Chapter IV:
Error Protection

IV.1 – INTRODUCTION

Electromagnetic radiation and internal system disturbances (such as distortion and noise) can alter transmitted information, resulting in bit errors. Given the expansion of networks and the extensive use of optical fiber, the loss of clock synchronization is now the primary source of errors.

IV.2 – BIT ERROR RATE (BER)

The ***Bit Error Rate (BER)*** is defined as the ratio of the number of erroneously received information bits to the total number of transmitted information bits.

$$BER = \frac{Number\ of\ bit\ errors}{Total\ number\ of\ bits\ transmitted}$$

For example, consider the transmission of the sequence "011001001100100101001010", which is received as "011001**1**011001010**11**01000010".

The received message differs from the transmitted message by 3 bits. The total number of transmitted bits is 24.

The Bit Error Rate (BER) is therefore: **BER = 3 / 24 = 0.125**

IV.3 – ERROR DETECTION

Error detection refers to the mechanisms implemented to allow a destination system to verify the validity of received data. Error detection relies on the introduction of a certain amount of redundancy into the transmitted information. Four techniques can be used to detect and potentially correct errors:

- **Echo Check:** The receiver sends the received message back to the sender as an echo. If the echoed message is different from the one originally sent, the sender retransmits the message. This technique is used in asynchronous terminals (e.g., Telnet).
- **Detection by Repetition:** Each transmitted message is followed by its duplicate. If the two messages are different, the receiver requests a retransmission. This technique is used in highly disturbed (noisy) environments and in certain real-time applications.
- **Error Detection via Calculated Checksum/Key:** Supplementary information (a "key" or checksum), derived from the transmitted information, is appended to it. Upon reception, the receiver recalculates the checksum. If the result matches the received checksum, the data is considered correct. Otherwise, the receiver discards the received data and may request a retransmission (error recovery).
- **Error Detection and Correction by Code:** This technique consists of substituting the characters to be transmitted with a binary combination different from the base encoding, creating a self-correcting code.

Author: Dr. H. Zerroiki

IV.4 – ERROR DETECTION VIA CALCULATED CHECKSUM/KEY

IV.4.1 Principle

In systems using a calculated checksum (key), a control sequence (CTL1), derived from a mathematical operation applied to the message to be sent, is transmitted along with the message. The receiver performs the same operation. If the resulting checksum (CTL2) is identical to the one calculated by the source (CTL1), the block is considered correct; otherwise, the block is rejected.

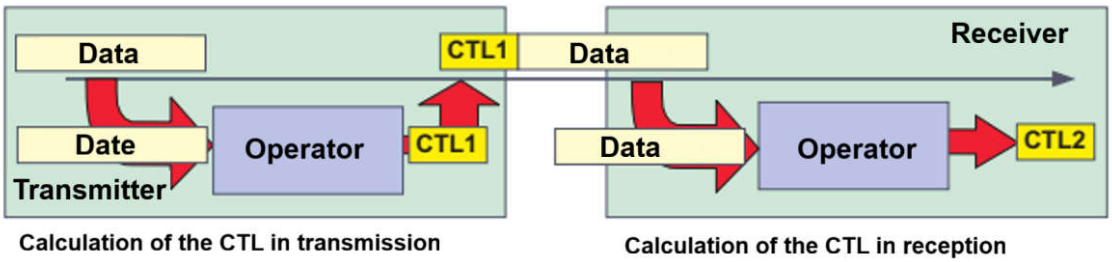

Figure 1: *Principle of Error Detection via Calculated Checksum*

IV.4.2 The Parity Bit Technique

The parity bit technique consists of adding one bit to the binary sequence to be protected, such that the sum of the '1' bits in the transmitted sequence is either even (parity bit) or odd (oddness bit). This modulo-2 arithmetic is simple but introduces only weak redundancy. The protection it offers is limited to a single character.

Letter	ASCII Code (7bits)	Codeword (Even Parity)	Codeword (Odd Parity)
B	1000010	10000100	10000101
D	1000100	10001000	10001001
F	1000110	10001101	10001100
R	1010010	10100101	10100100

This technique, known as **VRC (Vertical Redundancy Check)**, can only detect an odd number of bit errors within a character. It is primarily used in asynchronous transmissions.

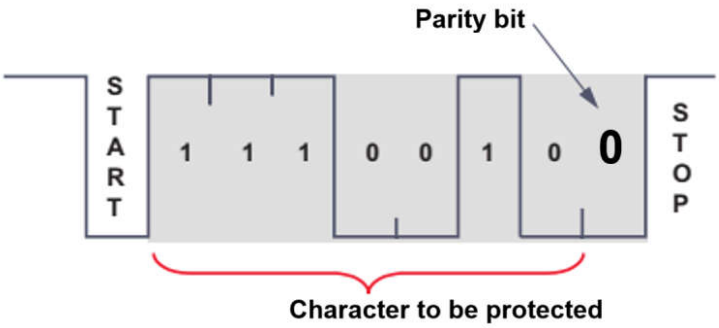

Figure 2: *Asynchronous Character Frame with Parity Bit*

In synchronous transmissions, characters are sent in blocks. The single parity bit technique is insufficient and is supplemented with another piece of information: the **LRC (Longitudinal Redundancy Check)**.

Author: Dr. H. Zerroiki

Character to Transmit	Parity Bit	Character to Transmit	Parity Bit	...	LRC Character	Parity Bit

In this control mode, known as a **two-dimensional parity check**, an LRC character is appended to the transmitted block. Each bit of the LRC character corresponds to the parity of the bits in the same position across all characters in the block: the first bit of the LRC is the parity of all the 1st bits of each character, the second bit is the parity of all the 2nd bits, and so on. This newly formed character is appended to the message. The LRC character is itself protected by a parity bit (VRC).

Letter	ASCII Code (7bits)	VRC
H	1001000	0
E	1000101	1
L	1001100	1
L	1001100	1
O	1001111	1
LRC	1000010	0

1001000	0	1000101	1	1001100	1	1001100	1	1001111	1	1000010	0
H		E		L		L		O		LRC	

IV.4.3 Cyclic Codes (CRC, Cyclic Redundancy Check)

In calculated checksum detection, the redundant information, the key (**CRC, Cyclic Redundancy Check**), is determined by a complex mathematical operation applied to the block of data to be transmitted and is sent along with it.

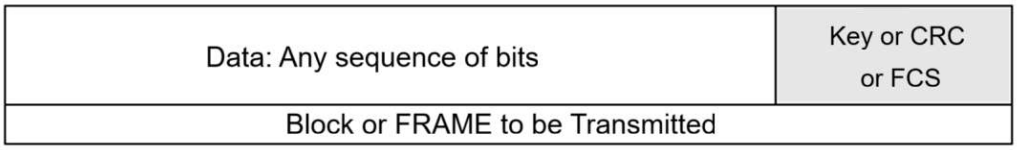

Figure 3: *Structure of a Bit Block Protected by a Calculated Checksum.*

The calculated checksum method treats the block of *N* bits to be transmitted as a polynomial P(x) of degree *N-1*. This polynomial is divided by another polynomial, known as the generator polynomial G(x), using the rules of Boolean or modulo-2 arithmetic. The remainder of this division, R(x), constitutes the CRC, sometimes also called the **FCS (Frame Check Sequence)**. The calculated CRC is transmitted following the data block.

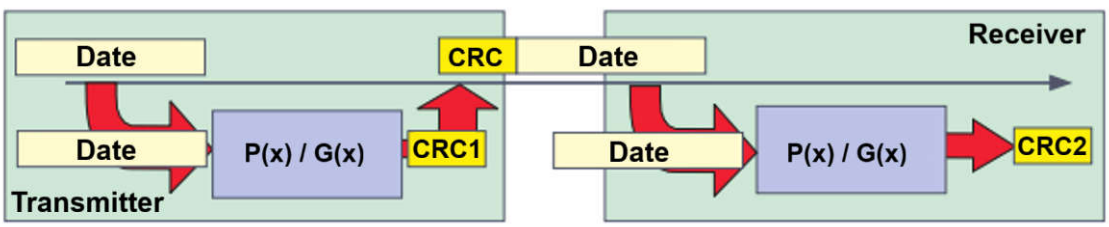

Figure 4: *CRC Error Detection Process*

Author: Dr. H. Zerroiki

Upon reception, the destination performs the same operation on the received block. The transmitted CRC and the receiver's calculated CRC are compared; if the values differ, an error is signaled.

In practice, the method used is slightly different. If **D** is the dividend, **d** is the divisor, and **R** is the remainder, then the division **(D - R) / d** yields a **zero** remainder. Following this principle, the division of the entire **block (data + CRC)**, i.e., **P(x) + R(x)**, by the generator polynomial **G(x)** will yield a remainder of **zero**.

At the receiver, the hardware performs the division on the entire data block, including the calculated checksum. When the calculated remainder is zero (and the next character is a flag), the block is considered correct.

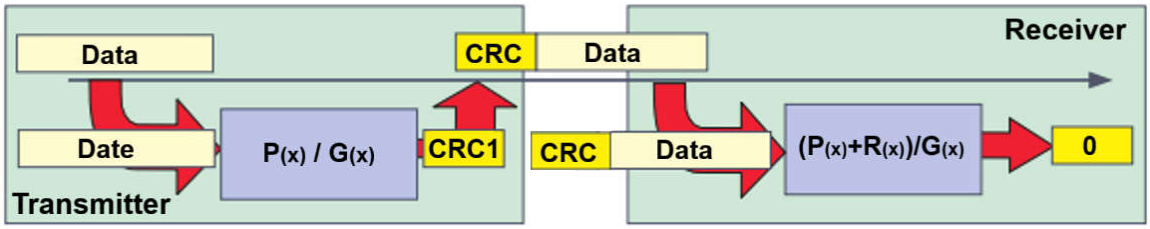

Figure 5: *CRC Error Detection*

Modulo-2 arithmetic is arithmetic without a carry. To ensure this condition, before performing the division, the polynomial **P(x)** is multiplied by $\mathbf{x^m}$, where ***m*** is the degree of the generator polynomial. This corresponds to shifting the data by ***m*** positions (i.e., appending *m* zero bits).

Recall that the remainder of a division by a divisor of degree ***m*** is of degree ***m-1*** and thus contains ***m*** terms. This operation effectively inserts ***m*** **zero** bits at the end of the message, creating space to add the terms of the remainder.

Example:
We wish to protect the message "**110111**" with a checksum calculated using the generator polynomial $\mathbf{x^2 + x + 1}$.

The message **110111** corresponds to the polynomial: $x^5 + x^4 + 0x^3 + x^2 + x^1 + x^0$

To allow for the addition of the checksum to the message, we multiply the representative polynomial by $\mathbf{x^m}$, where ***m*** is the degree of the generator polynomial (**m=2**). The dividend becomes:

$$(x^5 + x^4 + 0\,x^3 + x^2 + x^1 + 1) \times x^2 = x^7 + x^6 + 0x^5 + x^4 + x^3 + x^2 + 0 + 0$$

Division is performed by hardware systems that use exclusive OR operations. Therefore, let's apply exclusive OR division to the polynomial **1010010111**.

If the generator polynomial is $x^4 + x^2 + x + 1$, it corresponds to the binary sequence **10111**.

Since the degree of the generator polynomial is **4**, we add **4 zeros** to the data frame (initializing a 4-position register to zero).

Applying exclusive OR division to the polynomial **1010010111**, we obtain the division below:

```
1 0 1 0 0 1 0 1 1 1 0 0 0 0 | 1 0 1 1 1
1 0 1 1 1                   | 100 110 0100
0 0 0 1 1 1 0 1             |
      1 0 1 1 1             |
      0 1 0 1 0 1           |
        1 0 1 1 1           |
        0 0 0 1 0 1 0 0     |
              1 0 1 1 1     |
              0 0 0 1 1 0 0 |
```

The remainder (key) has 4 bits and is of degree 1 with respect to the generating polynomial. The remainder, or CRC4, is therefore **1100**. The message to be transmitted is :

P(x) + R(x): 1010010111**1100**.

At reception, the entire message, including data and key, undergoes the same operation; if the remainder of the division is equal to zero, it is assumed that the message was not affected by a transmission error.

Let's verify this statement with the previous example:

```
message          CRC
1 0 1 0 0 1 0 1 1 1 1100 | 1 0 1 1 1
1 0 1 1 1                |
0 0 0 1 1 1 0 1          |
      1 0 1 1 1          |
      0 1 0 1 1 1        |
        0 0 0 1 0 1 11   |
              1 0 1 11   |
              0 0 0 0000 |
```

IV.5 – SELF-CORRECTING CODES (HAMMING CODE)

The principle of error-correcting codes is the same as for detector codes: the original word to be transmitted (data word) is substituted with a new word (codeword). This involves:

- **At the transmitter:** adding supplementary check bits.
- **At the receiver:** detecting errors using the check bits and having the ability to correct those errors.

The **Hamming code** is a linear error-correcting code. It allows for the automatic detection and correction of a single-bit transmission error. This correction is made possible by adding redundant information. It is based on parity tests. The simplest version can correct a single-bit error.

To ***m*** information bits, we add ***k*** parity check bits. We therefore have **m + k = n** total bits. Since the ***k*** check bits must indicate the **n + 1** possible error outcomes (including the 'no error' case), the following condition must be met: $\mathbf{2^k \geq n + 1}$

The $\mathbf{2^k}$ possible combinations of the ***k*** bits are used to encode the position of the error. Once this position is calculated, the bit at that position can be corrected.

Author: Dr. H. Zerroiki

It is advantageous to choose codes with minimal redundancy that provide a maximum number of information positions. This is achieved by setting $\mathbf{n = 2^k - 1}$.

If we number the bits from right to left starting at 1, the check bits (or parity bits) are placed at positions that are powers of 2 (bit numbers 1, 2, 4, 8, 16...). Each check bit performs a parity check (even or odd) on a specific set of data bits. This process determines the *n* bits to be transmitted or stored. Note that the rightmost bit is position 1, not 0.

If the number of information bits is 4 (*m = 4*), we can construct a 7-bit Hamming code (*n = 7*) by adding 3 check bits (*k = 3*).

7	**6**	**5**	**4**	**3**	**2**	**1**
m4	m3	m2	**k3**	m1	**k2**	**k1**

The 3 check bits k3, k2, and k1 are placed at the power-of-2 positions: k1 at position 1, k2 at position 2, and k3 at position 4.

We will now see which check bits verify the parity of each message bit:

Bit Position	**Binary Rep.**	**Sum of Powers of 2**	**Checked by**
7	(0111)	4 + 2 + 1	k3, k2, k1
6	(0110)	4 + 2	k3, k2
5	(0101)	4 + 1	k3, k1
4	(0100)		**is check bit k3**
3	(0011)	2 + 1	k2, k1
2	(0010)		**is check bit k2**
1	(0001)		**is check bit k1**

Conversely, which bits are checked by which parity bit?

- **k1** checks bits 1, 3, 5, 7
- **k2** checks bits 2, 3, 6, 7
- **k3** checks bits 4, 5, 6, 7

For even parity, for example, **k1** must be set such that the total number of '1's across bits 1, 3, 5, and 7 is even.

For even parity, we simply perform a modulo-2 addition (XOR). For odd parity, we would perform an inverted modulo-2 addition.

When the information is received or read, the parity check is performed again. For each check bit, the received value is compared to the recalculated value:

- If they are identical, we assign a value of **0** to the binary variable $\mathbf{A_i}$ associated with check bit $\mathbf{k_i}$.
- Otherwise, we assign it a value of **1**.

Author: Dr. H. Zerroiki

The resulting binary number $\mathbf{A_k...A_2A_1}$ is called the **syndrome**, and its value indicates the position of the error.

Let's revisit the previous example:

- Suppose the value associated with k_1 is A_1=1;
- Suppose the value associated with k_2 is A_2=1;
- Suppose the value associated with k_3 is A_3=0;

We then know that an error exists, and the syndrome is: $\mathbf{A_3A_2A_1 = 011}$.

An error detected by $\mathbf{k_1}$ can only come from bits whose binary address ends in 1, i.e., 1, 3, 5, or 7. Similarly, the $\mathbf{k_2}$ test covers bits 2, 3, 6, 7. Finally, the $\mathbf{k_3}$ check covers bits 4, 5, 6, 7.

You can now understand that a syndrome value of $\mathbf{A_3A_2A_1=000}$ indicates the absence of an error.

A value of $\mathbf{A_3A_2A_1=001}$ indicates an error in bit number 1.

A value of $\mathbf{A_3A_2A_1=110}$ indicates an error in bit number 6.

Example:

Encode **1010 1011 001** using even parity. Here, m=11, so k=4 is required (since $2^4 \geq 11+4+1$). The message to be transmitted will contain n=15 bits.

Position	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Type	m11	m10	m9	m8	m7	m6	m5	**k4**	m4	m3	m2	**k3**	m1	**k2**	**k1**
Value	1	0	1	0	1	0	1	**?**	1	0	0	**?**	1	*?*	*?*

In the data word to be transmitted, we have '1's in the following (final) positions: 15, 13, 11, 9, 7, 3. We transform these positions into their binary values and add them using modulo-2 (XOR) to find the parity bits.

15	=	1	1	1	1
13	=	1	1	0	1
11	=	1	0	1	1
9	=	1	0	0	1
7	=	0	1	1	1
3	=	0	0	1	1

		0	**1**	**0**	**0** → Parity bits
		k_4	**k_3**	**k_2**	**k_1**

So, k_4=**0**, k_3=**1,** k_2=**0,** k_1=**0**.

The encoded message is therefore 1010 101**0** 100**1** **1**00.

Author: Dr. H. Zerroiki

Now, suppose the following Hamming word was received: **1010 0010 1001 100**. We have '1' bits in the following positions: 15, 13, 9, 7, 4, 3.

We perform the same modulo-2 addition to calculate the syndrome:

15	=	1	1	1	1	
13	=	1	1	0	1	
9	=	1	0	0	1	
7	=	0	1	1	1	
4	=	0	1	0	0	
3	=	0	0	1	1	
		---	---	---	---	Modulo-2 Sum
		1	**0**	**1**	**1**	
		$\mathbf{A_4}$	$\mathbf{A_3}$	$\mathbf{A_2}$	$\mathbf{A_1}$	→ error at position **11**

The syndrome $\mathbf{A_4A_3A_2A_1 = 1011}$, which corresponds to the decimal number **11**. This indicates an error at **position 11**.

After correcting the bit at position 11 (flipping it from 0 to 1), we get the correct message: **1010 1010 1001 100**.