Topics

- Error detecting codes
- Error correcting codes
- Hamming algorithm

Where Do Errors Come From?

- Voltage spikes from power supply
- Coupling with nearby signals
- Noise on transmission line
- Radiation
- Manufacturing defect
- Etc.
Error Detecting Code

- **Hamming Distance**
  - Number of bit positions in which two codewords differ

- **Minimum Distance**
  - Minimum Hamming distance between all distinct pairs of codewords

A code with minimum distance $d_{min}$ can **detect** all error patterns of weight less than or equal to $(d_{min} - 1)$

Parity Check

- For even or odd parity
  - $d_{min}$ equals 2
  - Detects error patterns $\leq (d_{min} - 1)$
    - Single bit error
    - Generates an error flag

- Uses of parity check
  - Memory
  - Disk storage
Error Correcting Code

- **Hamming Distance**
  - Number of bit positions in which two codewords differ

- **Minimum Distance**
  - Minimum Hamming distance between all distinct pairs of codewords

A code with minimum distance $d_{\text{min}}$ can **correct** all error patterns of weight less than or equal to $[(d_{\text{min}} - 1)/2]$

Error Correction

- **Scenario**: transmit a yes/no answer
  - Codeword for “yes” = $1111_2$
  - Codeword for “no” = $0000_2$

- Minimum distance equals 4
  - Error correction less than or equal to $[(d_{\text{min}} - 1)/2]$

- Therefore our code corrects all errors $\leq 1.5$
  - The 0.5 correction capability is wasted because errors occur as whole bits
  - Generally want $d_{\text{min}}$ to be odd number
Hamming Codes

- First major class of binary codes designed for error correction
- Originally used in error control for long-distance telephony
- Encodes parity for groups of bits within data

Error Correcting for 4-Bit Words

Encode 0100₂ with even parity

\[
m_3 \ m_2 \ m_1 \ m_0 \ r_A \ r_B \ r_C
\]

0 1 0 0 1 1 1
Error Correcting for 4-Bit Words

What about a single error?

\[
\begin{array}{cccccccc}
m_3 & m_2 & m_1 & m_0 & r_A & r_B & r_C \\
0 & 1 & 0 & 0 & 1 & 0 & 1 \\
\end{array}
\]

Method for Parity Check

- Parity check for Circle A
  \[ m_3 \oplus m_2 \oplus m_1 \oplus r_A \]

- Parity check for Circle B
  \[ m_3 \oplus m_2 \oplus m_0 \oplus r_B \]
  Even parity \( \rightarrow 0 \)
  Odd parity \( \rightarrow 1 \)

- Parity check for Circle C
  \[ m_2 \oplus m_1 \oplus m_0 \oplus r_C \]
Hamming Algorithm

• In a Hamming code
  – $r$ parity bits added to $m$-bit word
  – Forms codeword with length $(m + r)$ bits

• Bit numbering
  – Starts at 1 with leftmost (high-order) bit
  – All powers of 2 are parity bits
  – Remaining bits are for data

Requirements for Single-Bit Error Correction

<table>
<thead>
<tr>
<th>Word size</th>
<th>Check bits</th>
<th>Total size</th>
<th>Percent overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>4</td>
<td>12</td>
<td>50</td>
</tr>
<tr>
<td>16</td>
<td>5</td>
<td>21</td>
<td>31</td>
</tr>
<tr>
<td>32</td>
<td>6</td>
<td>38</td>
<td>19</td>
</tr>
<tr>
<td>64</td>
<td>7</td>
<td>71</td>
<td>11</td>
</tr>
<tr>
<td>128</td>
<td>8</td>
<td>136</td>
<td>6</td>
</tr>
<tr>
<td>256</td>
<td>9</td>
<td>265</td>
<td>4</td>
</tr>
<tr>
<td>512</td>
<td>10</td>
<td>522</td>
<td>2</td>
</tr>
</tbody>
</table>

Fig. 2-13. Number of check bits for a code that can correct a single error.
Bit Numbering for Hamming Algorithm

Given an 8-bit data word to encode

Data bits (remaining bits)

Parity bits (powers of 2)

<table>
<thead>
<tr>
<th>Bit No.</th>
<th>Bit No. in Binary</th>
<th>Encoded by</th>
<th>Encodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8’s 4’s 2’s 1’s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0 0 0 1</td>
<td>3, 5, 7, 9, 11</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0 0 1 0</td>
<td>3, 6, 7, 10, 11</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0 0 1 1</td>
<td>2, 1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0 1 0 0</td>
<td>5, 6, 7, 12</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0 1 0 1</td>
<td>4, 1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0 1 1 0</td>
<td>4, 2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0 1 1 1</td>
<td>4, 2, 1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1 0 0 0</td>
<td>9, 10, 11, 12</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1 0 0 1</td>
<td>8, 1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1 0 1 0</td>
<td>8, 2</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1 0 1 1</td>
<td>8, 2, 1</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1 1 0 0</td>
<td>8, 4</td>
<td></td>
</tr>
</tbody>
</table>

Parity Bit Assignment
Example

Given an 8-bit data word to encode

- Codeword has 12 bits (8 data, 4 parity)
  - Bit 1 checks: 1, 3, 5, 7, 9, 11
  - Bit 2 checks: 2, 3, 6, 7, 10, 11
  - Bit 4 checks: 4, 5, 6, 7, 12
  - Bit 8 checks: 8, 9, 10, 11, 12

Method for Parity Check

- For parity bit 1
  - $b_1 \oplus b_3 \oplus b_5 \oplus b_7 \oplus b_9 \oplus b_{11}$

- For parity bit 2
  - $b_2 \oplus b_3 \oplus b_6 \oplus b_7 \oplus b_{10} \oplus b_{11}$

- For parity bit 4
  - $b_4 \oplus b_5 \oplus b_6 \oplus b_7 \oplus b_{12}$

- For parity bit 8
  - $b_8 \oplus b_9 \oplus b_{10} \oplus b_{11} \oplus b_{12}$

Even parity $\rightarrow$ 0
Odd parity $\rightarrow$ 1
Class Exercise

- Compute the codeword for the 8-bit data:

0111 0001

- Bit 1 checks: 1, 3, 5, 7, 9, 11 correct
- Bit 2 checks: 2, 3, 6, 7, 10, 11 incorrect
- Bit 4 checks: 4, 5, 6, 7, 12 incorrect
- Bit 8 checks: 8, 9, 10, 11, 12 correct

Class Exercise

- Find the bit that is incorrect in the following codeword. This 12 bit codeword encodes 8 data bits.

0001 1001 1101

- Bit 1 checks: 1, 3, 5, 7, 9, 11 correct
- Bit 2 checks: 2, 3, 6, 7, 10, 11 incorrect
- Bit 4 checks: 4, 5, 6, 7, 12 incorrect
- Bit 8 checks: 8, 9, 10, 11, 12 correct

Bit 6