1 Executive Summary

Modern flash memory is reliant on error-correcting code to ensure error-free operation. Current flash memory architectures often use linear block codes for this purpose (e.g. Reed-Solomon, Hamming or BCH codes). However, the recent rediscovery of LDPC (Low Density Parity Check) codes, which can achieve superior performance close to the Shannon Limit, has generated much interest in the NAND memory industry. These codes could be used to further improve error correction capability in flash memory, thus allowing for more densely packed memory cells and thus larger capacity drives.

The general aim of this project is to produce a MATLAB simulation of how an error correction system using LDPC codes would work for flash memory. By combining both an error generation and an error correction model, it will be possible to benchmark these rediscovered codes, and subsequently compare them to current generation technologies.

Contents

2 Introduction

3 Overview of Linear Block Codes

Linear block codes are one of the two main classes of forward error correction (FEC), with the other main type being convolutional coding. A linear block code essentially takes a block of binary data, and adds additional redundant data onto it. This block can then be transmitted over a noisy channel, and subsequently decoded at the receiver. The redundant bits in the block are used as parity check equations, which allows a linear block code to both detect and correct errors.

3.1 Definitions for Linear Block Codes

All linear block codes can be described using a set of standard terms and symbols. For this project, all codes use a binary alphabet of $\{0, 1\}$ and hence all operations are over this binary field. n is the block length, the total size of the output codeword. k is the message length, the size of the information vector prior to encoding. An (n, k) error correcting *code* $\mathcal{C}$, will produce a set of 2^k output *codewords* $\mathbf{c}$. Hence, $\mathbf{c} \in \mathcal{C}$.

The rate of any linear block code, R, is defined as:

$$R = \frac{k}{n} \tag{3.1.1}$$

The rate is a measure of the number of information bits compared to the total number of transmitted codeword bits. A high rate code will be more efficient in terms of useful information transmitted, but will have a poorer error correction capability. In Flash Memory, very high rate ($R > 0.9$) codes are used in order to maximise the amount of usable storage space. Conversely, an example use of low rate codes would be in deep-space probe transmissions, where receiving error-free data is more important than rate of transmission.

A linear block code can be represented in two ways: through the $k \times n$ generator matrix $\mathbf{G}$, or the $(n - k) \times n$ parity check matrix $\mathbf{H}$. Each is the null-space of the other, such that:

$$\mathbf{G}\mathbf{H}^\top = 0 \tag{3.1.2}$$

Unsurprisingly, the generator matrix $\mathbf{G}$ is used in the transmission side when encoding data, and the

parity check matrix $\mathbf{H}$ is used at the receiver to detect and correct any errors.

At the transmitter, if we take a $1 \times k$ input vector of binary data $\mathbf{x}$, the method of encoding this data into a codeword $\mathbf{c}$, is simply a multiplication operation:

$$\mathbf{c} = \mathbf{x}\mathbf{G} \tag{3.1.3}$$

At the receiver, a similar operation is performed:

$$\mathbf{s} = \mathbf{c}\mathbf{H}^{\top} \tag{3.1.4}$$

where $\mathbf{s}$ is known as the *syndrome*. If the syndrome is the all zero vector, then error free transmission has occurred. Conversely, if any bit of the syndrome is 1, then this represents a particular error pattern. For small block lengths, these error pattern's can be pre-calculated and saved as a table, allowing for syndrome lookup decoding. The table identifies the exact location of a bit error in the codeword, which can then be 'flipped' in order to perform error correction.

An example of a *systematic* generator matrix, in this case a matrix known as the *Hamming(7,4) code*, takes the form:

$$\mathbf{G} = \left(\begin{array}{ccc|cccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right) \tag{3.1.5}$$

with the first three columns labelled $\underbrace{}_{n-k}$ and the last four labelled $\underbrace{}_{k}$.

In this code, 4 information bits are encoded into 7 output bits. Notice the identity matrix in the right portion of the generator matrix. This means that the 4 message (k) bits are always encoded at the end of the codeword, with the parity check ($n-k$) bits at the start of the codeword. This is why it is *systematic*. At the decoder, it is then easy to extract the (uncorrected) message bits from the codeword, simply by looking at the last 4 bits.

3.2 Hamming weight, distance, and error correction capability

An important metric when discussing error correcting codes is the concept of the Hamming weight. For any codeword, the Hamming weight is defined as the total number of non-zero elements in a given codeword. Another metric, the minimum weight (w_{min}), is simply the minimum value from the set of all

Hamming weight's for a given code, excluding the all-zero case.

Example: A fictional example (6,2) code could have the following codewords:

Message (2 bits)	Codeword (6 bits)	Hamming weight
0 0	0 0 0 0 0 0	0
0 1	1 0 1 0 1 0	3
1 0	0 0 1 1 0 0	2
1 1	1 0 0 1 1 0	3

For this code, the minimum weight (w_{min}) is 2, since that is the smallest value of the Hamming weight's excluding the all-zero case.

Another metric used is the Hamming distance. The Hamming distance defines how 'close' any two codewords are to each other. Codewords that are 'far away' from each other are less likely to be decoded in error, and hence the Hamming distance determines how 'good' a code is at error detection and correction. Formally, the Hamming distance is defined as the number of (binary) places that any 2 codewords differ. Analogous to the minimum weight, there is also a minimum distance (d_{min}), which is the minimum value from the set of all Hamming distances for a given code, excluding the trivial case of comparing a codeword to itself.

An important result arises because of the use of binary arithmetic, in that the minimum Hamming weight is in fact equal to the minimum Hamming distance:

$$d_{min} = w_{min} \tag{3.2.1}$$

It is now possible to present the results that describe, for linear block codes, their error correction and detection performance:

Error Detection Theorem: *A linear block code with minimum weight w_{min} is able to detect up to e errors:*

$$e_{detectable} = w_{min} - 1 \tag{3.2.2}$$

Error Correction Theorem: *A linear block code with minimum weight w_{min} is able to correct up to e errors:*

$$e_{correctable} = \frac{w_{min} - 1}{2} \tag{3.2.3}$$

3.3 Low Density Parity Check codes

A particular class of codes, known as "Low Density Parity Check" (LDPC) codes, are of particular interest and relevance to this project. LDPC codes are generally considered to be some of the best performing linear block codes available, in terms of error performance, with some codes getting within a fraction of the Shannon Limit. Additionally, LDPC codes have no patent and hence no licensing costs, making them attractive for real world use.(??!).

LDPC codes were originally discovered by R.G. Gallager in 1962. Then known as "Gallager codes", they were defined by a sparse parity check matrix with low column weights. Gallager also worked on a probabilty based decoding method for these codes, which proved to have promising performance. However for various reasons, these codes were essentially lost in favour of other more practical codes available at the time. It is possible that the decoding complexity for LDPC was, at the time, too great for the computational power then available.[1]

Modern LDPC codes were re-discovered by J.C. MacKay in 1996. MacKay demonstrated that LDPC codes could be decoded using probabilistic methods, even beyond the bound set by their minimum distance. Today, LDPC codes are seeing a resurgence in various applications. Most notably in the DVB-S2 standards for digital HD satellite broadcast, 10GBase-T Ethernet and as optional 'add-ons' to the 802.11n/ac wi-fi standards.

Disadvantages of LDPC include the fact that there still exists a small (often in the region of 10^{-6} to 10^{-9}) probability of error after decoding, known as the 'error-floor'. This can be avoided by using a second high rate, 'inner' Error Correcting Code such as BCH or Reed-Solomon to remove the last few bit errors. Other issues include decoding complexity. Whilst decoding time is linear with block length, decoding using the belief propagation algorithm is still problematic, especially for low power mobile devices. Most applications of LDPC so far have been on mains-powered equipment.

Example: A specific DVB-S2 code, with $n = 64800$ and $r = 0.9$, has a total of $194,399$ non-zero elements in it's parity check matrix $\mathbf{H}$. However, the non-zero elements account for just 0.04% of the total matrix: The vast majority of $\mathbf{H}$ is empty. It is therefore easy to see why they are called "Low Density". In this project, this specific high-rate code is used extensively, especially when modelling the memory-specific case in section ?????????.

[1] Personal opinion. Even today, decoding LDPC using near-optimum belief propagation on a PC is computationally expensive, whilst on dedicated ASIC hardware consumes large amounts of power.