 \( a_{n-1}, a_{n-2}, \ldots, a_0 \) is a codeword, so is its cyclic shift \( a_0, a_{n-1}, a_{n-2}, \ldots, a_1 \).
- Example: A 5-bit cyclic code:
  \( \{00000, 00011, 00110, 01100, 11000, 10001, 00101, 01010, 10100, 01001, 10010, 01111, 11110, 11101, 11011, 10111\} \)
**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

\[
G(X) = 1 \cdot X^0 + 0 \cdot X^1 + 0 \cdot X^2 + 1 \cdot X^3 + 1 \cdot X^4 \\
= 1 + X^3 + X^4
\]

**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 no longer than **n**-**k**
- Useful in applications like wireless communication - 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 \(X^{15} - 1\) ⇒ (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**
**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:

  ![Encoding Circuit Diagram]

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

---

**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 11000100011101 are not separable

![Modulo-2 Multiplication Diagram]

\[
\begin{array}{c}
10001100101 \\
\times \quad 11001 \\
\hline
10001100101 \\
00000000000 \\
00000000000 \\
10001100101 \\
10001100101 \\
110000100011101
\end{array}
\]
Operation of Encoding Circuit

<table>
<thead>
<tr>
<th>shift clock</th>
<th>input data</th>
<th>O₄</th>
<th>i₃</th>
<th>O₂, O₁, O₀</th>
<th>encoded output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>000</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>110</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>111</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>111</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>011</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>101</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>010</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>101</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>010</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>001</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>110</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>011</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>001</td>
<td>1</td>
</tr>
</tbody>
</table>

♦ i₃ is the input to the O₃ delay element

Decoding Through Division by Generator Polynomial

Division by 11001
- Subtraction modulo-2 - identical to addition
- Zero remainder indicates no error detected
- If a single error occurs - 11000010011101 - non-zero remainder
Three Bit Errors

1100001 110011101 11001

1100001 110011101 11001

110001 11001

11001

11001

11001

11001

11001

00000

- 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
- Notation: 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: \( E(X) = D(X) \cdot G(X) = D(X)(1 + X^3 + X^4) \)
  \[ D(X) = E(X) - D(X)(X^3 + X^4) = E(X) + D(X)(X^3 + X^4) \]
  \* (since addition = subtraction in mod-2 arithmetic)
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.

Operation of Divider

\[ i_3 \text{ - input to the } O_3 \text{ delay element} = i_4 \oplus O_4 \]

Any error in received sequence \( E(X) \) will result in a non-zero remainder.
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 \( 2^{15} - 1 = 32,767 \) bits

Separable Cyclic Codes

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

♦ Codes that are preserved under a set of arithmetic operations
♦ Enable detection of errors occurring during execution of arithmetic operations
♦ Error detection can be attained by duplicating the arithmetic processor - too costly
♦ A 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 ⊗ satisfying X’ ⊗ Y’ = (X * Y)’
  • The result of ⊗ 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

♦ Arithmetic codes should be able to detect all single-bit errors
♦ 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 $A$
- $X' = AX$ and the operations $\ast$ 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
♦ We attach a separate check symbol \( C(X) \) to every operand \( X \)
♦ For the residue code, \( C(X) = X \mod A = |X|_A \)
  \* \( A \) is called the check modulus
♦ For the inverse residue code, \( C(X) = A - (X \mod A) \)
♦ For both codes \( C(X) \odot C(Y) = C(X \times Y) \)
  \* \( \odot \) equals \* - either addition or multiplication
♦ \( |X+Y|_A = ||X|_A + |Y|_A|_A \)
♦ \( |X \times Y|_A = ||X|_A \times |Y|_A|_A \)
♦ Division: \( X - S = Q \odot D \) - \( X \) is the dividend, \( D \) is the divisor, \( Q \) is the quotient, \( S \) is the remainder
♦ The check is: \( ||X|_A - |S|_A|_A = ||Q|_A \odot |D|_A|_A \)

Examples

♦ \( A=3 \), \( X=7 \), \( Y=5 \) - the residues are:
  \( |X|_3 = 1 \) and \( |Y|_3 = 2 \); \( |7+5|_3 = 0 = ||7|_3 + |5|_3|_3 = |1+2|_3 = 0 \)
♦ \( |7 \times 5|_3 = 2 = ||7|_3 \times |5|_3 = |1 \times 2|_3 = 2 \)
♦ \( A=3 \), \( X=7 \) and \( D=5 \) - \( Q=1 \) and \( S=2 \) - the residue check is:
  \( ||7|_3 - |2|_3|_3 = ||5|_3 \odot |1|_3|_3 = 2 \)
♦ 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

♦ 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
  * Compute the residue modulo A of the result

♦ Same increase in word length - |log₂A|

♦ Most important difference - separability
  * The unit for C(X) in the residue code is separable
  * Single unit for the AN code

Low Cost Arithmetic Codes

♦ AN and residue codes with A=3 - simplest examples of a class of arithmetic codes with A = 2ᵃ − 1 (a integer)

♦ Simplifies the calculation of remainder when dividing by A (checking algorithm)

♦ Calculating the remainder is simple since
  ♦ |Zᵢ⁺ᵢ |ᵢ−₁ = |Zᵢ |ᵢ−₁ ; r = 2ᵃ

♦ Allows the use of modulo-(2ᵃ − 1) summation of the groups of size a bits that compose the number (each group has a value 0 ≤ Zᵢ ≤ 2ᵃ − 1)
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

$$|Z_ir^j |_{r-1} = |Z_i |_{r-1}$$

$r = 2^a$

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 $2^n - 1$
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 -
  * 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, \( R = 2^n = 64 \) for two's complement - \( R-1=63 \) is divisible by 7
♦ 001010₂ =10₁₀ 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 \( 110110=54₁₀ \)
♦ 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

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

♦ 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 $n = \text{l.c.m.}(a, b)$
  - *n* is the number of bits in the operands
- Any single-bit error can be corrected