Hamming Codes

Dr John Waldron

1 Hamming Codes

1.1 Parity

A *parity bit* can be used to detect whether a single bit error occurred during the transmission of data. Consider the following 7-bit value:

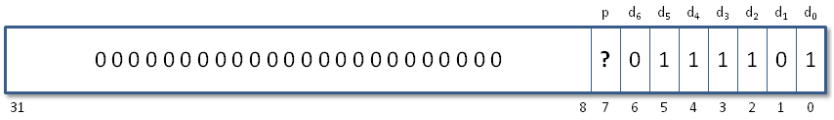

Figure 1: 7-bit value

By setting the 8^{th} bit (bit 7) of the byte such that the number of set bits (1's) is even, as shown below, we can detect whether any single bit error has occurred during the transmission (or storage and retrieval) of the value.

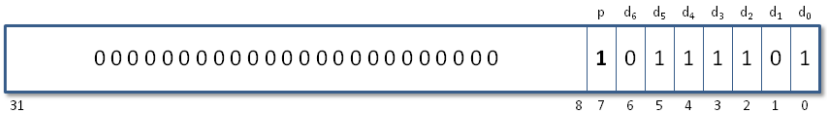

Figure 2: Even parity encoded value

For example, if we received the following even parity encoded value, we would know that an error has occurred in transmission, since the number of set bits is now odd:

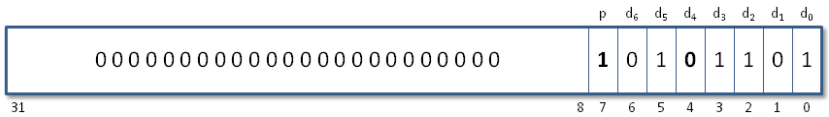

Figure 3: Even parity encoded value with error

1.2 Hamming Codes

Unfortunately, a single parity bit does not provide us with enough information to determine which of the 12 bits in the encoded value is incorrect. By using *Hamming Codes* with more than one check bit, we can not only detect whether a single bit error has occurred but also determine which bit is incorrect, thus allowing us to correct the error by inverting it. Hamming Codes are named after Richard Hamming, the American mathematician.

1.2.1 Generating Hamming Codes

The following example demonstrates how a Hamming Code that adds four *check bits*, c_0, c_1, c_2 and c_3, to an eight-bit value with data bits $d_0 \ldots d_7$ can be used to detect and correct single bit errors.
Consider the following byte value:

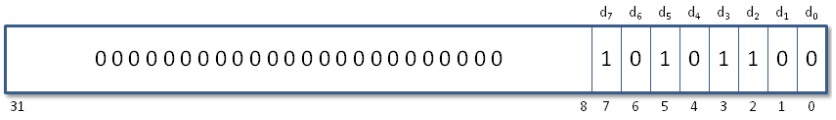

Figure 4: Byte value

Insert four check bits at bit positions 0, 1, 3 and 7. We will assume that bit position 0 is the right-most (least significant) bit position and bit position 11 is the left-most (most significant) bit position. The original eight-bit value is stored in the non-check-bit positions as follows:

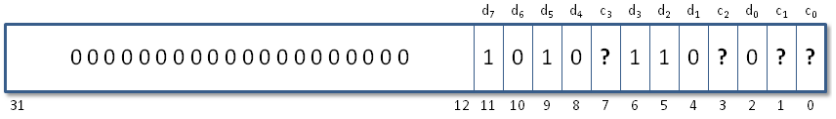

Figure 5: Hamming code check bit positions

The check bits are calculated as follows:

- Check bit c_0 in bit position 0 is calculated to produce even parity for bits 2, 4, 6, 8, and 10. (i.e. c_0 should be 1 to produce even parity for the bit sequence 0, 0, 1, 0, 0)
- Check bit c_1 in bit position 1 is calculated to produce even parity for bits 2, 5, 6, 9 and 10. (i.e. c_1 should be 1 to produce even parity for the bit sequence 0, 1, 1, 1, 0)
- Check bit c_2 in bit position 3 is calculated to produce even parity for bits 4, 5, 6 and 11. (i.e. c_2 should be 1 to produce even parity for the bit sequence 0, 1, 1, 1)
- Check bit c_3 in bit position 7 is calculated to produce even parity for bits 8, 9, 10 and 11. (i.e. c_3 should be 0 to produce even parity for the bit sequence 0, 1, 0, 1)

The key thing about Hamming Codes that can be seen from visual inspection is that any given bit is included in a unique set of parity bits. If only one parity bit indicates an error, the parity bit itself is in error. Hamming codes have a minimum distance of 3, which means that the decoder can detect and correct a single error, but it cannot distinguish a double bit error of some codeword from a single bit error of a different codeword. Thus, some double-bit errors will be incorrectly decoded as if they were single bit errors and therefore go undetected, unless no correction is attempted.
Setting each of the four check bits appropriately produces the following Hamming Code for this example:

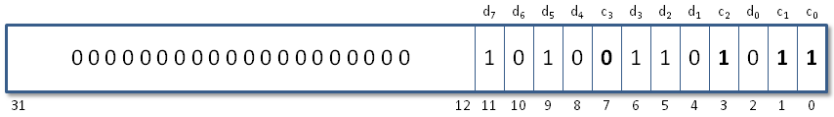

Figure 6: Example of Hamming Code check bits

1.2.2 Detecting and Correcting Errors

To detect single bit errors in a Hamming Code, the value of each of the four check bits, c_0, c_1, c_2 and c_3 is cleared and then re-calculated using the scheme described above. The check bits of the original (possibly incorrect value) can then be compared with the new check bits. If the value has changed, then one or more of the re-calculated check bits will be different. To compare the check bits, an exclusive-OR operation can be performed. The resulting value will have a 1 in any bit positions that differ and a 0 otherwise.
In the following example there is an error in bit position 8, causing check bits c_0 and c_3 to differ between the original value and the value with re-calculated check bits:

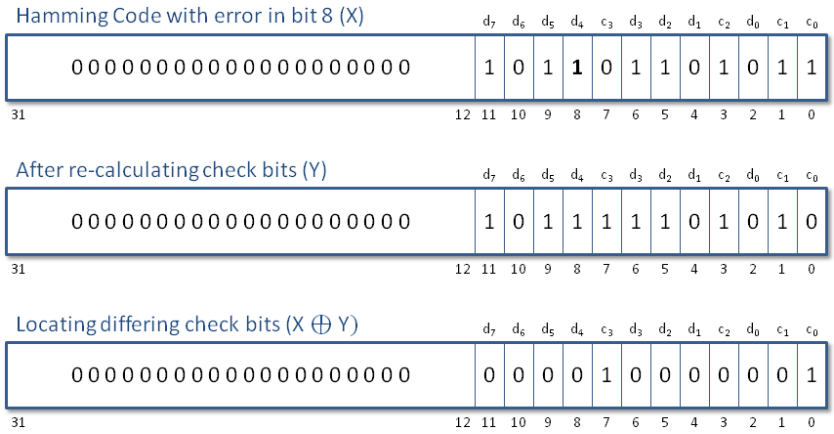

Figure 7: Comparing received check bits with re-calculated check bits

By isolating the bits of the result of the exclusive-OR operation that correspond to the check bits (i.e. bits 0, 1, 3 and 7) and subtracting 1, we get the bit position of the error, as shown below for the previous example.

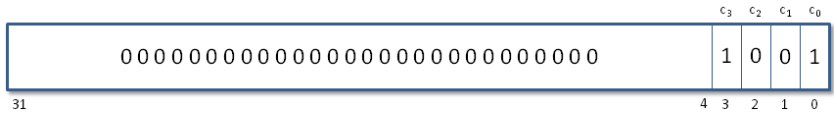

Figure 8: Determining the bit position of the error $(9 - 1 = 8)$

The error can now be corrected by inverting the incorrect bit.