Hamming Codes. Part II - Decoding

Information Theory Lab 10

Objective

Perform decoding of a data file encoded with the Hamming (8,4) SECDED (Single Error Correction - Double Error Detection) code, including both detection and correction (when possible) of errors.

Theoretical notions

Encoding

Hamming (8,4) SECDED encoding procedure operates on a block of 4 information bits (denoted as i_3, i_5, i_6, i_7) and produces a block of 8 output bits:

$$\mathbf{c} = c_0 c_1 c_2 i_3 c_4 i_5 i_6 i_7$$

The bits denoted with c are parity bits (or *control* bits), and are computed as follows:

$$\begin{cases} c_0 = i_3 \oplus i_5 \oplus i_6 = c_1 \oplus c_2 \oplus i_3 \oplus c_4 \oplus i_5 \oplus i_6 \oplus i_7 \\ c_1 = i_3 \oplus i_5 \oplus i_7 \\ c_2 = i_3 \oplus i_6 \oplus i_7 \\ c_4 = i_5 \oplus i_6 \oplus i_7 \end{cases}$$

The coding is done for every group of 4 bits in the data.

In the C language, modulo-2 sum is done by the bitwise `XOR` operation (`^`).

Decoding

Considering the received 8-bit block

$$\mathbf{r} = r_0 r_1 r_2 r_3 r_4 r_5 r_6 r_7,$$

decoding is done by computing the **syndrome**:

$$\begin{cases} z_0 &= r_0 \oplus r_1 \oplus r_2 \oplus r_3 \oplus r_4 \oplus r_5 \oplus r_6 \oplus r_7 \\ z_1 &= r_4 \oplus r_5 \oplus r_6 \oplus r_7 \\ z_2 &= r_2 \oplus r_3 \oplus r_6 \oplus r_7 \\ z_3 &= r_1 \oplus r_3 \oplus r_5 \oplus r_7 \end{cases}$$

The following cases may take place:

- $z_0 = z_1 = z_2 = z_3 = 0$: no error
- $z_0 = 1$: 1 error on position given by $z_1 z_2 z_{3(10)}$
- $z_0 = 0$, other $z_i \neq 0$: 2 errors on unknown positions

Exercises

1. Write a C program that performs decoding of a data file previously with the encoded Hamming (8,4) SECDED code.

   The program shall be called as follows:

   `HammingDecode.exe encoded.dat decoded.dat`

   - The arguments are:
     - `encoded.dat`: the input file, previously encoded
     - `decoded.dat`: the output decoded file
   - The program should consist of the following steps:
     - declare two large vectors of `unsigned char`, for input and output bits
     - open the input file and read everything into the input vector
     - for every group of 8 bits from the input vector:
       * compute the syndrome bits z_0, z_1, z_2, z_3
       * if one error is detected, fix the error (by toggling the erroneous bit) and display a message "Fixed 1 error\n"
       * if two errors are detected, display a message "Detected 2 errors, can't fix\n"
       * write the 4 information bits (i_3, i_5, i_6, i_7) in the output vector
       * advance and repeat
     - write the output vector to the output data file

2. Take one encoded file (use the program from the previous lab for encoding) and use the website `hexed.it` to introduce a few random errors, not more than 1 error per byte. Save the modified file and run the program on it. Check that the errors have been fixed and the decoded output is identical to the original file.

3. Take one encoded file (use the program from the previous lab for encoding) and use the website `hexed.it` to introduce also 2 errors in a single byte. Save the modified file and run the program on it. Check the output against the original file. Have all errors been fixed?

Program design

1. The bit operation macro definitions are available in `bitmacros.h`.

   For quick access to a bit number i in a large array, use the following macros:

```
#define VECREAD_BIT(v,i) (READ_BIT((v[(i)/8]),(i)%8))
#define VECWRITE_BIT(v,i,val) (WRITE_BIT((v[(i)/8]),(i)%8, val))
#define VECTOGGLE_BIT(v,i) (TOGGLE_BIT((v[(i)/8]),(i)%8))
```

Final questions

1. TBD