

Hamming Codes*

Charles B. Cameron

April 19, 2005

1 The Hamming Distance

Let n be the number of bits used to encode a message. Let k be the number of message bits included in the encoding. Therefore $n - k$ is the number of bits used for error checking.

The Hamming Distance between two codes is the number of bits which differ between the two codes. For example, consider the following two 5-bit codes, where the leftmost bit is bit $5 - 1 = 4$ and the rightmost bit is bit 0: 00110 and 01010 differ in positions 4 and 3. Therefore the Hamming Distance between these two codes is 2.

In any coding scheme you choose, let d_{min} be the minimum Hamming Distance.

2 Detecting Errors

If $d_{min} = 1$, at least two valid codes differ in only one bit. If this bit is inadvertently changed in a memory or during a data transmission then it is impossible to detect the error. On the other hand, if $d_{min} = 2$, an error in one single bit guarantees that the resultant code is not valid. Why not? Because with a one-bit error it has moved a Hamming Distance 1 from the (original) correct code: no other valid code exists this close to the original code. So with $d_{min} = 2$ we can detect any single-bit error.

In general, if we want to detect l -bit errors, we need to be sure that $d_{min} \geq l + 1$.

3 Correcting Errors

Consider now the problem of correcting errors. Suppose as before $d_{min} = 2$. If a single-bit error occurs, the resultant code will have been transformed into a new, invalid code. It is impossible to know for this invalid code which of two possible

*Course notes for EE461 Microprocessor-based Digital Design

Bit															
Locations	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Check-bits								c_3				c_2		c_1	c_0
Message-bits	m_{10}	m_9	m_8	m_7	m_6	m_5	m_4		m_3	m_2	m_1		m_0		

Figure 1: Layout of Message and Check Bits in a Hamming Code

adjacent valid codes¹ was intended. Each is equidistant from the erroneous code and there is no reason to suppose one of these more likely than the other to have been intended. We cannot correct a 1-bit error with this coding scheme. If two errors occur, we cannot even detect the errors because the resultant code will be a valid, albeit unintended, code.

Suppose now that $d_{min} \geq 3$. A single-bit error will produce an invalid code with a Hamming Distance of 1 from the original code and a Hamming distance of at least 2 from any other valid code. If it is reasonable to suppose that only one error occurred, then it is reasonable to correct the invalid code to the nearest valid code.

If more than one error occurred, then we are in trouble. The result of two errors is an erroneous code which is closer to an incorrect valid code than to the correct one. If three errors occur, the erroneous code will be an incorrect valid code. This error cannot even be detected.

In general, if you want to be able to correct t -bit errors, you need to be sure that $d_{min} \geq 2t + 1$.

4 The Hamming Code

Hamming proposed an error-correcting code which could correct a 1-bit error and detect 2-bit errors. In his scheme multiple parity-check bits are included with the message bits. Any particular bit, whether part of the message or just a parity bit, would have a unique combination of check-bits associated with it.

Suppose we decide to have four check-bits, c_3 , c_2 , c_1 , and c_0 . We therefore have $n - k = 4$ because there are four check-bits. A Hamming Code sets $n \leq 2^{n-k} - 1$. In this case, then, $n \leq 2^4 - 1 = 15$. Thus $k = n - (n - k) = n - 4 \leq 15 - 4 = 11$. In other words, there are four check-bits and up to 11 message-bits in a combined total of 15 stored or transmitted bits.

In a Hamming Code, the message- and the check-bits are located at particular locations in the code. A check-bit c_i is located at position 2^i and positions are numbered starting at 1. So c_0 is at location 1, c_1 is at location 2, c_2 is at location 4, and c_3 is at location 8. This pattern is followed in any Hamming Code, no matter how many check-bits are included. The other locations are

¹Adjacent means "being separated by a Hamming Distance of 1".

Decimal Number	Binary Number
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Figure 2: Four-bit Numbers

used by message bits. Figure 1 on page 2 shows one obvious way in which the message bits could be distributed in the other bit locations. Notice that there is no column 0. We will see how we can incorporate an extra bit in column 0 shortly to make the code detect—but not correct—two-bit errors. For now, the code we are designing will only be capable of detecting and correcting one-bit errors.

In a Hamming Code, parity bit c_i is used to hold the parity bit for all bits in the code whose locations have a binary representation with a 1 in position i . Consider the table of four-bit binary numbers and their decimal equivalents shown in Figure 2. Since decimal numbers 1, 3, 5, 7, 9, 11, 13, and 15 all have a 1 in position 0 in their binary representations, c_0 is used to store parity information for each of the bits in these locations. In other words, c_0 is used to store parity information on itself and on message bits $m_0, m_1, m_3, m_4, m_6, m_8,$ and m_{10} . The calculation of an even-parity can be done by applying the

Bit Locations	12	11	10	9	8	7	6	5	4	3	2	1
Check-bits					c_3				c_2		c_1	c_0
Message-bits	m_7	m_6	m_5	m_4		m_3	m_2	m_1		m_0		
Received message bits												
$r =$ Received check-bits												
$c =$ Calculated check-bits												
$r \oplus c$												
Corrected message bits												

Corrected Message
(Hex and ASCII)

Figure 3: A Worksheet for Correcting Erroneous Messages

exclusive-or operation to each of the bits. So

$$c_0 = m_{10} \oplus m_8 \oplus m_6 \oplus m_4 \oplus m_3 \oplus m_1 \oplus m_0 \quad (1)$$

Similarly, since decimal numbers 2, 3, 6, 7, 10, 11, 14, and 15 all have a 1 in position 1 in their binary representations, c_1 is used to store parity information for each of the bits in these locations. In other words, c_1 is used to store parity information for itself and for message bits $m_0, m_2, m_3, m_5, m_6, m_9,$ and m_{10} . So

$$c_1 = m_{10} \oplus m_9 \oplus m_6 \oplus m_5 \oplus m_3 \oplus m_2 \oplus m_0. \quad (2)$$

Similarly, since decimal numbers 4, 5, 6, 7, 12, 13, 14, and 15 all have a 1 in position 2 in their binary representations, c_2 is used to store parity information for each of the bits in these locations. In other words, c_2 is used to store parity information for itself and for message bits $m_1, m_2, m_3, m_7, m_8, m_9,$ and m_{10} . So

$$c_2 = m_{10} \oplus m_9 \oplus m_8 \oplus m_7 \oplus m_3 \oplus m_2 \oplus m_1. \quad (3)$$

We can repeat this reasoning for parity check-bit c_3 and satisfy ourselves that

$$c_3 = m_{10} \oplus m_9 \oplus m_8 \oplus m_7 \oplus m_6 \oplus m_5 \oplus m_4. \quad (4)$$

What would we do if we didn't need all 11 message bits? For example, the 7-bit ASCII codes should only need message bits m_0 through m_6 . Message bits m_7 through m_{10} are unnecessary. We can simply drop them from the equations for the check bits c_i , making the equations simpler. For example, with 7-bit ASCII codes we would have $c_0 = m_6 \oplus m_4 \oplus m_3 \oplus m_1 \oplus m_0$, the same as (1) but with the terms m_{10} and m_8 omitted.

Bit Locations	12	11	10	9	8	7	6	5	4	3	2	1
Check-bits					c_3				c_2		c_1	c_0
Message-bits	m_7	m_6	m_5	m_4		m_3	m_2	m_1		m_0		
Received message bits	1	1	1	0		0	1	0		1		
$r =$ Received check-bits					0				1		0	0
$c =$ Calculated check-bits					1				0		0	0
$r \oplus c$					1				1		0	0
Corrected message bits	0	1	1	0		0	1	0		1		

Corrected Message (Hex and ASCII) 65 = 'E'

Figure 4: Example Showing How to Correct the Received Code 0xE2C

5 Correcting the Errors

Figure 3 contains a worksheet useful for correcting errors manually. The easiest way to explain how to use the worksheet is through an example. Suppose our code is used to transmit 8-bit bytes with four correction bits. In other words, each 8-bit message code is represented by a 12-bit error-correcting code.

As a specific example, assume we have retrieved the code 0xE2C. We enter the 12-bit equivalent 1110 0010 1100 into the table in the two rows labeled *Received message bits* and $r = 0100$ *Received check bits* as shown in Figure 4.

Next we recompute the check bits using equations (1), (2), (3), and (4) shown in Section 4. These bits are written in the row of Figure 4 labeled $c =$ *Calculated check bits*. The four bits are $c = c_3c_2c_1c_0 = 1000$.

Next, we compute the exclusive-or of r and c : $r \oplus c = 0100_2 \oplus 1000_2 = 1100_2 = C_{16} = 12_{10}$. The result is calculated bit by bit: bit i of 0100 is combined with bit i of 1000. The result of this operation is the location within the 12-bit code which is in error. In this case, it is location 12 (remember, for this version of the Hamming code, indices run from 1 to n , not from 0 to $n - 1$.) This means that bit 12 was received as a 1; it is in error; so it needs to be replaced by a 0.

Finally, we extract the message bits, discarding the check bits, to get the message $0110\ 0101_2 = 65_{16}$. If this message is actually an eight-bit ASCII code, it represents the character *E*.

In practice we would use either hardware or software to calculate the check bits and to alter the indicated bit.

6 Detecting Two-bit Errors

In the scheme described so far, two-bit errors cannot be reliably detected. We can extend the scheme by using column 0 for an overall parity bit. For the

15-bit code we have considered up to now, we add a sixteenth bit in column 0 and compute it by incorporating *all* the message bits as well as *all* the check bits computed so far.

$$\begin{aligned}
 p_0 = & m_{11} \oplus m_{11} \oplus m_{10} \oplus m_9 \oplus m_8 \oplus m_7 \\
 & \oplus m_6 \oplus m_5 \oplus m_4 \oplus m_3 \oplus m_2 \oplus m_1 \\
 & \oplus m_0 \oplus c_3 \oplus c_2 \oplus c_1 \oplus c_0 \oplus
 \end{aligned}$$

If a single-bit error occurs, the received and the calculated overall parity bits will not match. In this case we can use the remaining Hamming check bits as described earlier to discover which bit is in error and fix it.

When a double-bit error occurs, the received and the calculated overall parity bits will match. Only by using the remaining Hamming check bits as described earlier will we discover whether no errors or a double-bit error occurred. In the case of no errors, the indicated erroneous bit will be in column 0. But the Hamming check bits do not use column 0. In this case, we know there are no errors. On the other hand, if a two-bit error occurred, the Hamming check bits will indicate some other column than column 0. This, combined with the fact that the overall parity bit p_0 is correct, indicates that two bits are in error.

This modification to the basic coding scheme is sometimes referred to as *Single-Error Correction, Double-Error Detection* or SECDED.

References

- [1] S. Lin and J. D. J. Costello, *Error Control Coding: Fundamentals and Applications*. Englewood Cliffs, NJ: Prentice Hall, 1983.
- [2] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. New York, NY: North-Holland, 1977.
- [3] W. W. Peterson and J. E. J. Weldon, *Error-Correcting Codes*, 2nd ed. Cambridge, MA.: MIT Press, 1972.