

Computer Engineering

Electronic and Communication Systems

Error correction code

Project Report

Federica Perrone

Summary

[Introduction 3](#_Toc89956711)

[General Algorithm 3](#_Toc89956712)

[Applications 4](#_Toc89956713)

[Possible Architecture 4](#_Toc89956714)

[Architecture description 5](#_Toc89956715)

[VHDL Code 6](#_Toc89956716)

[Test-plan 7](#_Toc89956717)

[Synthesis 8](#_Toc89956718)

[Conclusion 9](#_Toc89956719)

# Introduction

An error correcting code is an algorithm for expressing a sequence of numbers such that any errors, which are introduced, can be detected and corrected (within certain limitations) based on the remaining numbers.

The error correcting codes are used for controlling errors in data over unreliable or noisy communication channels.

The central idea is the sender encodes the message with redundant information in the form of an ECC. The redundancy allows the receiver to detect a limited number of errors that may occur anywhere in the message, and often to correct these errors without retransmission.

The two main categories of ECC codes are block codes and convolutional codes.

Hamming codes are a family of linear error-correcting block codes. Hamming codes can detect one-bit and two-bit errors, or correct one-bit errors without detection of uncorrected errors. Richard W. Hamming invented Hamming codes in 1950 as a way of automatically correcting errors introduced by punched card readers.

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.

Hamming codes have a minimum distance of three, which means that the decoder can detect and correct a single error, but it cannot distinguish a double bit error of some code word from a single bit error of a different code word. 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.

To remedy this shortcoming, Hamming codes can be extended by an extra parity bit. This way, it is possible to increase the minimum distance of the Hamming code to four, which allows the decoder to distinguish between single bit errors and two-bit errors. Thus the decoder can detect and correct a single error and at the same time detect (but not correct) a double error.

This extended Hamming code is popular in computer memory systems, where it is known as SECDED (abbreviated from single error correction, double error detection).

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.

## 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 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.

* 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.
* Parity bit 2 covers all bit positions which have the second least significant bit set: bits 2-3, 6-7, 10-11, etc.
* Parity bit 4 covers all bit positions which have the third least significant bit set: bits 4–7, 12–15, 20–23, etc.
* Parity bit 8 covers all bit positions which have the fourth least significant bit set: bits 8–15, 24–31, 40–47, etc.
* In general each parity bit covers all bits where the bitwise AND of the parity position and the bit position is non-zero.

If a byte of data to be encoded is 10011010, then the data word (using \_ to represent the parity bits) would be \_\_1\_001\_1010, and the code word is 011100101010.

The choice of the parity, even or odd, is irrelevant but the same choice must be used for both encoding and decoding.

## Applications

## Possible Architecture

# Architecture description

# VHDL Code

# Test-plan

# Synthesis

# Conclusion