Lab Assignment 2 CS301 2024FALL

Hamming code is an error-detecting encoding method that adds redundant bits to data for error detection and correction. This experiment aims to familiarize students with the principles and implementation of Hamming code, using the MiniSTM32 development board combined with STM32CubeIDE software to implement the encoding and decoding functions of Hamming code.

Example of Hamming code

Example 1

Suppose the original data is `1011` , then we need to include 3 additional parity bits and arrange the 7 bits as shown in the table:

Bit Position	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Encoded data bits	p1	p2	d1	p4	d2	d3	d4
Hamming Code	0	1	1	0	0	1	1

The encoding and decoding processes are as follows:

Encoding process

- Confirm the number of parity bits
  - Confirm the bit number k in the original data, in this example, $k = 4$.
  - Calculate the number of additional bit r required based on $2^r >= k + r + 1$, in this example, $r = 3$, which means we need 3 parity bits in addition to the original 4-bit data.
- Confirm the positions of parity bits
- - Parity bit positions are 2^n, in this example, parity bit positions are 1, 2, 4
  - Noted the position starts from 1 and are counted rightward.
- Calculate the parity bits
  - Bit 1 checks the parity of the bits in positions 1,3,5,7,... it is stored in p1
  - Bit 2 checks the parity of the bits in positions 2,3,6,7,... it is stored in p2
  - Bit 4 checks the parity of the bits in positions 4,5,6,7,... it is stored in p4
- Assuming the parity bits are P_1, P_2, P_4, then in this example,

  $$P_1 = XOR\ of\ bits(3, 5, 7) = 1 \oplus 0 \oplus 1 = 0$$

  $$P_2 = XOR\ of\ bits(3, 6, 7) = 1 \oplus 1 \oplus 1 = 1$$

  $$P_4 = XOR\ of\ bits(5, 6, 7) = 0 \oplus 1 \oplus 1 = 0$$
- Obtain the Hamming code `0110011` where p1 = 0, p2 = 1, and p4 = 0.
  - Noted each parity bit is added so that the total number of 1's in the checked positions, including the parity bit, is always even.

Bit Position	1	2	3	4	5	6	7
Encoded data bits	p1	p2	d1	p4	d2	d3	d4
Parity bit coverage for p1	×		×		×		×
Parity bit coverage for p2		×	×			×	×
Parity bit coverage for p4				×	×	×	×
Hamming code	0	1	1	0	0	1	1

Decoding process (even parity):

- When receiving the 7 bit message, they are checked again for errors. The parity is checked over the same combination of bits, including the parity bit. the checks are evaluated as follows:
  - $C_1 = XOR\ of\ bits(1,3,5,7) = 0 \oplus 1 \oplus 0 \oplus 1 = 0$
  - $C_2 = XOR\ of\ bits(2,3,6,7) = 1 \oplus 1 \oplus 1 \oplus 1 = 0$
  - $C_4 = XOR\ of\ bits(4,5,6,7) = 0 \oplus 0 \oplus 1 \oplus 1 = 0$
- For even parity check, a 0 check bit designates even parity over the checked bits and a 1 designates odd parity. The result $C = C_4C_2C_1 = 000$, indicates that no error has occurred.
- However, if $C \neq 0$, then the binary number formed by the check bits gives the position of the erroneous bit. For example, if $C = C_4C_2C_1 = 101$ (a decimal 5), then the erroneous bit is at position 5.

Example 2

Suppose the original data is `1100 0100` , then we need to include 4 additional parity bits and arrange the 12 bits as shown in the table:

Bit Position	1	2	3	4	5	6	7	8	9	10	11	12
Encoded data bits	p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8
Hamming Code	0	0	1	1	1	0	0	1	0	1	0	0

The encoding and decoding processes are as follows:

Encoding process

- Confirm the number of parity bits: $k = 8$ and $r = 4$, which means we need 4 parity bits in addition to the original 8-bit data.
- Confirm the positions of parity bits: Parity bit positions are 2^n, in this example, parity bit positions are 1, 2, 4, 8
- Calculate the parity bits
  - Bit 1 checks the parity of the bits in positions 1,3,5,7,9,11,... it is stored in p1
  - Bit 2 checks the parity of the bits in positions 2,3,6,7,10,11,... it is stored in p2
  - Bit 4 checks the parity of the bits in positions 4,5,6,7,12,... it is stored in p4
  - Bit 8 checks the parity of the bits in positions 8,9,10,11,12,... it is stored in p8
- Assuming the parity bits are P_1, P_2, P_4, P_8, then in this example,

$$P_1 = XOR\ of\ bits(3,5,7,9,11) = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0$$

$$P_2 = XOR\ of\ bits(3,6,7,10,11) = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0$$

$$P_4 = XOR\ of\ bits(5,6,7,12) = 1 \oplus 0 \oplus 0 \oplus 0 = 1$$

$$P_8 = XOR\ of\ bits(9,10,11,12) = 0 \oplus 1 \oplus 0 \oplus 0 = 1$$

- Obtain the Hamming code `001110010100` where p1 = 0, p2 = 0, p4 = 1 and p8 = 1.

Bit Position	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Encoded data bits	p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8
Parity bit coverage for p1	×		×		×		×		×		×	
Parity bit coverage for p2		×	×			×	×			×	×	
Parity bit coverage for p4				×	×	×	×					×
Parity bit coverage for p8								×	×	×	×	×
Hamming Code	0	0	1	1	1	0	0	1	0	1	0	0

Decoding process (even parity):

- When receiving the 7 bit message, they are checked again for errors. The parity is checked over the same combination of bits, including the parity bit. the checks are evaluated as follows:
  - $C_1 = XOR\ of\ bits(1,3,5,7,9,11) = 0 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0$
  - $C_2 = XOR\ of\ bits(2,3,6,7,10,11) = 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0$
  - $C_4 = XOR\ of\ bits(4,5,6,7,12) = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0$
  - $C_8 = XOR\ of\ bits(8,9,10,11,12) = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 = 0$
- For even parity check, a 0 check bit designates even parity over the checked bits and a 1 designates odd parity. The result $C = C_8C_4C_2C_1 = 0000$, indicates that no error has occurred.
- However, if $C \neq 0$, then the binary number formed by the check bits gives the position of the erroneous bit. For example, if $C = C_8C_4C_2C_1 = 0011$ (a decimal 6), then the erroneous bit is at position 6.

Note

To check whether your decoded and encoded results are correct, you can refer to the content of the website: https://www.dcode.fr/hamming-error-correction

Tasks

Part One (90%)

Task 1. System Basic Configuration (5%)

- Set buttons KEY_WAKEUP, KEY0 and KEY1 as input devices, LED0, LED1 and LCD screen as display devices.
- Use KEY_WAKEUP to switch working modes, the basic working modes should include encoding mode and decoding mode.
- Debounce processing of the buttons.(5%)

Task 2. Encoding (30%)

- Use buttons as input, where KEY0 represents 0 and KEY1 represents 1. When KEY0 is pressed, LED0 should light on at the same time, and LED1 should light on when KEY1 is pressed. (2%)
- After the length of input data reaches the configuration value, show the input data in LCD screen. For example, if the length of the input data is set to 4, and the original data will display on LCD after we enter 4 digital numbers. (8%)
- After receiving the original data from the buttons, the program sends the encoded Hamming code to the LCD screen for display. The LCD screen is divided into 4 zones for displaying input content and results. Figure 1~3 present a simple example for LCD display.(20%)

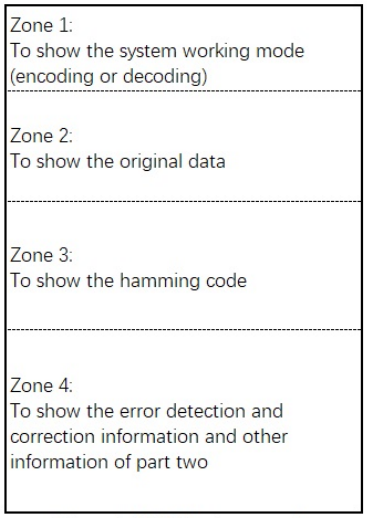

Figure 1. LCD display example

Figure 2. Encoding mode example

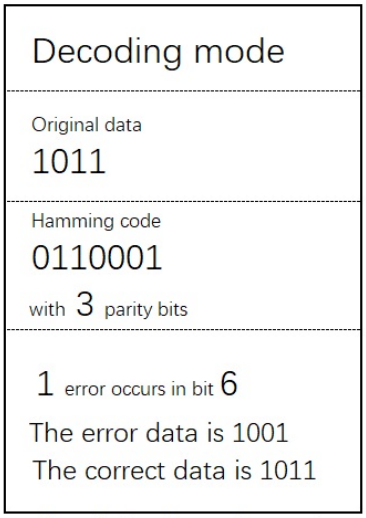

Figure 3. Decoding mode example

Task 3. Working mode switch (5%)

- Use KEY_WAKEUP to switch working modes.
- If the system works under encoding mode and KEY_WAKEUP is pressed, LED0 and LED1 flash **once** and the system switches to decoding mode.(2%)
- If the system works under decoding mode and KEY_WAKEUP is pressed, LED0 and LED1 flash **twice** and the system switches to encoding mode.(2%)
- After each switching, the new working mode should show on LCD screen.(1%)

Task 4. Decoding (45%)

- Use buttons as input, where KEY0 represents 0 and KEY1 represents 1. When KEY0 is pressed, LED0 should light on at the same time, and LED1 should light on when KEY1 is pressed.

- After the length of input data reaches the configuration value, show the input data in LCD screen. For example, if the length of the input data is set to 7, and the Hamming code will display on LCD after we enter 7 digital numbers.
- Decode the Hamming code entered via the buttons and display the decoded result on the LCD screen, including:
    - The correct original data. (5%)
    - Length of the parity bits. Takes `1010101` for example, it owns 3 parity bits, so `3` should be displayed on the LCD screen. (10%)
    - Whether an error occurred. You may use 0 or 1 to indicate the result, show 1 on the LCD screen when an error occurs and 0 when no error occurs. And you can also show some strings that show the information, such as `No error` or `1 error occurs` . (10%)
    - Position of the error. For example, if the 10th bit of the Hamming code is found to be incorrect during verification, display `P10` on the LCD screen. (10%)
- Display some intermediate values during the decoding verification process. Takes encoded Hamming code `1010101` for example, during verification processing, we need to check $P_1(1,3,5,7), P_2(2,3,6,7), P_4(4,5,6,7)$. The values of $P_1P_2P_4$ can be displayed on the LCD screen. (10%)

Task 5. Code Writing Standards (5%)

- The code should be written between the BEGIN and END comments with consistent remarks.
- Variables, macro definitions, and function declarations should be written in their corresponding positions.
- The codes for an individual peripheral should be generated as a pair of ".c/. h" files.
- The codes related to interruption handling should be written in the file of "stm32f1xx_it.c".
- Necessary comments are required.

Part Two (no more than 10%)

- Capable of performing two-bit error correction. (10%)
- Ability to manually adjust the maximum number of input data (both original data and Hamming code) bits and provide a warning when exceeding the maximum number of bits. (10%)
- Ability to insert images. (e.g., displaying whether decoding was successful or not using images.) (5%)
- Display decoding progress. (e.g., when verifying `1010101`, after completing P_1, the progress should be `one-third`.) (10%)
- Other function that is suitable for the whole system. (no more than 10%)

Submission demands

- Finish the assignment before DDL.
- Package the whole project into a compressed package and submit on Blackboard site.