DIGITAL MATHEMATICS APPLIED IN DEFENCE AND SECURITY EDUCATION (DIMAS)

KA220-HED - Cooperation Partnerships in Higher Education

2023-1-BG01-KA220-HED-000156664

Educational resource

(A3.2)

May 2025

SCENARIO 1.3

Title	**ERROR CORRECTION CODING WITH HAMMING CODES** Parity check
TASK 1: Perform parity calculation over binary codewords. **Knowledge:** Digital encoding - representation of information, binary digits. **Mathematics areas:** Algebra; Boolean Algebra. **Abilities:** Describe binary codewords for digital information, calculate Boolean operations.	The simplest example of error check in a given codeword is to perform parity check. With such a check, the number of *1*ˢ is count until getting an even number. This is a code in which the parity check is performed for each code word (code group), as a result of which the number of *1*ˢ is completed to an even number. To the codeword (group) of the information code, a check symbol is appended on the right in the form of a ***1*** if the number of *1*s in the information code word was odd, or in the form of a ***0***, if the number of *1*s in the information code word was even. Thus, the total number of units in each allowed code combination must be even. This code is dividing and systematic, since **the check bit value is a sum modulo 2** of the information symbols in the code combination. It has a code distance (a.k.a Hamming distance) α=2 and detects all errors with an odd multiplicity. Example of a code distance:

2-bit information codeword (*m*=2)	**code distance [in bits]**	**5-bit information codeword** (*m*=5)	**code distance [in bits]**
0 0	1	0 1 1 1 0	2
1 0		0 1 **0** **0** 0	

Parity check:

5-bit information codeword (*m*=5)					**number of 1ˢ inside**	**with appended check-bit (*n*=6)**					
0	1	0	0	0	3 (odd)	0	1	1	1	0	**1**
0	1	1	0	0	2 (even)	0	1	1	0	0	**0**
0	1	1	1	0	1 (odd)	0	1	0	0	0	**1**

TASK 2:

Perform parity calculation over binary codewords with summation modulo 2 operation.

Knowledge:

Digital encoding - representation of information, binary digits.

Mathematics areas:

Boolean Algebra.

Abilities:

Describe binary codewords for digital information, calculate Boolean operations.

Formula of sum modulo 2 for 2 digits:

$$S = M.\overline{K} + \overline{M}.K$$

where $\overline{M}$ and $\overline{K}$ represent the respectively opposite values (negated values, binary NO values), and Math operation is normal multiplication and summation.

Example: Let's have the [M,K] 2-bit codeword [1 0]. Here, M=1 and K=0. For their sum modulo 2 we will have:

$$S = 1.1 + 0.0 = 1 + 0 = 1$$

The symbol for sum modulo 2 in Boolean algebra is $\oplus$

Exaples for 2-bit M$\oplus$K :

$0 \oplus 0 = 0.1 + 1.0 = 0 + 0 = 0$
$0 \oplus 1 = 0.0 + 1.1 = 1$
$1 \oplus 0 = 1$
$1 \oplus 1 = 0$

Let's perform several parity calculations:

3-bit information codeword (m=3)	**Odd or even number of 1ˢ**	**Parity check bit value**	**Sum modulo 2** (n=4)
000	even	**0**	$0 \oplus 0 \oplus 0 \oplus \mathbf{0} = 0$
001	odd	**1**	$0 \oplus 0 \oplus 1 \oplus \mathbf{1} = 0$
010		**1**	$0 \oplus 1 \oplus 0 \oplus \mathbf{1} = 0$
011	even	**0**	$0 \oplus 1 \oplus 1 \oplus \mathbf{0} = 0$
100		**1**	$1 \oplus 0 \oplus 0 \oplus \mathbf{1} = 0$
101	even	**0**	$1 \oplus 0 \oplus 1 \oplus \mathbf{0} = 0$
110	even	**0**	$1 \oplus 0 \oplus 1 \oplus \mathbf{0} = 0$
111		**1**	$1 \oplus 1 \oplus 1 \oplus \mathbf{1} = 0$

Examples of parity check practical usage in contemporary systems is the 16-bit Header Checksum field onto IPv4 and TCP network protocols.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 1: IPv4 header [2]

Hamming code (n,m) (7,4)

TASK 3:

Perform error correction encoding with Hamming code (7,4)

Knowledge:

Algebra, Boolean algebra, code distances, Hamming codes, Matrix operations

Abilities:

Calculate Boolean operations, perform Algebra and matrix operations

One of the simplest systematic error correction code is the Hamming code. From Mathematical point of view Hamming codes (***n,m***) are a class of binary codes with error detecting and error correcting capability defined by the number of check bits ***k*** , which is in a direct connection with the code distance (Hamming distance) e.g. α_{min} bits. The **k** bits are considered check positions. The values in these ***k*** positions will be determined during the encoding process by performing parity checks over certain (selected, specially chosen) information bit positions. To give the position of any single error, the check number must describe m+k+1 different combinations, such that [1]:

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

Writing $n=m+k$ it is shown that [1]:

$$2^m \leq \frac{2^n}{n+1}$$

Using those inequalities, we calculate the minimum ***n*** for a given ***m*** presented in table 1.

Table 1: Calculated results for *k* correction bits [1]

total bits ***n***	**information bits** ***m***	**check bits** ***k***
7	4	3
8	4	4
9	5	4
10	6	4
11	7	4

From Table 1 is obvious that if we have **4** information bits, we need (at least) **3** correction bits. After determining the number of correcting bits, their positions are on focus - the positions over which each of the various parity checks to be applied. The checking number is obtained digit by digit, from left to right, by applying the parity checks in order and writing down the corresponding 0 or 1 (for even or odd number of 1^s that is calculated). Since the checking number is to give the position of any error in a code word, any position which has a 1 on he right of its binary representation must cause the first check to fail. Observing the binary form of the following integers it is seen they have something in common:

decimal integer		binary representation			
1	=	0	0	0	**1**
3	=	0	0	1	**1**
5	=	0	1	0	**1**
7	=	0	1	1	**1**
9	=	1	0	0	**1**

they all have 1^s on the extreme right. Here is the reason the first parity check must use positions 1, 3, 5, 7, 9 ,…

In an exactly similar fashion, the second parity check must use only the positions which have 1s for the second digit from the right of their binary representation [1].

decimal integer		binary representation			
2	=	0	0	**1**	0
3	=	0	0	**1**	1
6	=	0	1	**1**	0
7	=	0	1	**1**	1
10	=	1	0	**1**	0

The 3rd parity check:

decimal integer		binary representation			
4	=	0	**1**	0	0
5	=	0	**1**	0	1
6	=	0	**1**	1	0
7	=	0	**1**	1	1
12	=	1	**1**	0	0

The choice of the positions 1, 2, 4, 8 for check positions has the advantage of making the setting of the check positions independent of each other. All other positions are information positions [1].

Constructing a single error correcting code concludes in assigning $\boldsymbol{m}$ number of bits as information bits, and the number of check bits (extra, redundant bits) $\boldsymbol{k}$. Adding them together, the total number of bits will be calculated as $\boldsymbol{n}$. In our given task, $\boldsymbol{m}$ is 4 (fixed). To perform error correction coding with Hamming (7,4), we assume first that the information codeword contains **4** bits ($\boldsymbol{m}$=4). The added 3 correction (check) bits ($\boldsymbol{k}$=3) realize a code distance of 3 (α=3) in a way to find 1 error and correct this 1 error.

Let's have the following 4-bit information codeword: [1 1 1 0]. The calculated number of bits (n-bits) in the new codeword will be:

$$n = m + k = 4 + 3 = 7$$

7-bit encoded codeword	n_7	n_6	n_5	n_4	n_3	n_2	n_1
bit position in decimal (little endian logic, LSB in most right)	7	6	5	4	3	2	1
bit position in binary	0111	0110	0101	0100	0011	0010	0001
information bits	m_4	m_3	m_2		m_1		
values of inf. bits	1	1	1		0		
which of the m bits has 1 as LSB in its bit position?	m_4		m_2		m_1		

which of the *m* bits has 1 as a bit next to LSB (second bit)?	m_4	m_3			m_1		
which of the *m* bits has 1 as 3rd bit in its bit position?	m_4	m_3	m_2				
check bits (extra added, used for error correction)				k_3		k_2	k_1

Error correction bits will have the task to find (and correct) **positions** where errors occurred. To put control (to order) the positions, Hamming advised to check where **1**s are located in the bit positions hierarchically and perform a parity check with the involved information bit values. For this reason we must create the following linear equation system, answering the questions "Who has 1 at the corresponding positions?". Very important property of the check bits is that they appear only **once** in a given system of linear equations! Actually, the check bits positions give the correct parity check algorithm - only one appearance of check bit k_n in the separate hierarchical bit-positions equations.

$$\left|\begin{array}{l} k_1 = m_1 \oplus m_2 \oplus m_4 \\ k_2 = m_1 \oplus m_3 \oplus m_4 \\ k_3 = m_2 \oplus m_3 \oplus m_4 \end{array}\right.$$

$$\left|\begin{array}{l} k_1 = 0 \oplus 1 \oplus 1 \\ k_2 = 0 \oplus 1 \oplus 1 \\ k_3 = 1 \oplus 1 \oplus 1 \end{array}\right.$$

$$\left|\begin{array}{l} k_1 = 0 \oplus 1 \oplus 1 = 0 \\ k_2 = 0 \oplus 1 \oplus 1 = 0 \\ k_3 = 1 \oplus 1 \oplus 1 = 1 \end{array}\right.$$

TASK 4:

Perform 1-bit error correction decoding with Hamming code (7,4)

Knowledge:

Boolean algebra, Hamming codes, code distances, Matrix operations

The searched encoded 7-bit codeword is:

n_7	n_6	n_5	n_4	n_3	n_2	n_1
1	1	1	0	1	0	0

Now, let's assume that due to AWGN in the telecommunication channel, 1 bit is errored and the digital processor (software, code) should correct it. Let's assume the 3-rd bit position n_3 is received in error i.e. instead of a **1**, the receiver has received a **0**.

Abilities:

Calculate Boolean operations, perform matrix operations

The received 7-bit codeword is:

n_7	n_6	n_5	n_4	n_3	n_2	n_1
1	1	1	0	**0**	0	0

To reverse the encoding operation, namely, to decode, we shall calculate the parity check with the received codeword following the algorithm with the system of linear equations. The algorithm requires parity check for the bit positions ordered hierarchically in the form:

7-bit codeword	n_7	n_6	n_5	n_4	n_3	n_2	n_1
values	1	1	1	0	**0**	0	0
bit positions, MSB … LSB	111	110	101	100	011	010	001
a 1 in LSB	n_7		n_5		n_3		n_1
a 1 in the second bit	n_7	n_6			n_3	n_2	
a 1 in MSB	n_7	n_6	n_5	n_4			

The binary calculation (the parity check) is performed, and the result will be observed bottom-up:

$$\left|\begin{array}{l} n_1 \oplus n_3 \oplus n_5 \oplus n_7 = \\ n_2 \oplus n_3 \oplus n_6 \oplus n_7 = \\ n_4 \oplus n_5 \oplus n_6 \oplus n_7 = \end{array}\right.$$

$$\left|\begin{array}{l} 0 \oplus \mathbf{0} \oplus 1 \oplus 1 = \mathbf{0} \\ 0 \oplus \mathbf{0} \oplus 1 \oplus 1 = \mathbf{0} \\ 0 \oplus 1 \oplus 1 \oplus 1 = \mathbf{1} \end{array}\right.$$

The result read bottom-up is: 100. The result gives the bit-position of the error. 100 in binary is 3 in decimal.

Feel free to perform bit correction if other position is errored.

Hamming code 11,7

TASK 5:

Perform error correction encoding with Hamming code (11,7)

Knowledge:

Algebra, Boolean algebra, code distances, Hamming codes, Matrix operations

Abilities:

Calculate Boolean operations, perform Algebra and matrix operations

Let's have the following 7-bit (m=7) information codeword: [1 0 1 1 0 1 0]. According to the Hamming's theory, the calculated number of bits (n-bits) in the newly encoded codeword will be:

$$n = 7 + 4 = 11$$

The number of check bits is 4, their positions are: 1st, 2nd, 4th, and 8th, according to Hamming's theory - appear only once in the system of bit-position equations.

11-bit word	n_{11}	n_{10}	n_9	n_8	n_7	n_6	n_5	n_4	n_3	n_2	n_1
values?											
bit position (dec)	11	10	9	8	7	6	5	4	3	2	1
bit position (bin)	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
infor-mation bits	m_7	m_6	m_5		m_4	m_3	m_2		m_1		
values	1	0	1		1	0	1		0		
check bits				k_4				k_3		k_2	k_1
values?				?				?		?	?

Again, error correction bits will have the task to find (and correct) **positions** where errors occurred. To put control (to order) the positions, Hamming advised to check where **1**s are located in the bit positions hierarchically and perform a parity check with the involved information bit values. For this reason we must create the following linear equation system, answering the questions "Who has 1 at the corresponding positions?":

$$\left|\begin{aligned} k_1 &= m_1 \oplus m_2 \oplus m_4 \oplus m_5 \oplus m_7 \\ k_2 &= m_1 \oplus m_3 \oplus m_4 \oplus m_6 \oplus m_7 \\ k_3 &= m_2 \oplus m_3 \oplus m_4 \\ k_4 &= m_5 \oplus m_6 \oplus m_7 \end{aligned}\right.$$

$$\left|\begin{aligned} k_1 &= 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 0 \\ k_2 &= 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = 0 \\ k_3 &= 1 \oplus 0 \oplus 1 = 0 \\ k_4 &= \oplus 1 \oplus 0 \oplus 1 = 0 \end{aligned}\right.$$

11-bit word	n_{11}	n_{10}	n_9	n_8	n_7	n_6	n_5	n_4	n_3	n_2	n_1
infor-mation bits	m_7	m_6	m_5		m_4	m_3	m_2		m_1		
values	1	0	1		1	0	1		0		
check bits				k_4				k_3		k_2	k_1
values				0				0		0	0
11-bit word	1	0	1	0	1	0	1	0	0	0	0

TASK 6:

Perform 1-bit error correction decoding with Hamming code (11,7)

Knowledge:

Algebra, Boolean algebra, code distances, Hamming codes, Matrix operations

Abilities:

Calculate Boolean operations, perform Algebra and matrix operations

Now, let's assume that due to electromagnetic interference in the telecommunication channel, 1 bit is errored and the digital processor (software, code) should correct it. Let's assume the 7th bit position (n_7) is received in error i.e. instead of a **1**, the receiver has received a **0**.

received 11-bit word	n_{11}	n_{10}	n_9	n_8	n_7	n_6	n_5	n_4	n_3	n_2	n_1
values	1	0	1	0	**0**	0	1	0	0	0	0
bit position (bin)	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001

Now, we perform parity check hierarchically for every bit-position equation:

$$\left|\begin{array}{c} n_1 \oplus n_3 \oplus n_5 \oplus n_7 \oplus n_9 \oplus n_{11} = \\ n_2 \oplus n_3 \oplus n_6 \oplus n_7 \oplus n_{10} \oplus n_{11} = \\ n_4 \oplus n_5 \oplus n_6 \oplus n_7 = \\ n_8 \oplus n_9 \oplus n_{10} \oplus n_{11} = \end{array}\right.$$

$$\left|\begin{array}{c} 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 1 \\ 0 \oplus 0 \oplus 0 \oplus 0 \oplus 0 \oplus 1 = 1 \\ 1 \oplus 1 \oplus 0 \oplus 1 = 1 \\ 0 \oplus 1 \oplus 0 \oplus 1 = 0 \end{array}\right.$$

The result read bottom-up is: 0111. The result gives the bit-position of the error. 0111 in binary is 7 in decimal - the position of the erorr.

Feel free to perform bit correction if other position is errored.

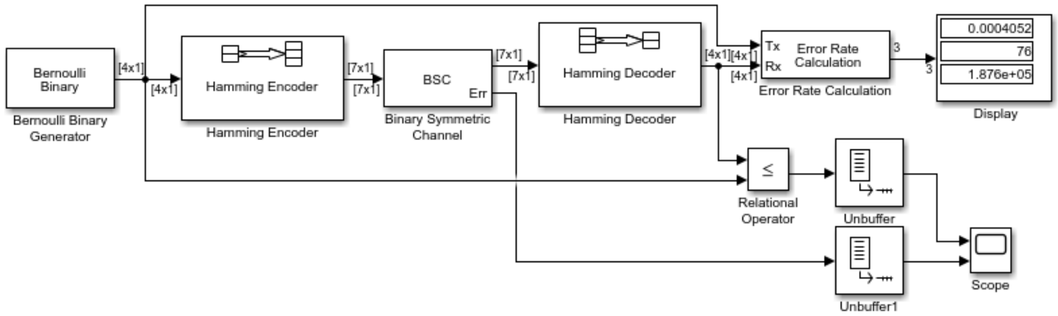

Fig. 1: Hamming encoder in Simulink [3]

Python program for Hamming encoding [4]:

```
# Python program to demonstrate hamming code

def calcRedundantBits(m):

    # Use the formula 2 ^ r >= m + r + 1  to calculate the number of redundant bits
    # Iterate over 0 .. m and return the value that satisfies the equation

    for i in range(m):
        if(2**i >= m + i + 1):
            return i

def posRedundantBits(data, r):

   # Redundancy bits are placed at the positions which correspond to the power of 2.
    j = 0
    k = 1
    m = len(data)
    res = ''

    # If position is power of 2 then insert '0'
    # Else append the data
    for i in range(1, m + r+1):
        if(i == 2**j):
            res = res + '0'
            j += 1
        else:
            res = res + data[-1 * k]
            k += 1
```

```
    # The result is reversed since positions are
    # counted backwards. (m + r+1 ... 1)
    return res[::-1]

def calcParityBits(arr, r):
    n = len(arr)

    # For finding rth parity bit, iterate over 0 to r - 1

    for i in range(r):
        val = 0
        for j in range(1, n + 1):

            # If position has 1 in ith significant position then Bitwise OR the array value
            # to find parity bit value.
            if(j & (2**i) == (2**i)):
                val = val ^ int(arr[-1 * j])
                # -1 * j is given since array is reversed

        # String Concatenation
        # (0 to n - 2^r) + parity bit + (n - 2^r + 1 to n)
        arr = arr[:n-(2**i)] + str(val) + arr[n-(2**i)+1:]
    return arr

def detectError(arr, nr):
    n = len(arr)
    res = 0

    # Calculate parity bits again
    for i in range(nr):
        val = 0
        for j in range(1, n + 1):
            if(j & (2**i) == (2**i)):
                val = val ^ int(arr[-1 * j])

        # Create a binary number by appending parity bits together.

        res = res + val*(10**i)
```

<table>
<tr><td></td><td>

```
    # Convert binary to decimal
    return int(str(res), 2)

# Enter the data to be transmitted
data = '1011001'

# Calculate the number of Redundant Bits Required
m = len(data)
r = calcRedundantBits(m)

# Determine the positions of Redundant Bits
arr = posRedundantBits(data, r)

# Determine the parity bits
arr = calcParityBits(arr, r)

# Data to be transferred
print("Data transferred is " + arr)

# Stimulate error in transmission by changing a bit value.
# 10101001110 -> 11101001110, error in 10th position.

arr = '11101001110'
print("Error Data is " + arr)
correction = detectError(arr, r)
if(correction==0):
    print("There is no error in the received message.")
else:
    print("The position of error is ",len(arr)-correction+1,"from the left")
```

</td></tr>
<tr><td>Digital mathematics tools</td><td>Matlab and Simulink [3].
Python programming environment [4].</td></tr>
<tr><td>Methodologies adopted</td><td>- problem-based learning - learning based on problems - consists in solving binary math problems related to digital errors.</td></tr>
<tr><td>The role of Digital/Software</td><td>Innovative teaching solutions can be the use of modelling and programming software. Practical graphics analyses and personally created models may be used.</td></tr>
</table>

Mathematics instruments	The specialized software for binary simulations – depicting error rates assures deeper understandings. Math formulas, describing the code words may be adopted in a programming manner for Python language for example, in order to calculate it over a software. By this adoption, the students are more likely to understand the semantics of the digital codewords' nature, and furthermore - the error correction process.
Estimated time	4 hours (including self-studies and syndicate work)
Assessment	Ability to calculate binary errors. Documentation – equations, matrices, correct calculations and real results. Presentation with screenshots available and proper explanation of questions that arise. Completed feedback.

References:

[1] Richard W. Hamming, Error Detecting and Error Correcting Codes, The Bell System Technical Journal, April, 1950, ATT©.
[2] IPv4 RFC 791, URL: https://datatracker.ietf.org/doc/html/rfc791
[3] Simulink©, URL: https://uk.mathworks.com/help/comm/ug/error-detection-and-correction.html
[4] URL: https://www.geeksforgeeks.org/python/hamming-code-implementation-in-python/

ERROR CORRECTION CODING WITH CYCLICAL REDUNDANCY CHECK

Generator polynomials

One step ahead towards encoding efficiency in the face of ability to correct more binary errors is the binary operation “Cyclical redundancy check” (CRC). It is found as a standard in telecommunications and is used today for example in Data Link Ethernet packets [1, 2].

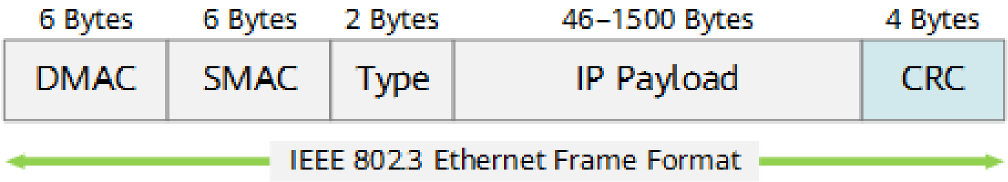

Fig. 1: Ethernet protocol with CRC appended [2]

The telecommunication binary operations can be programmed into software by typing the equations of codewords and Math operations with generator polynomials. Examples of equations are:

Table 1: Example 1 for codeword and polynomial

code-word ***m***	1	0	1	1	0	0	0	1	1
equa-tion *F(x)* =	X^8		X^6	X^5				X^1	X^0

Pure Math description:

$F(x) = [101100011]_2$

$F(x) = 1.X^8+0.X^7+1.X^6+1.X^5+0.X^4+0.X^3+0.X^2+1.X^1+1.X^0$

$F(x) = X^8+X^6+X^5+X^1+1$

Important to be stated is the value of X. Speaking for binary, it is 2.

$F(x) = 1.2^8+0.2^7+1.2^6+1.2^5+0.2^4+0.2^3+0.2^2+1.2^1+1.2^0$

$F(x) = 2^8+2^6+2^5+2^1+1$

The specifics of redundancy in error correction encoding propose appended bits. Those appended bits will have specific values, calculated in the for of special checksum - Cyclical checksum.

CRC-4-ITU

TASK 1: Perform error correction encoding of letter “O” with Cyclical Redundancy Check (CRC-4-ITU) **Knowledge:** Algebra, Boolean algebra, generator polynomials, multiplication of polynomials **Abilities:** Calculate Boolean operations, perform multiplication of polynomials **Mathematics areas:** Operations with polynomials.	The information codeword for encoding is the Letter “O”. We find the binary codeword from the ASCII table: $O_{(2, ASCII)} = [1\ 1\ 1\ 1\ 0\ 0\ 1] =$ $= 1.X^6+1.X^5+1.X^4+1.X^3+0.X^2+0.X^1+1 =$ $=1.2^4+0.2^3+0.2^2+1.2^1+1.2^0$ Here, the information bits are ***m***=7, and the information polynomial is: $\boldsymbol{m}(x) = X^6+X^5+X^4+X^3+1$ The error correction bits appended are 4, respectively from the CRC-4-ITU polynomial (the largest degree = 4, so 4 bits appended). The CRC-4-ITU polynomial is: $\boldsymbol{k}(x)= X^4+X+1 =$ $= 2^4+2^1+2^0 =$ $=1.2^4+0.2^3+0.2^2+1.2^1+1.2^0$ The algorithm for CRC encoding is: 1. Multiply the information codeword with X^4 2. Divide the information polynomial with the CRC generator polynomial 3. Append the calculated redundancy bits at the right position - form the new encoded codeword ***n***(x). ….
TASK 2: Perform 1-bit error correction decoding of letter “O” with Cyclical Redundancy Check (CRC-4-ITU). Let the error be in the 5-th position	The algorithm for CRC decoding is: 1. Divide the received codeword with the CRC generator polynomial a. Check the redundancy. If the number of „*1*s“ in the redundancy bits is zero, no errors have occurred in the channel. or b. Check the redundancy. If the number of „*1*s“ in the redundancy bits is more than 1, shift the register to right 2. Check the redundancy. If the number of „*1*s“ in the redundancy bits is more than 1, shift the register to right 3. Check the redundancy. If the number of „*1*s“ in the redundancy bits is 1, then append the redundancy onto the received codeword. 4. Perform shift register to left as many times as performed to right.

5. The correct codeword is shown.

.

EXAMPLE POLYNOMIALS

Standard	Polynomial
CRC-1	$x + 1$ (most hardware; also known as *parity bit*)
CRC-4-ITU	$x^4 + x + 1$ (ITU-T G.704, p. 12)
CRC-5-EPC	$x^5 + x^3 + 1$ (Gen 2 RFID)
CRC-5-ITU	$x^5 + x^4 + x^2 + 1$ (ITU-T G.704, p. 9)
CRC-5-USB	$x^5 + x^2 + 1$ (USB token packets)
CRC-6-ITU	$x^6 + x + 1$ (ITU-T G.704, p. 3)
CRC-7	$x^7 + x^3 + 1$ (telecom systems, ITU-T G.707, ITU-T G.832, MMC, SD)
CRC-8-CCITT	$x^8 + x^2 + x + 1$ (ATM HEC), ISDN Header Error Control and Cell Delineation ITU-T I.432.1 (02/99)
CRC-8-Dallas/Maxim	$x^8 + x^5 + x^4 + 1$ (1-Wire bus)
CRC-8	$x^8 + x^7 + x^6 + x^4 + x^2 + 1$
CRC-8-SAE J1850	$x^8 + x^4 + x^3 + x^2 + 1$
CRC-8-WCDMA	$x^8 + x^7 + x^4 + x^3 + x + 1$[16]
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x + 1$ (ATM; ITU-T I.610)
CRC-11	$x^{11} + x^9 + x^8 + x^7 + x^2 + 1$ (FlexRay)
CRC-12	$x^{12} + x^{11} + x^3 + x^2 + x + 1$ (telecom systems)
CRC-15-CAN	$x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$
CRC-16-IBM	$x^{16} + x^{15} + x^2 + 1$ (Bisync, Modbus, USB, ANSI X3.28, many others; also known as *CRC-16* and *CRC-16-ANSI*)
CRC-16-CCITT	$x^{16} + x^{12} + x^5 + 1$ (X.25, V.41, HDLC, XMODEM, Bluetooth, SD, many others; known as *CRC-CCITT*)
CRC-16-T10-DIF	$x^{16} + x^{15} + x^{11} + x^9 + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ (SCSI DIF)
CRC-16-DNP	$x^{16} + x^{13} + x^{12} + x^{11} + x^{10} + x^8 + x^6 + x^5 + x^2 + 1$ (DNP, IEC 870, M-Bus)
CRC-16-DECT	$x^{16} + x^{10} + x^8 + x^7 + x^3 + 1$ (cordless telephones)

CRC-16-Fletcher	Not a CRC; see Fletcher's checksum
CRC-24	$x^{24} + x^{22} + x^{20} + x^{19} + x^{18} + x^{16} + x^{14} + x^{13} + x^{11} + x^{10} + x^8 + x^7 + x^6 + x^3 + x + 1$ (FlexRay)
CRC-24-Radix-64	$x^{24} + x^{23} + x^{18} + x^{17} + x^{14} + x^{11} + x^{10} + x^7 + x^6 + x^5 + x^4 + x^3 + x + 1$ (OpenPGP)
CRC-30	$x^{30} + x^{29} + x^{21} + x^{20} + x^{15} + x^{13} + x^{12} + x^{11} + x^8 + x^7 + x^6 + x^2 + x + 1$ (CDMA)
CRC-32-IEEE 802.3	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ (V.42, Ethernet, SATA, MPEG-2, PNG, POSIX cksum)
CRC-32C (Castagnoli)	$x^{32} + x^{28} + x^{27} + x^{26} + x^{25} + x^{23} + x^{22} + x^{20} + x^{19} + x^{18} + x^{14} + x^{13} + x^{11} + x^{10} + x^9 + x^8 + x^6 + 1$ (iSCSI & SCTP, G.hn payload, SSE4.2)
CRC-32K (Koopman)	$x^{32} + x^{30} + x^{29} + x^{28} + x^{26} + x^{20} + x^{19} + x^{17} + x^{16} + x^{15} + x^{11} + x^{10} + x^7 + x^6 + x^4 + x^2 + x + 1$
CRC-32Q	$x^{32} + x^{31} + x^{24} + x^{22} + x^{16} + x^{14} + x^8 + x^7 + x^5 + x^3 + x + 1$ (aviation; AIXM)
CRC-40-GSM	$x^{40} + x^{26} + x^{23} + x^{17} + x^3 + 1$ (GSM control channel)
CRC-64-ISO	$x^{64} + x^4 + x^3 + x + 1$ (HDLC — ISO 3309, Swiss-Prot/TrEMBL; considered weak for hashing)
CRC-64-ECMA-182	$x^{64} + x^{62} + x^{57} + x^{55} + x^{54} + x^{53} + x^{52} + x^{47} + x^{46} + x^{45} + x^{40} + x^{39} + x^{38} + x^{37} + x^{35} + x^{33} + x^{32} + x^{31} + x^{29} + x^{27} + x^{24} + x^{23} + x^{22} + x^{21} + x^{19} + x^{17} + x^{13} + x^{12} + x^{10} + x^9 + x^7 + x^4 + x + 1$ (as described in ECMA-182 p. 51)

References:

[1] URL: https://www.geeksforgeeks.org/computer-networks/ethernet-frame-format/

[2] URL: https://info.support.huawei.com/info-finder/encyclopedia/en/CRC.html

SCENARIO 3

ERROR CORRECTION CODING WITH CONVOLUTIONAL ENCODERS	
TASK 1: . **Knowledge**: . **Mathematics areas:** . **Abilities:** s.	