https://www.tutorialspoint.com/digital\_circuits/digital\_circuits\_error\_detection\_correction\_codes.htm

Hamming Code

Hamming code is useful for both detection and correction of error present in the received data. This code uses multiple parity bits and we have to place these parity bits in the positions of powers of 2.

The **minimum value of 'k'** for which the following relation is correct validvalid is nothing but the required number of parity bits.

2^k≥n+k+1

Where,

‘n’ is the number of bits in the binary code informationinformation

‘k’ is the number of parity bits

Therefore, the number of bits in the Hamming code is equal to n + k.

Let the **Hamming code** is bn+kbn+k−1.....b3b2b1bn+kbn+k−1.....b3b2b1 & parity bits pk,pk−1,....p1pk,pk−1,....p1. We can place the ‘k’ parity bits in powers of 2 positions only. In remaining bit positions, we can place the ‘n’ bits of binary code.

Based on requirement, we can use either even parity or odd parity while forming a Hamming code. But, the same parity technique should be used in order to find whether any error present in the received data.

Follow this procedure for finding **parity bits**.

* Find the value of **p1**, based on the number of ones present in bit positions b3, b5, b7 and so on. All these bit positions suffixessuffixes in their equivalent binary have ‘1’ in the place value of 20.
* Find the value of **p2**, based on the number of ones present in bit positions b3, b6, b7 and so on. All these bit positions suffixessuffixes in their equivalent binary have ‘1’ in the place value of 21.
* Find the value of **p3**, based on the number of ones present in bit positions b5, b6, b7 and so on. All these bit positions suffixessuffixes in their equivalent binary have ‘1’ in the place value of 22.
* Similarly, find other values of parity bits.

Follow this procedure for finding **check bits**.

* Find the value of c1, based on the number of ones present in bit positions b1, b3, b5, b7 and so on. All these bit positions suffixessuffixes in their equivalent binary have ‘1’ in the place value of 20.
* Find the value of c2, based on the number of ones present in bit positions b2, b3, b6, b7 and so on. All these bit positions suffixessuffixes in their equivalent binary have ‘1’ in the place value of 21.
* Find the value of c3, based on the number of ones present in bit positions b4, b5, b6, b7 and so on. All these bit positions suffixessuffixes in their equivalent binary have ‘1’ in the place value of 22.
* Similarly, find other values of check bits.

The decimal equivalent of the check bits in the received data gives the value of bit position, where the error is present. Just complement the value present in that bit position. Therefore, we will get the original binary code after removing parity bits.

## https://www.techopedia.com/definition/24161/error-correction-code--ecc

## What Does Error Correction Code (ECC) Mean?

Error correction code (ECC) checks read or transmitted data for errors and corrects them as soon as they are found. ECC is similar to parity checking except that it corrects errors immediately upon detection. ECC is becoming more common in the field of data storage and network transmission hardware, especially with the increase of data rates and corresponding errors.

## Techopedia Explains Error Correction Code (ECC)

Error correction code is applied to data storage via the following steps:

1. When a data byte or word is stored in RAM or peripheral storage, a code-specifying bit sequence is estimated and stored. Each fixed number of bits per word has an additional fixed number of bits to store this code.
2. When the byte or word is called for reading, a code for the retrieved word is calculated according to the original algorithm and then compared to the stored byte’s extra fixed bits.
3. If the codes match, the data is error free and is forwarded for processing.
4. If the codes do not match, the changed bits are caught through a mathematical algorithm and the bits are immediately corrected.

Data is not verified during its storage period, but is tested for errors when it is requested. If required, the error correction phase follows detection. Frequent recurring errors at the same storage address indicate a permanent hardware error. In this case, the system sends the user a message, which is logged to record the error location(s).

<https://en.wikipedia.org/wiki/Hamming_code>

### General algorithm**[[edit](https://en.wikipedia.org/w/index.php?title=Hamming_code&action=edit&section=7" \o "Edit section: General algorithm)]**

The following general algorithm generates a single-error correcting (SEC) code for any number of bits. The main idea is to choose the error-correcting bits such that the index-XOR (the [XOR](https://en.wikipedia.org/wiki/Exclusive_or) of all the bit positions containing a 1) is 0. We use positions 1, 10, 100, etc. (in binary) as the error-correcting bits, which guarantees it is possible to set the error-correcting bits so that the index-XOR of the whole message is 0. If the receiver receives a string with index-XOR 0, they can conclude there were no corruptions, and otherwise, the index-XOR indicates the index of the corrupted bit.

An algorithm can be deduced from the following description:

1. Number the bits starting from 1: bit 1, 2, 3, 4, 5, 6, 7, etc.
2. Write the bit numbers in binary: 1, 10, 11, 100, 101, 110, 111, etc.
3. All bit positions that are powers of two (have a single 1 bit in the binary form of their position) are parity bits: 1, 2, 4, 8, etc. (1, 10, 100, 1000)
4. All other bit positions, with two or more 1 bits in the binary form of their position, are data bits.
5. Each data bit is included in a unique set of 2 or more parity bits, as determined by the binary form of its bit position.
   1. Parity bit 1 covers all bit positions which have the **least** significant bit set: bit 1 (the parity bit itself), 3, 5, 7, 9, etc.
   2. Parity bit 2 covers all bit positions which have the **second** least significant bit set: bits 2-3, 6-7, 10-11, etc.
   3. Parity bit 4 covers all bit positions which have the **third** least significant bit set: bits 4–7, 12–15, 20–23, etc.
   4. Parity bit 8 covers all bit positions which have the **fourth** least significant bit set: bits 8–15, 24–31, 40–47, etc.
   5. In general each parity bit covers all bits where the bitwise AND of the parity position and the bit position is non-zero.

In [computer science](https://en.wikipedia.org/wiki/Computer_science) and [telecommunication](https://en.wikipedia.org/wiki/Telecommunication), **Hamming codes** are a family of [linear error-correcting codes](https://en.wikipedia.org/wiki/Linear_code). Hamming codes can detect one-bit and two-bit errors, or correct one-bit errors without detection of uncorrected errors. By contrast, the simple [parity code](https://en.wikipedia.org/wiki/Parity_bit) cannot correct errors, and can detect only an odd number of bits in error. Hamming codes are [perfect codes](https://en.wikipedia.org/wiki/Perfect_code), that is, they achieve the highest possible [rate](https://en.wikipedia.org/wiki/Block_code#The_rate_R) for codes with their [block length](https://en.wikipedia.org/wiki/Block_code#The_block_length_n) and [minimum distance](https://en.wikipedia.org/wiki/Block_code#The_distance_d) of three.[[1]](https://en.wikipedia.org/wiki/Hamming_code#cite_note-1) [Richard W. Hamming](https://en.wikipedia.org/wiki/Richard_Hamming) invented Hamming codes in 1950 as a way of automatically correcting errors introduced by [punched card](https://en.wikipedia.org/wiki/Punched_card) readers. In his original paper, Hamming elaborated his general idea, but specifically focused on the [Hamming(7,4)](https://en.wikipedia.org/wiki/Hamming(7,4)) code which adds three parity bits to four bits of data.[[2]](https://en.wikipedia.org/wiki/Hamming_code#cite_note-FOOTNOTEHamming1950153%E2%80%93154-2)

In [mathematical](https://en.wikipedia.org/wiki/Mathematics) terms, Hamming codes are a class of binary linear code. For each integer *r* ≥ 2 there is a code with [block length](https://en.wikipedia.org/wiki/Block_code#The_block_length_n) *n* = 2*r* − 1 and [message length](https://en.wikipedia.org/wiki/Block_code#The_message_length_k) *k* = 2*r* − *r* − 1. Hence the rate of Hamming codes is *R* = *k* / *n* = 1 − *r* / (2*r* − 1), which is the highest possible for codes with minimum distance of three (i.e., the minimal number of bit changes needed to go from any code word to any other code word is three) and block length 2*r* − 1. The [parity-check matrix](https://en.wikipedia.org/wiki/Parity-check_matrix) of a Hamming code is constructed by listing all columns of length *r* that are non-zero, which means that the [dual code](https://en.wikipedia.org/wiki/Dual_code) of the Hamming code is the [shortened Hadamard code](https://en.wikipedia.org/wiki/Hadamard_code). The parity-check matrix has the property that any two columns are pairwise [linearly independent](https://en.wikipedia.org/wiki/Linear_Independence).

Due to the limited redundancy that Hamming codes add to the data, they can only detect and correct errors when the error rate is low. This is the case in computer memory (usually RAM), where bit errors are extremely rare and Hamming codes are widely used, and a RAM with this correction system is a ECC RAM ([ECC memory](https://en.wikipedia.org/wiki/ECC_memory)). In this context, an extended Hamming code having one extra parity bit is often used. Extended Hamming codes achieve a Hamming distance of four, which allows the decoder to distinguish between when at most one one-bit error occurs and when any two-bit errors occur. In this sense, extended Hamming codes are single-error correcting and double-error detecting, abbreviated as **SECDED**.

<https://www.xilinx.com/support/documentation/application_notes/xapp383.pdf>

<https://www.hindawi.com/journals/jece/2019/3905094/>