Elias-style block protection

Elias-style block protection uses horizontal and vertical control bits. It should be used if the protected data can be written in matrix form. During the decoding process, the logical values of the equations are examined both individually and combined.

Example

A binary data which is stored in a 3×3 matrix is given: **101011001**

Let's write it down in matrix form and attach parity bits, too.

1	0	1	**0**
0	1	1	**0**
0	0	1	**1**
1	**1**	**1**	**1**

Let's suppose that the first bit was wrong during transmission, then the parity bit in the row and column will show us the bad bit:

1→0	0	1	**0 → 1**
0	1	1	**0**
0	0	1	**1**
1 → 0	**1**	**1**	**1 → 0**

Error Detection and Correction Using Hamming Codes

Hamming codes are a family of error-correcting codes that can detect up to two-bit errors and correct single-bit errors in transmitted data. They use the Hamming distance concept to determine where errors may have occurred.

If we have \(m\) number of data bits then let's attach \(r\) number of redundant parity bits to it, so the whole bit length will be:

$$ n = m + r $$

If two code-words are given, for example : **0101110** and **0111110** and the only difference between them is 1 bit, then the '**Hamming distance**' of these code-words will be 1. This is an interesting measure for distance because it does not matter which bit in the row is different or whether it belongs to a binary or decimal system. So for example : **45635263** and **45615263** have the **Hamming distance** of 1 too. It does not matter that 10 different digits could be in the place of the wrong digit.

The **Hamming-style** correction code supposed to increase the number of parity bits. To correct single bit errors we have to use **\(r\)** number of parity bits using this formula:

$$ 2^r \ge m + r + 1 $$

According to this formula the necessary number of parity bits which are needed to correct single bit errors are stated in the following table:

Data bits (\(m\))	number of parity bits (\(r\))	whole bit length (\(n = m + r\))	% of added bits
4	3	7	66
8	4	12	50
16	5	21	31
32	6	38	19
64	7	71	11
128	8	136	6
256	9	265	4
512	10	522	2

How Hamming Codes Work?

Hamming codes introduce additional bits, called parity bits, into the data. These parity bits are strategically placed within the data to allow the detection of errors. The positions of the parity bits are powers of two (e.g., positions 1, 2, 4, 8, etc.).

For example, in an 11-bit data block, we might have 4 parity bits, making it a 15-bit Hamming code.

- **Detecting Errors**: After transmission, the parity bits are recalculated and compared with the received parity bits. If there is any mismatch, the system can determine which bit is incorrect by checking the positions covered by the erroneous parity bits.
- **Correcting Errors**: If a single-bit error is detected, the Hamming code identifies which bit is incorrect (from the parity bits' positions) and corrects it by flipping the bit.

Example 1.

In this simple example we encode 4 bits of data into 7 bits by adding 3 parity bits.

Data Bits: Let’s say we want to transmit the 4-bit data **1010**.

Placement of Parity Bits: The 7-bit Hamming code places the parity bits in positions that are powers of 2: positions 1, 2, and 4. So, the 7-bit frame will look like this (with p1, p2, p4 being the parity bits):

```
p1 p2 d1 p4 d2 d3 d4
```

Filling in the data **1010**:

```
p1 p2 1 p4 0 1 0
```

Calculating Parity Bits: We calculate the parity bits such that each one covers a specific set of positions:

- p1 covers bit positions: 1, 3, 5, 7
- p2 covers bit positions: 2, 3, 6, 7
- p4 covers bit positions: 4, 5, 6, 7

Parity bit **p1** covers all positions where the **least significant** bit of the position number is **1**. Positions covered: 1, 3, 5, 7 (since 001, 011, 101, and 111 in binary all have a 1 in the least significant bit).

Parity bit **p2** covers all positions where the **second least significant** bit is 1. Positions covered: 2, 3, 6, 7 (since 010, 011, 110, and 111 in binary all have a 1 in the second bit).

Parity bit **p4** covers all positions where the **third least significant** bit is 1. Positions covered: 4, 5, 6, 7 (since 100, 101, 110, and 111 in binary all have a 1 in the third bit).

In general we can build a table to help to determine covers positions:

1. bit (p1)	1	3	5	7	9	11	13	15	17	19	21
2. bit (p2)	2	3	6	7	10	11	14	15	18	19	
4. bit (p4)	4	5	6	7	12	13	14	15	20	21	
8. bit (p8)	8	9	10	11	12	13	14	15			
16. bit (p16)	16	17	18	19	20	21					

Now calculate the values of parity bits:

1. bit	**2. bit**	**3. bit**	**4. bit**	**5. bit**	**6. bit**	**7. bit**
p1	p2	1	p4	0	1	0

p1: Covers 1, 0, 0 (positions 1, 3, 5, 7), so p1 = 1 (to make the total even).

p2: Covers 0, 1, 0 (positions 2, 3, 6, 7), so p2 = 1 (to make the total even).

p4: Covers 0, 1, 0 (positions 4, 5, 6, 7), so p4 = 1 (to make the total even).

Thus, the final transmitted Hamming code is:

```
1 1 1 1 0 1 0
```

Let's suppose that because of an error the 3rd bit goes wrong. In this case **p1** and **p2** will be wrong. Because of the **3rd** bit the first and second parity bit will give us wrong values, but the others will not because they do not calculate with the bit standing at the **3rd** place.

The common values of the first and second parity bits are the following: 1, 2, 3, 5, 6, 7. The defective one has to be among these.

However from 5, 6, 7 are included in the good parity bits → therefore the wrong parity bit is the **3rd** one. (note that: 1, 2 cannot be wrong in this case. If 1 was wrong, only the **p1** parity bit would be wrong.)

Example 2.

Assume that, we try to transmit 16 bits: `1001011101011011`

The following table show the placement of the parity bits and the data bits:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
p1	p2	1	p4	0	0	1	p8	0	1	1	1	0	1	0	p16	1	1	0	1	1