Second Order Reed-Muller Decoding Algorithm in Quantum Computing

Minji Kim

June 2, 2011

Contents

Abstract

The goal of this paper is to construct a quantum algorithm to decode second order Reed-Muller codes. We derive a way to apply the second order Reed Muller code in the quantum applications, just as the way it is done for the first order Reed-Muller code. In order to fully understand the relationship between the classical model and the quantum model of the Reed Muller decoder, we first analyze different algorithms to decode the first order Reed Muller decodes and realize that there exists an efficient quantum algorithm called "Hadamard Decoding Algorithm". Building on the first order Reed Muller decoder, we derive a method to decode the second order Reed-Muller code with a higher decoding probability.

1 Introduction

In information theory, the original message sent is often distorted by noise from the channel and thus the received message is different from the original message. To decode the original message, we use the error-correcting codes and apply to the distorted message received. The basic process is : message goes through an encoder, then the channel (where the noise distorts messages), and then a decoder. If the coding is successful, then the decoded message will be identical to the original message sent.

$Notations.$
The message is a binary number with length n. This is a n bit string of either "0" or "1". RM(r,n-1) is a Reed-Muller code of order r, and length n-1.It is an operation mapping n $\rightarrow 2^{n-1}$ bits.
Hamming Distance, denoted by $d_{hamming}$ is defined to be the number of positions at which the corresponding symbols are different. For Reed Muller codes, $d_{hamming}$=2^{n-1-r}. So for the first order Reed-Muller code, where $r = 1$, $d_{hamming}$=2^{n-2}.
Not all codes are correctly decodable. If the channel has too much noise and distorts too many bits, then a decoder cannot correctly guess the original message. For Reed-Muller codes, the number of correctable errors=$2^{n-3} - 1$. [3]

1.1 How to identify the order

Let $x_1, ...x_m$ be binary variables, then a Boolean monomial $p = x_1^{r_1} x_2^{r_2} ... x_m^{r_m}$, where $r_i \in 0, 1$and $1 \leq i \leq m$, since $x_i x_j = x_j x_i$ and$x_i^2 = x_i$

The degree of p is the number of variables in p. So, for example, $q = x_1 + x_2 + x_1x_2 + x_2x_3x_4$ has degree 3. We now focus on the first order Reed-Muller code with length n, so RM(1,n-1).

2 First order Reed Muller Code

We first look at the existing algorithms for decoding the first order Reed-Muller codes. With thorough examples and explanations, we examine how to encode and decode a simple message and what methods are available. This paper will briefly explain the classical version (Majority Decoding Algorithm) and give a detailed explanations for the quantum version (Hadamard Decoding Algorithm). The first order terms are all linear and do not have any quadratic or cubic terms such as x_1x_2 or $x_2x_3x_4$.

2.1 Encoder

Let the message = $m_{n-1}m_{n-2}...m_1m_0$ and create a matrix

$$\left.\begin{array}{c c}
1 & \\
x_1 & \\
x_2 & \\
\vdots & \\
x_{n-1} &
\end{array}
\underbrace{\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
 & & & \vdots & & & & \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0
\end{bmatrix}}_{2^{n-1}}\right\} n$$

Then encoded message $M_e = m_{n-1} + m_{n-2}x_1 + \ldots + m_0x_{n-1}$
Again, n is the length of the message.
Example:
Let's encode the message '0110'. Here, the message length n=4.
So the encoder matrix for RM(1,n-1)=RM(1,3)is:

$$\begin{array}{c} 1 \\ x_1 \\ x_2 \\ x_3 \end{array}
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0
\end{bmatrix}$$

Then for the message '0110' the most significant bit corresponds to m_3, second bit corresponds to m_2, third bit corresponds to m_1, and the last bit corresponds to m_0.

Hence encoded message

$$\begin{aligned} M_e &= m_3 \cdot 1 + m_2 \cdot x_1 + m_1 \cdot x_2 + m_0 \cdot x_3 \\ &= 0 \cdot (11111111) + 1 \cdot (11110000) + 1 \cdot (11001100) + 0 \cdot (10101010) \\ &= 11110000 + 11001100 = 00111100. \end{aligned}$$

So M_e=00111100.
Matlab code for 1st order encoder of any length is in the Appendix - section 4.1.

2.2 Noise

When the message goes through the noisy channel, some of the message bits get flipped. In Reed-Muller code, the noise can flip up to $2^{n-3}-1$ bits and still be able to detect and correct errors, i.e., "1" becomes "0" or "0" becomes "1". For example, in the case the message is '0110', we can correct up to $2^{n-3}-1 = 2^{4-3}-1 = 2-1$ bit errors. Both the Majority decoding algorithm and the Hadamard decoding algorithm for the first order guarantee to correct up to $2^{n-3}-1$ errors (bit flips). [3]
Matlab code for Noise (flipping $2^{n-3}-1$ random bits) is included in the Appendix - section 4.2.

2.3 Majority Decoding Algorithm

Currently, there exists a classical Reed-Muller Decoding algorithm for any order codes- first order, second order, etc. This method is called "Majority decoding algorithm" and the basic idea behind it is based on the distance between vectors (also called Hamming distance). The Hamming distance between any two vectors is the number of places in the two vectors that have different values. The decoder assumes that the closest codeword in RM(r,n-1) to the received message is the original encoded message. The decoder checks each row of the encoded matrix and uses majority logic to determine whether that row was used in forming the encoded message. So it is possible to determine what the original message was. [1] The author has coded a majority decoding algorithm on Matlab and the code is attached in the Appendix - section 4.3.

2.4 Hadamard Decoding Algorithm

Now we consider another decoding algorithm, called the Hadamard decoding algorithm for Reed-Muller codes. This code is based on Hadamard matrices. This method is different from the Majority decoding algorithm in a way that it requires fewer steps and that it measures the distance between code words to determine the original message. Conveniently, this is the reason that there exists an efficient quantum implementation of the Hadamard decoding algorithm, but there isn't one for the Majority decoding algorithm. Here is how the algorithm works.

Definition: Hadamard transform H_m is a 2^m x 2^m matrix.

$H_0 = 1$

$H_m = \frac{1}{\sqrt{2}} \begin{pmatrix} H_{m-1} & H_{m-1} \\ \\ H_{m-1} & -H_{m-1} \end{pmatrix}$ for m>0

For example,

$H_0 = +1$

$H_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$

$H_2 = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$

.

.

.

$(H_n)_{i,j} = \frac{1}{2^{n/2}}(-1)^{i \cdot j}$, where i and j are bit string vectors.

Let $k = k_{m-1}2^{m-1} + k_{m-2}2^{m-2} + ... + k_1 2 + k_0$

and $n = n_{m-1}2^{m-1} + n_{m-2}2^{m-2} + ... + n_1 2 + n_0$

Then $i \cdot j$ is equivalent to $\sum_j k_j n_j$.

The Hadamard decoding algorithm utilizes these Hadamard transform matrices. A message of length 4, let's say $m_3m_2m_1m_0$, gets encoded to M_e=$M_{e7}M_{e6}M_{e5}M_{e4}M_{e3}M_{e2}M_{e1}M_{e0}$. Notice that the encoded message is either 0 or 1, but the Hadamard transform ma-

trices consist of 1 or -1. Thus, we convert 0 to 1 and 1 to -1 and get F.
The mathematical formula is as follows:

$$F = [(-1)^{M_{e7}}, (-1)^{M_{e6}}, ...(-1)^{M_{e0}}].$$

Then we transform the message using the Hadamard transform matrix.
Let $\hat{F} = FH$, where H is the Hadamard matrix of the appropriate size. Using the examples, we can conclude that the appropriate Hadamard matrix size for a message of length n is H_{n-1}. Then the message decoded is the position of the entry of $\hat{F}$ with the largest magnitude. Again, the Hadamard decoding algorithm can correct up to $2^{n-3} - 1$ errors.
As an example, let us now decode the same message as in the Majority decoding algorithm example, namely, the messae '0110'. As before, the encoded message $M_e = [00111100]$ and the message length n=4.
We then convert 0 to 1 and 1 to -1.

$$F = (-1)^{M_{e7}}(-1)^{M_{e6}}...(-1)^{M_{e0}}.$$

So F=[1 1 -1 -1 -1 -1 1 1].
The appropriate Hadamard transform matrix is:

$$H_3 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

Then the Hadamard transform $\hat{F}$ is simply the product of F and H.

$$\begin{aligned} \hat{F} &= FH \\ &= [1\,1\,-1\,-1\,-1\,-1\,1\,1]H \\ &= [0\,0\,0\,0\,0\,0\,8\,0]. \end{aligned}$$

Because 8 is the largest magnitude entry of the matrix $\hat{F}$, we look at the position and realize that it's 6 (the position is counted from 0 instead of 1). 6 in binary is '0110', which indeed is our original message. Thus, the message is successfully

decoded (with no noise, of course). Hadamard decoding algorithm would correct up to 1 error in the case message length is 4. So even if any one of the bits from M_e=[0 0 1 1 1 1 0 0] is flipped, the decoder would still give out '0110'. The Hadamard decoding algorithm Matlab code is attached in the appendix section 4.4.
$General\ idea\ of\ the\ proof$. When there is no noise, the problem is simple. Since the rows of a Hadamard matrix are orthogonal, when we take the product of one of the rows to the encoded message, then the magnitude would be 2^{n-1} at the position of the message bit and 0 everywhere else. If there is noise, then the magnitude of the product would be greatest at the position of the message bit (but less than 2^{n-1} with some small values elsewhere.

2.5 Why does Hadamard decoding algorithm have an efficient quantum implementation?

Now we examine how the first order Hadamard classical decoding algorithm can be implemented as a quantum decoding algorithm.
Quantum state is determined by vector complex whose norm^2=1. A quantum algorithm is a step-by-step procedure, where each steps can be performed on a quantum computer. It is usually described by a quantum circuit which acts on some input qubits and terminates with a measurement. [4] A qubit is either a 0, or a 1, or a superposition of both. There are two ways to make a measurement. First, the complete measurement is where we pick a basis and take the magnitude of the vector. The second is POVM, which will be discussed later in this paper.
We let $\psi = \frac{1}{\sqrt{2^n}} \sum_{i=1}^{2^{n-1}} (-1)^{Me_i} |i\rangle \in (C^2)^{\otimes n} = C^{2^n}$
C^2 is a quantum analog of a bit, namely a qubit. And $(C^2)^{\otimes n}$ is n qubits with unitary transformations.
$|i\rangle | i = 0, ..., 2^n - 1$ and $Me_i = (M_{en-1}...M_0 = Me)$
$H^{\otimes n}$=$H \otimes H \otimes ... \otimes H$
Then $\psi\prime = H^{\otimes n}\psi$
This is the quantum algorithm, where each transforms are quantum operations. This quantum algorithm is much more efficient than the classical algorithm, because instead of creating a big Hadamard transform matrix, the quantum algorithm makes n single qubit operations.

2.6 Summary of Number of Operations

We see that Quantum algorithms take in less number of operations than others, and thus makes the circuit complexity simpler and faster to process.
Here, $N = 2^n$ where n is the length of the message.

Method	Number of operations
Hadamard Transform	N^2
Fast Hadamard Transform	$NlogN$
Quantum Fast Hadamard Transform	$logN$

Table 1: Different methods and its number of operations. [2]

3 Second Order Reed Muller Code

Now we finally examine the Reed-Muller decoding algorithms for second order terms. These include both the first order terms, and any quadratic terms as well, such as x_1x_2 orx_1x_3. The difficulty arises when these quadratic terms show up, since the Hadamard decoding algorithm for the 1st order does not apply to second order anymore. The classical majority decoding algorithm still holds for second order, but our goal is to develop a quantum decoding algorithm for second order Reed-Muller codes.

3.1 Encoder

The encoder for the second order Reed-Muller codes is similar to the encoder for the first order, except that now we have to consider all products of two linear terms.
For example, RM(2,3) will have an encoder matrix of the following:

$$
\begin{aligned}
x_0 &= \begin{bmatrix}1 & 1 & 1 & 1 & 0 & 0 & 0 & 0\end{bmatrix}\\
x_1 &= \begin{bmatrix}1 & 1 & 0 & 0 & 1 & 1 & 0 & 0\end{bmatrix}\\
x_2 &= \begin{bmatrix}1 & 0 & 1 & 0 & 1 & 0 & 1 & 0\end{bmatrix}\\
x_0x_1 &= \begin{bmatrix}1 & 1 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}\\
x_0x_2 &= \begin{bmatrix}1 & 0 & 1 & 0 & 0 & 0 & 0 & 0\end{bmatrix}\\
x_1x_2 &= \begin{bmatrix}1 & 0 & 0 & 0 & 1 & 0 & 0 & 0\end{bmatrix}
\end{aligned}
$$

And the encoder is the following (only the right of the vertical line is the encoder matrix):

$$
\begin{array}{cccccc|cccccccc}
x_1x_2 & x_0x_2 & x_0x_1 & x_2 & x_1 & x_0 & & & & & & & & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0\\
 & & \vdots & & & & & & & & \vdots & & & \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0
\end{array}
$$

For RM(2,3) Encoder matrix is:

$$
\begin{aligned}
f_{ij}(x) &= i_2x_2 + i_1x_1 + i_0x_0 + j_{01}x_0x_1 + j_{02}x_0x_2 + j_{12}x_1x_2\\
&= i_0 \cdot 11110000 + i_1 \cdot 11001100 + i_2 \cdot 10101010\\
&\quad + j_{01} \cdot 11000000 + j_{02} \cdot 10100000 + j_{12} \cdot 10001000
\end{aligned}
$$

The encoder matrix for RM(2,3) is 64 by 8. Matlab code for RM(2,3) is in section 4.5

3.2 Results for on RM(2,3)

The following results are for a particular second order Reed-Muller code of length 4.

As mentioned in the earlier section, the second way of making a measurement is POVM (Positive Operator Valued Measurement). It is a measure of whose values are non-negative self-adjoint operators on a Hilbert space. [6]

Definition: POVM

A POVM is a set of Hermitian positive semidefinite operators Fi on a Hilbert space H that sum to unity,

$$\sum_{i=1}^{n} F_i = I_H.$$

This formula is similar to the decomposition of a Hilbert space by a set of orthogonal projectors:

$$\sum_{i=1}^{N} E_i = I_H, E_iE_j = \delta_{ij}E_i$$

[4]

Now we introduce a specific type of POVM, called PGM (Pretty Good Measurement).
Definition: Pretty Good Measurement (PGM)
Let σ_j be a set of quantum states, written as density matrices. That is, if $|\psi_j\rangle$ is a set of pure states, $\sigma_j = |\psi_j\rangle\langle\psi_j|$. Let $\sigma = \sum \sigma_j$.
Then the PGM for this set of states is the set of positive operaters Ej,

$$E_j = \sigma^{-1/2}\sigma_j\sigma^{-1/2}.$$

Theorem: for RM (2,3), Success Probability for PGM=$\sum_j Trace(\sigma_j E_j) = \frac{1}{8}$
Proof:
Let x_m= binary encoded messages (with 0,1)
And let

$$\psi = \frac{1}{\sqrt{8}} x_m|_{\pm 1}$$

$$\phi_m = \frac{1}{\sqrt{2^n}} \psi_m$$

Then

$$\begin{aligned}
SuccessProbabilityforPGM &= \frac{1}{2^{2n}} \sum |\langle \phi_m | \psi_m \rangle|^2 \\
&= \frac{1}{2^{2n}} \sum_m \frac{1}{\sqrt{2^n}} |\langle \psi_m | \psi_m \rangle|^2 \\
&= \frac{1}{2^{2n}} \frac{1}{2^n} \sum_m |\langle \psi_m | \psi_m \rangle|^2 \\
&= \frac{1}{2^{3n}} |\langle \psi_m | \psi_m \rangle|^2 \\
&= \frac{1}{2^9} 2^6 \\
&= \frac{1}{2^3} \\
&= \frac{1}{8}
\end{aligned}$$

We then compare this result (1/8 success probability) to just pure Hadamard decoding algorithm for the 1st order. In other words, we treated the 2nd order encoder matrix as the first order because the length of each row is the same as that of the 1st order, and got the following result:
8 vectors with probability $\frac{1}{8}$
48 vectors with probability $\frac{1}{32}$
8 vectors with probability 0
Thus, the overal probability

$$p = \frac{8/8 + 48/32 + 0}{64} = \frac{5}{128} \approx \frac{1}{32}$$

.
Success probability for PGM 1/8 gives approximately 3.2 times better result than blindly applying Hadamard decoding algorithm to the 2nd order.

3.3 Generalized 2nd order

Notation. Let $N = 2^n$, let $m = n + \binom{n}{2}$, and let M=2^m. Let $f_i : Z_N \to Z_2$ be the N-bit codeword for message $i \in Z_2^m$. Let $O_f : |x\rangle|b\rangle \to |x\rangle|b + f_i(x)\rangle$ act unitarily on $C^N \otimes C^2$.

3.4 Single Query

This function produces M vectors in an N¡M dimensional space, so the maximum average probability of identifying a message i is N/M.[5]
Proposition:

$$\frac{1}{M}\sum_{i=1}^{M-1}|\psi_i\rangle\langle\psi_i| = I_N\frac{1}{N}$$

Proof: Let

$$\psi_{i,j} = \frac{1}{\sqrt{N}}\sum_{x=0}^{2^n-1}(-1)^{f_{ij}(x)}|x\rangle.$$

Here, i is the n bit string codeword of the coefficient of the linear terms, and j is the $\binom{n}{2}$ bit string codeword of the coefficients of the quadratic terms.
The function $f_{ij}(x)$ creates the encoder matrix.

$$f_{ij}(x) = f_{i_0,i_1,\ldots i_{n-1}j_{01},\ldots,j_{n-2n-1}}(x_{n-1}\ldots x_0)$$

.
For example, for encoding RM(2,3),

$$\begin{aligned} f_{ij}(x) &= i_2x_2 + i_1x_1 + i_0x_0 + j_{01}x_0x_1 + j_{02}x_0x_2 + j_{12}x_1x_2 \\ &= i_0\cdot 11110000 + i_1\cdot 11001100 + i_2\cdot 10101010 \\ &\quad +j_{01}\cdot 11000000 + j_{02}\cdot 10100000 + j_{12}\cdot 10001000 \end{aligned}$$

Then

$$f_{ij}(x) = \sum_{k=0}^{n-1} i_kx_k + \sum_{0\le a\le b\le n-1} j_{ab}x_ax_b$$

Using these definitions of encoding function,

$$\begin{aligned} \frac{1}{M}\sum_{i,j}|\psi_{ij}\rangle\langle\psi_{ij}| &= \frac{1}{MN}\sum_{i,j}\sum_{x,y}(-1)^{f_{ij}(x)+f_{ij}(y)}|x\rangle\langle y| \\ &= \frac{1}{MN}\sum_{x,y}(\sum_{i,j}(-1)^{f_{iy}(x)+f_{iy}(y)})|x\rangle\langle y| \end{aligned}$$

When $x = y$,

$$\frac{1}{MN}\sum_{x,y} M = \frac{1}{N}$$

.
If $x \neq y$, then one of the linear terms is nonzero. Then $(-1)^{f_{ij}(x)+f_{ij}(y)}$ has just as many +1 as -1, and thus the sum is zero. .
So when $x \neq y$,

$$\frac{1}{MN}\sum_{x,y}(\sum_{i,j}(-1)^{f_{iy}(x)+f_{iy}(y)})|x\rangle\langle y| = 0$$

.
So this N x N matrix has 1/N on the diagonal entries, and zero everywhere else.
Therefore,

$$\frac{1}{M}\sum_{i=1}^{M-1}|\psi_i\rangle\langle\psi_i| = I_N\frac{1}{N}$$

3.5 Double Query

Because a single query followed by a PGM maximizes the success probability, we cannot do better by asking multiple sequential queries using the same query register. So of just one query, we know consider double query, namely, two parallel queries. This is done by creating two of the same queries.
Definition:

$$v_i = (\frac{1}{\sqrt{N}}\sum_{x\in\Re_N}(-1)^{f_i(x)}|x\rangle)^{\otimes 2}$$

In n=3, 64 vectors span 29 dimensional space, which means that the best we can do is 29/64.
Success probability of PGM= 28.4/64 < 29/64. The theorem states that if the sum of the density operators are proportional to the identity matrix, then the PGM is optimal. [5] However, in double query for n=3 case, the sum of density operators were close to the identity matrix, but not quite. It had zeros almost everywhere except for the diagonal entries, but still had a few non-zero values. Thus, it is reasonable that the success probability of PGM doesn't reach as much as the optimal value.

So for n=2, dim. space is 7; n=3, dim. space is 29; n=4, dim. space is 211. Thus we make a combinatorical argument for the generalized length n.
Proposition: dim(span$v_i|i \in Z_2^m$) = $2^{n-1}(2^n-1)+1$.
Proof: We will prove that dim(span$w_i|i \in Z_2^m$) = $2^{n-1}(2^n-1)$ and argue that dim(span$v_i|i \in Z_2^m$) is only one more than dim(span$w_i|i \in Z_2^m$), because v_i has x=y terms, whereas w_i doesn't have x=y terms.

Now we consider leaving out the terms x=y, because they are not adding any additional information. Consider the coefficients of $|x\rangle|x\rangle$ in v_i. They are the same, 1/N, for all i, so they don't contain any information about i. So we leave them out of the query and get the following:

$$w_i = \frac{1}{\sqrt{N^2-N}} \sum_{x \neq y \in Z_N} (-1)^{f_i(x)+f_i(y)} |x\rangle|y\rangle$$

.
Proposition:

$$\sum |w_{ij}\rangle\langle w_{ij}| \propto I$$

Proof:
Some notations.

$$\frac{1}{M}\sum |w_{ij}\rangle\langle w_{ij}| = \frac{1}{N^2-N} I_{N^2-N}$$

.

$$\sum |w_i\rangle\langle w_i| = \frac{1}{\sqrt{N^2-N}} \sum_{i,j} \sum_{x \neq y \in Z_N} (-1)^{f_{iy}(x)+f_{iy}(y)} |x\rangle|y\rangle \sum_{u \neq v \in Z_N} (-1)^{f_{iy}(u)+f_{iy}(v)} \langle u|\langle v|$$

When $x=u$ and $y=v$, the above equals a constant c for all x,y
When $x \neq u$, OR $y \neq v$, the above equals 0 for all x,y,u,v.
When x=u, y=v,

$$\begin{aligned}\sum |w_i\rangle\langle w_i| &= \frac{1}{\sqrt{N^2-N}}\sum_{i,j}(-1)^{2f_{ij}(x)+2f_{ij}(y)}|x\rangle|y\rangle\langle x|\langle y| \\ &= \frac{1}{\sqrt{N^2-N}}\sum_{i,j}1|x\rangle|y\rangle\langle x|\langle y| \\ &= \frac{M}{\sqrt{N^2-N}}|x\rangle|y\rangle\langle x|\langle y|.\end{aligned}$$

Thus, $\sum |w_{ij}\rangle\langle w_{ij}| \propto I$

Proposition: dim(span$w_i|i \in Z_2^m$) = $2^{n-1}(2^n-1)$
. Proof: w_i is an N(N-1) by M marix.
Because the in the rows (x,y) are identical to (y,x) the rank is at most

$$\begin{aligned}\frac{N(N-1)}{2} &= \frac{2^{2n}-2^n}{2} \\ &= 2^{2m-1}-2^{n-1} \\ &= 2^{n-2}(2^n-1).\end{aligned}$$

So the rank $\leq 2^{n-1}(2^n-1)$.

Showed that $\leq$ - still need to show that it's =.

4 Appendix

Matlab Codes

4.1 RM 1st order Encoder

```
function [sum] = rmencoder( message )
m=length(message)-1;
encoder = zeros(m+1,2^m);
encoder(1,:) = ones(1,2^m);
```

```
mess(1,1)=message(1)-48;
for i = 1:m

    for j = 1:2^(m-i+1):2^m
        encoder(i+1,j:j+2^(m-i)-1) = ones(1,2^(m-i));
    end

    mess(1,i+1)=message(i+1)-48;

end

for i = 1:m+1
    enc(i,:) = mess(i)*encoder(i,:);
end
sum = zeros(1,length(enc));
for i = 1:length(enc(:,1));
    sum = xor(sum,enc(i,:));
end

end
```

4.2 RM Noise

```
function [ Me ] = rmnoise( Mc )

%To get Me= rmnoise(rmencoder('whatever message i want')

t=length(Mc);
C=2^t;
n=log2(t) + 1;
while(C>2^(n-3)-1)
    B=round(rand(1,t));
    C=sum(B);
end

Me = xor (Mc, B);

end
```

4.3 RM Majority Decoder

```
function [decmsg] = rmdecodermajority1(Me)

% n=length of message=m+1;
% Mc=encoded matrix
% # of errors correctable = 2^(n-3) - 1
% Me=message after the errors

t=length(Me);
n=log2(t) + 1;
m=n-1;

% 1. that one matrix we formed for encoding with 1, x1, x2, etc.

encoder = zeros(m+1,2^m);
encoder(1,:) = ones(1,2^m);

for i = 1:m
    for j = 1:2^(m-i+1):2^m
        encoder(i+1,j:j+2^(m-i)-1) = ones(1,2^(m-i));
    end
end

% 2. Xn= every combinations of xj and xi where i!=j and j < n, i < n

for i=1:m
    x{i,1}=encoder(1+i,:);
    %now "bars" for "2"
    x{i,2}=ones(1,2^m)-x{i,1};
end

perms=zeros(m,m-1);
y=[1:m];
for i=1:m
    perms(i,:)=y(y~=i);
end
```

```
tempbin=dec2bin([0:2^(m-1)-1],m-1);

for i=1:m % Number of Xs
    for k=1:length(tempbin) % Number of tempbin
        if (str2num(tempbin(k,1))==0)
            temp(k,:)=x{perms(i,1),1};
        else
            temp(k,:)=x{perms(i,1),2};
        end
        for l=1:length(tempbin(1,:))
            if (str2num(tempbin(k,l))==0)
                temp(k,:) = and(temp(k,:),x{perms(i,l),1}); %=x2
            else
                temp(k,:) = and(temp(k,:),x{perms(i,l),2});
            end
        end
        c{i}=temp;
    end
end

% 3. R(Xn) = Me * Xn loop

for i=1:m
    Rx{i}=mod(Me*c{i}',2);
end

% 4. R(Xn)*xn + R(Xn-1)*xn-1 + R(Xn-2)*xn-2 etc. = My

for i=1:m
    if (sum(Rx{i}) > length(Rx{i})/2)
        decoded(1,i)=1;
    else
        decoded(1,i)=0;
    end
end

My = 0;
```

```
for i=1:m
    My = My+decoded(1,i)*x{i};
end
My = mod(My,2);

% 5. My + Me = Mye
Mye = mod(My+Me,2);

% 6. If Mye has more 0s than 1s —> R(x0)=0; 1 otherwise
if (sum(Mye) > length(Mye)/2)
    decoded=[1 decoded];
else
    decoded=[0 decoded];
end

% 7. Encoded message = R(xn)*xn R(xn-1)*xn-1 .... R(x0)

decmsg = '';
for i=1:length(decoded)
    decmsg = strcat(decmsg,num2str(decoded(1,i)));
end

end
```

4.4 RM Hadamard Decoder

```
function [ Ff ] = rmdecoderhadamard( Me )

%n=length of message=m+1;


t=length(Me);
n=log2(t) + 1;
m=n-1;

%converting 0, 1 to 1, -1

F= ((-2)*Me(:)+ 1)';
```

```
%forming Hadamard Transform

H=[1, 1; 1, -1];
 for i=2:m
     H=[H H; H -H];
 end

 Ff=F*H;
%Getting the original message
 Ff=abs(Ff);
 [z,p] = max(Ff);
 p=p-1;
 dec2bin(p,n);
 end
```

4.5 RM(2,3) Encoder

```
x1= [ -1 -1 -1 -1 1 1 1 1 ]; % x1
x2= [-1 -1 1 1 -1 -1 1 1]; % x2
x3= [-1 1 -1 1 -1 1 -1 1]; % x3
x4= [-1 -1 1 1 1 1 1 1]; % x1x2
x5= [-1 1 -1 1 1 1 1 1]; % x1x3
x6= [-1 1 1 1 -1 1 1 1]; % x2x3

x=[x6' x5' x4' x1' x2' x3']';

rm2encoder=ones(2^6, 2^3);
rm2encoder(1,:)=ones(1, 2^3);
rm2encoder(1,:)= -1.*rm2encoder(1,:);

%tempbin=dec2bin([0:63], 6);

d = (0:63)';
b = de2bi(d, 'left-msb');
%disp('     Dec              Binary          ')
%disp('     -----      -------------------')
%disp([d, b])
```

```
tempbin=b;

    for i=1:64
         for j=1:6
        if tempbin(i,j) == 1
      rm2encoder(i,:)= rm2encoder(i,:).*x(j,:);
        end
         end
    end
```

4.6 PGM of RM(2,3)

```
rm8dim=zeros(8,8);
for k=1:64
    rm8dim=rm8dim+rm2encoder(k,:)'*rm2encoder(k,:);
end
rm8dim
eig(rm8dim)


%%%%Finding PGM (Pretty Good Measurement)%%%%
psim=(1/sqrt(8))*rm2encoder;
phim=(1/sqrt(8))*psim;
P=0;
for l=1:64
    P=P+(phim(l,:)*psim(l,:)')^2;
end
P=1/64*P
```

4.7 Hadamard Transform on RM(2,3)

```
L=zeros(64,8);

NEWrm2=(rm2encoder -1)/(-2);

for i=1:64
    L(i,:)=rmdecoderhadamard(NEWrm2(i,:));
```

```
end
```

4.8 Two queries RM(2,3)

```
twoqueries=zeros(64,64);
for j=1:64
    twoqueries(:,j)=1/8*kron(rm2encoder(j,:),rm2encoder(j,:));
end

rank(twoqueries)

%%finding sum of density operators of 64 states (hope it's identity)

rm64dim=zeros(64,64);
for k=1:64
    rm64dim=rm64dim+twoqueries(:,k)*twoqueries(:,k)';
end
rm64dim=1/64*rm64dim;
%eig(rm64dim)

%finding the PGM for two queries case
P=0;
[v,d]=eig(rm64dim);

for i=1:64
    if d(i,i)>0
        E(i,i)=1/d(i,i);
    else
        E(i,i)=0;
    end
end
E=sqrt(real(E));
rhominushalf=v*E*v';
for m=1:64
    P=P+(abs((twoqueries(:,m)'*(1/8)*rhominushalf*twoqueries(:,m))))^2
end

P=(1/64)*P
```

5 Acknowledgements

First and foremost, I want to thank my advisor David Meyer for his continuous support and guiding me through the research. Without his guide, this paper would not have been possible.
I'd also like to thank Asif Shakeel for helping me with the numerical analysis of the second order Reed-Muller codes, and clarifying many concepts.

6 References

[1] B. Cooke. "Reed-Muller Error Correcting Codes"
[2] A. Barg and S. Zhou. "A Quantum decoding algorithm of the simplex code"
[3] T. Moon. ECE 7670 Lecture9: Reed-Muller Codes
[4] M. Nielsen and I. Chuang. "Quantum Cmputation and Quantum Information."
[5] D. Meyer and J. Pommersheim. "Single-query Learning from Abelian and Non-Abelian Hamming Distance Oracles"
[6] J. Preskill, "Lecture Note for Physics: Quantum Information and Computation"