**Calculating the Number of Parity Bits**

We need to determine how many parity bits (r) are required for a message of length m. The formula is:

2r≥m+r+12^r \geq m + r + 1

**Why this formula?**

* m is the number of data bits.
* r is the number of parity bits.
* The +1 accounts for the overall parity check (used in extended Hamming codes).
* Each parity bit must be able to check a unique combination of bits. So we need enough parity bits to cover all positions in the final codeword (data + parity).

**2. Positioning Parity Bits**

Parity bits are placed at positions that are powers of 2:

* Positions: 1, 2, 4, 8, 16, ...

These positions are reserved for parity bits. The rest are filled with the actual data bits.

**Example:**

For a 4-bit message (1011), we need 3 parity bits (r = 3), so the total length becomes 7. The positions are:

| **Position** | **Type** |
| --- | --- |
| 1 | Parity |
| 2 | Parity |
| 3 | Data |
| 4 | Parity |
| 5 | Data |
| 6 | Data |
| 7 | Data |

**3. Calculating Parity Bits**

Each parity bit checks a subset of bits based on binary indexing.

**Rule:**

A parity bit at position 2k2^k checks all positions where the kthk^{th} bit in the binary representation is **1**.

**Example:**

Parity bit at position 1 (binary 0001) checks positions:

* 1 (0001)
* 3 (0011)
* 5 (0101)
* 7 (0111)

Parity bit at position 2 (binary 0010) checks:

* 2 (0010)
* 3 (0011)
* 6 (0110)
* 7 (0111)

**Parity Calculation:**

We use **even parity** (most common). So the parity bit is set such that the total number of 1s in the checked positions is even.

Parity=XOR of all bits it checks\text{Parity} = \text{XOR of all bits it checks}

**4. Error Detection**

When a message is received, we recalculate the parity bits using the same rules. If there's a mismatch, we identify the error position.

**How?**

We check each parity bit. If a parity bit fails, we note its position. The sum of all failed parity bit positions gives the **error location**.

**Example:**

If parity bits at positions 1 and 2 fail:

Error position=1+2=3\text{Error position} = 1 + 2 = 3

Flip the bit at position 3 to correct the error.

**🧮 5. Binary Arithmetic Used**

* **Bitwise AND (**&**)**: Used to check if a bit is set in a position.
* **Bitwise XOR (**^**)**: Used to calculate parity.
* **Binary indexing**: Every position is treated as a binary number to determine which parity bits affect it.

**🧠 Summary Table**

| **Concept** | **Formula / Rule** | **Purpose** |
| --- | --- | --- |
| Parity bits count | 2r≥m+r+12^r \geq m + r + 1 | Ensure coverage of all positions |
| Parity bit positions | Powers of 2 (1, 2, 4, ...) | Reserved for parity |
| Parity calculation | XOR of relevant bits | Enforce even parity |
| Error detection | Sum of failed parity positions | Locate single-bit error |