Overview of Hamming Code Error Detection and Correction

Christian Kfoury
Gina Cody School of Engineering and Computer Science
Concordia University
City: Montreal, QC., Canada
Email: ch_kfour@live.concordia.ca

Pratham Patel
Gina Cody School of Engineering and Computer Science
Concordia University
City: Montreal, QC., Canada
Email: pratham.patelpr@gmail.com

Ali Zoubeidi
Gina Cody School of Engineering and Computer Science
Concordia University
City: Montreal, QC., Canada
Email: alihsz2002@gmail.com

Abstract**—** The following report provides an overview of the Hamming code, which is an error detection and correction method used in digital data transmission. The report begins by explaining the basic purpose of the Hamming code. It then gives detailed calculation on how to calculate the redundancy of a 2-byte message that is being transmitted. Additionally, it gives a step-by-step explanation on how a sender generates the parity bits and how a receiver detects and corrects a 1-bit error. Overall, this report provides a comprehensive introduction to the Hamming code.

I. Introduction

In today's society, digital technology has expanded quickly and become an extremely important part of our daily lives. Each day, a sizable amount of data is generated as a result of this development in digitization. To ensure that a user may access all data at any time, it must be efficiently stored and transmitted. Unfortunately, data corruption is possible because neither data recording nor transmission is ever completely error-free. When data is transmitted from a transmitter and received by a receiver, there is a chance that noise will be added, which can lead to the following error [1].

To avoid and prevent any data distortion from occurring, Richard Wesley Hamming came up with the Hamming code concept [2]. Hamming code is an error detection and correction used in the digital communication system. In short, hamming code is an error-correction technique that adds extra bits to a data word to create a code word that can detect and correct the errors [2].

In this report, we will provide an overview of the Hamming code by implementing it and showing its ability to detect and correct errors in digital communication systems. In other words, we will give a detailed calculation of the minimum number of parity bits needed for a message 2-byte length and discuss the details of how a sender generates the parity bits and how the receiver detects and corrects a 1-bit error.

II. Discussion

A. *Calculating the Minimum Number of Parity Bits Needed*

The Hamming code uses parity bits to add redundancy to the data bits, to make it possible to detect and correct errors in the data that has been transmitted. [3]. To find the number of parity bits that exist, the hamming code uses a simple formula that describes a relation between the number of data bits in a word and the number of parity bits [2]. To calculate the minimum amount of parity bits needed it must follow some steps:

Step 1: Determine the number of data bits to be transmitted or stored. This can be done by measuring the length of the sender message.

Step 2: Calculate the number of parity bits needed according to the length of the sender message using the following formula:

$$2^r \geq m + r + 1$$

Where:

r = redundancy/number of parity bits

m = number of data bits

Step 3: Choose the smallest value of redundancy (r) that satisfies the equation above.

Example

Assume the sender message has a length of 2 bytes and is the following: 1000 1101 1010 1100.

Step 1: In order to determine the number of data bits to be transmitted, it is important to find the length. The following message word sent has a length of 2 bytes.

The 2 bytes are then converted into bits by the following relation: 1 byte is equivalent to 8 bits, which then gives us a word with 16 data bits, as 2 bytes is equivalent to 16 bits.

Step 2: To calculate the number of parity bits needed we must use the following formula:

$$2^r \geq m + r + 1$$

Here the redundancy/number of parity bits (r) is unknown, and the number of data bits (m) is equal to 16 bits. To find the number of parity bits simply replace the number of data bits in the equation above.

$$2^r \geq 16 + r + 1$$

$$2^r \geq 17 + r$$

Step 3: To choose the smallest value of redundancy that satisfies the equation above:

If r = 0 à $2^0 = 1 \geq 17 + 0\ (false)$

If r = 1 à $2^1 = 2 \geq 17 + 1\ (false)$

If r = 2 à $2^2 = 4 \geq 17 + 2\ (false)$

If r = 3 à $2^3 = 8 \geq 17 + 3\ (false)$

If r = 4 à $2^4 = 16 \geq 17 + 4\ (false)$

If r = 5 à $2^5 = 32 \geq 17 + 5\ (true)$

The smallest value of redundancy that satisfies the inequality is 5. Therefore, for this message sent, there will be 5 parity bits.

B. *How Sender Generates the Parity Bits*

Step 1: After establishing the number of parity bits, those parity bits need to be assigned to all the bit positions. The position of the parity bits can be identified as the ones that can be expressed as power of 2's. In other words, for a message that has a length of 2 bytes will have 5 parity bits which will be 1, 2, 4, 8, and 16. To assign the parity bits to bit positions we must express each bit position as the sums of those number in the form of power of 2.

1 = 2^0

2 = 2^1

3 = $2^1 + 2^0$

4 = 2^2

5 = $2^2 + 2^0$

6 = $2^2 + 2^1$

7 = $2^2 + 2^1 + 2^0$

8 = 2^3

9 = $2^3 + 2^0$

10 = $2^3 + 2^1$

11 = $2^3 + 2^1 + 2^0$

12 = $2^3 + 2^2$

13 = $2^3 + 2^2 + 2^0$

14 = $2^3 + 2^2 + 2^1$

15 = $2^3 + 2^2 + 2^1 + 2^0$

16 = 2^4

17 = $2^4 + 2^0$

18 = $2^4 + 2^1$

19 = $2^4 + 2^1 + 2^0$

20 = $2^4 + 2^2$

21 = $2^4 + 2^2 + 2^0$

Step 2: Write down which bits do the parity bits checking. With a code word of length 2 byte (16 bits), and by following the previous step we can observe the following:

Bit 1 (= 2^0) contributes to 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21

Bit 2 (= 2^1) contributes to 2, 3, 6, 7, 10, 11, 14, 15, 18, 19

Bit 3 (= 2^2) contributes to 4, 5, 6, 7, 12, 13, 14, 15, 20, 21

Bit 4 (= 2^3) contributes to 8, 9, 10, 11, 12, 13, 14, 15

Bit 5 (= 2^4) contributes to 16, 17, 18, 19, 20, 21

Step 3: Do the XOR operation on each bit to perform even parity calculation and insert the result of each parity bit in their corresponding parity bit position.

Example

Let the 2-byte data message be 1000 1101 1010 1100.

1	0	0	0	1	P	1	0	1	1	0	1	0	P	1	1	0	P	0	P	P
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Bit 1:

0 XOR 0 XOR 1 XOR 0 XOR 0 XOR 1 XOR 1 XOR 1 XOR 1 = 1

Bit 2:

0 XOR 1 XOR 1 XOR 1 XOR 0 XOR 0 XOR 1 XOR 0 XOR 0 = 0

Bit 3:

0 XOR 1 XOR 1 XOR 1 XOR 1 XOR 0 XOR 1 XOR 0 XOR 1 = 0

Bit 4:

0 XOR 1 XOR 0 XOR 1 XOR 1 XOR 0 XOR 1 = 0

Bit 5:

1 XOR 0 XOR 0 XOR 0 XOR 1 = 0

The final message generated is the following:

1	0	0	0	1	0	1	0	1	1	0	1	0	0	1	1	0	0	0	0	1
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

C. *How Receiver Detects and Corrects a 1-bit error*

Example

Let's introduce an error in bit position b17, the code word in part B, would be the following 1 0001 1101 1010 0110 0001.

1	0	0	0	0	0	1	0	1	1	0	1	0	0	1	1	0	0	0	0	1
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Step 1: For the receiver to detect the error, write down which bits do the parity bits checking. With a code word of length 2 bytes (16 bits), and by following the previous step we can observe the following:

Bit 1 (= 2^0) contributes to 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21

Bit 2 (= 2^1) contributes to 2, 3, 6, 7, 10, 11, 14, 15, 18, 19

Bit 3 (= 2^2) contributes to 4, 5, 6, 7, 12, 13, 14, 15, 20, 21

Bit 4 (= 2^3) contributes to 8, 9, 10, 11, 12, 13, 14, 15

Bit 5 (= 2^4) contributes to 16, 17, 18, 19, 20, 21

Step 2: XOR each bit to perform even parity calculation and compare the result of each parity bit to their corresponding parity bit position.

Bit 1:

0 XOR 0 XOR 1 XOR 0 XOR 0 XOR 1 XOR 1 XOR 0 XOR 0 XOR 1 = 0 [Error]

Bit 2:

0 XOR 1 XOR 1 XOR 1 XOR 0 XOR 0 XOR 1 XOR 0 XOR 0 = 0

Bit 3:

0 XOR 1 XOR 1 XOR 1 XOR 1 XOR 0 XOR 1 XOR 0 XOR 1 = 0

Bit 4:

0 XOR 1 XOR 0 XOR 1 XOR 1 XOR 0 XOR 1 = 0

Bit 5:

0 XOR 0 XOR 0 XOR 0 XOR 1 = 1 [Error]

Here, as we can see from the message above, Parity Bit 1 and Parity Bit 5 both produce an error.

Step 3: Detecting the single-bit

By looking at Parity Bit 1 and Bit 5, we can see that it shows errors. These two parities show both check 17, 19, 21, so the single bit error must be in either bit 17, bit 19 or bit 21.

However, since Parity Bit 2 checks bit 19 and Parity Bit 3 checks bit 21, this indicates that there are no errors in the subset of bits, and that therefore the error must be in bit 17.

III. Conclusion

In conclusion, Hamming code is a very useful technique used in digital communication, to avoid corruption of data. As shown in the report, Hamming code simply adds extra bits to the data words to detect and correct the error.

Calculating the number of parity bits needed is a crucial step in the Hamming code error detection and correction method. In order to find the number of parity bits needed in a 2-byte word message, the following formula is used ($2^r \geq m + r + 1$), where (m) is the number of data bits and (r) is the number of parity bits.

Additionally, in order to locate the created sender message, the parity bits are written alongside the data bits in the form of power of 2's. A final message is formed by performing the XOR operation on each bit with even parity calculation, after which each parity bit is placed into their bit position.

Furthermore, to detect the single-bit error, it is necessary to perform the XOR operation on each parity bit and then compare the parity bits of initially sent message to the parity bits of message generated. Once the error has been detected at a parity bit location, the single-bit error is discovered by inspecting the position of other data bit positions and determining whether there have been errors in other parity bit positions. By eliminating all the possibilities of non-error there should be one error left in a data bit at a certain position. Once the error is found it is easily corrected.

References

[1] Administrator. Electronics Hub. June 20, 2019 “Error Correction and Detection Codes.” [Online]. Available: https://www.electronicshub.org/error-correction-and-detection-codes/

[2] Sack, H. February 11, 2021. “Richard Hamming and the Hamming Code.” [Online]. Available: http://scihi.org/richard-hamming/

[3] Hajjdiab, H. 2023. COMP-228: System Hardware. Chapter 2: Data Representation in Computer Systems. Concordia University, Montreal, Canada, 2023. [PowerPoint].