Hamming Code for Error Detection & Correction

Overview

Hamming Code is used for **Error Detection & Correction** in data transmission. It uses approximately 50% redundancy, where half the bits are data bits and half are parity bits.

Basic Structure

7-bit Hamming Code

- **4 bits**: Data bits
- **3 bits**: Parity bits
- **Total**: 7 bits

Identifying Parity vs Data Bits

Use the formula **2^n** where n = 0, 1, 2, 3...

Parity bit positions:

- 2^0 = 1 → Position 1 is parity (P0)
- 2^1 = 2 → Position 2 is parity (P1)
- 2^2 = 4 → Position 4 is parity (P2)
- 2^3 = 8 → Position 8 is parity (P3)

Remaining positions are data bits.

Example: 7-bit Code Structure

```
Position: 1  2  3  4  5  6  7
Type:     P0 P1 D0 P2 D1 D2 D3
```

Example: 11-bit Code Structure

```
Position: 1  2  3  4  5  6  7  8  9  10 11
Type:     R1 R2 D  R4 D  D  D  R8 D  D  D
```

- **4 parity bits**: R1, R2, R4, R8
- **7 data bits**: Remaining positions

Encoding Process

Step 1: Place Data Bits

For data **1010**:

- D3 = 1 (MSB)
- D2 = 0
- D1 = 1
- D0 = 0 (LSB)

Step 2: Calculate Parity Bits

Calculating P0 (Position 1)

- **Rule**: Pick odd-numbered positions (1, 3, 5, 7...)
- **Bits involved**: D0, D1, D3
- **Values**: 0, 1, 1
- **XOR operation**: $0 \oplus 1 \oplus 1 = 0$
- **Result**: P0 = 0

Calculating P1 (Position 2)

- **Rule**: Pick 2, leave 2, pick 2, leave 2... (2,3,6,7,10,11...)
- **Bits involved**: D0, D2, D3
- **Values**: 0, 0, 1
- **XOR operation**: $0 \oplus 0 \oplus 1 = 1$
- **Result**: P1 = 1

Calculating P2 (Position 4)

- **Rule**: Pick 4, leave 4, pick 4... (4,5,6,7,12,13,14,15...)
- **Bits involved**: D1, D2, D3
- **XOR operation**: Calculate accordingly
- **Result**: P2 = calculated value

Error Detection & Correction Process

Example Scenario

Original codeword: Transmitted successfully **Received codeword**: 9th bit changed from 0 to 1 (intentional error)

Detection Steps

1. **Receiver calculates parity for each group**

2. **Compares with received parity bits**
3. **Mismatch indicates error presence**

Correction Steps

Step 1: Calculate R1 (checking odd positions)

- **Positions checked**: 1, 3, 5, 7, 9, 11
- **If error in position 9**: Parity check fails
- **Result**: R1 = 1 (indicates error in odd positions)

Step 2: Calculate R2

- **Positions checked**: 2, 3, 6, 7, 10, 11
- **If error not in these positions**: Parity check passes
- **Result**: R2 = 0 (no error in these positions)

Step 3: Calculate R4

- **Positions checked**: 4, 5, 6, 7, 12, 13, 14, 15
- **If error not in these positions**: Parity check passes
- **Result**: R4 = 0 (no error in these positions)

Step 4: Calculate R8

- **Positions checked**: 8, 9, 10, 11, 12, 13, 14, 15
- **If error in position 9**: Parity check fails
- **Result**: R8 = 1 (indicates error in positions 8-15)

Error Location Calculation

Error position = R8 × 8 + R4 × 4 + R2 × 2 + R1 × 1 **Error position** = 1 × 8 + 0 × 4 + 0 × 2 + 1 × 1 = **9**

Error Correction

- **Error found at position 9**
- **Current bit value**: 1
- **Correction**: Flip bit from 1 to 0
- **Result**: Error corrected

Key Points to Remember

1. **Parity positions follow 2^n pattern**: 1, 2, 4, 8, 16...
2. **Each parity bit covers specific bit patterns**

3. **XOR operation used for even parity calculation**
4. **Error position calculated using binary representation**
5. **Single-bit errors can be detected and corrected**
6. **Approximately 50% overhead for error correction capability**

Parity Checking Patterns

Parity Bit	Positions Covered	Pattern
P0 (R1)	1,3,5,7,9,11...	Pick odd positions
P1 (R2)	2,3,6,7,10,11...	Pick 2, skip 2, repeat
P2 (R4)	4,5,6,7,12,13...	Pick 4, skip 4, repeat
P3 (R8)	8,9,10,11,16,17...	Pick 8, skip 8, repeat

Applications

- **Data Link Layer error control**
- **Memory systems (ECC RAM)**
- **Digital communication systems**
- **Storage systems requiring error correction**