Enhanced Decoding of Triple Error Correction Reed-Muller Codes to Reduce Silent Data Corruption in Memories

Costas Argyrides, Pedro Reviriego, Costas Kokkinos and Juan Antonio Maestro

Received: date / Accepted: date

Abstract
Memories are typically protected with Error Correction Codes (ECCs) to prevent soft errors from causing data corruption. Codes that can correct one error in a memory word are commonly used. Those are known as Single Error Correction (SEC) codes. In addition to error correction, for memory applications, it is also important to detect errors that cannot be corrected to avoid Silent Data Corruption (SDC). For that reason, Single Error Correction Double Error Detection (SEC-DED) codes that can also detect double errors are preferred. As technology scales, there is an increased interest in the use of more advanced ECCs to protect memories. Among those, Reed-Muller (RM) codes have been proposed due to their efficient decoding. In this paper a modified decoding algorithm for RM codes is proposed. This algorithm, in addition to error correction provides error detection when the number of correctable errors is exceeded by one.

Keywords majority logic decoding · Reed-Muller codes · error correction codes · memory.

1 Introduction

Reliability is a major issue for advanced circuits. Soft errors are one of the main concerns as they change the logical values of memory cells [1]. To avoid data corruption, Error Correction Codes (ECCs) have been used for many years to protect memories from soft errors [2]. Single Error Correction Double Error Detection (SEC-DED) codes that can correct one error and detect two errors per memory are commonly used [3],[4]. They require few additional bits per word and the decoding process is simple. A SEC-DED code has a minimum Hamming distance of four between any two coded words. To perform single error correction only, a Hamming distance of three is enough. The additional distance enables double error detection. With a distance of four any word that suffers a double error would be in the worst case at a distance of two of any valid coded word. This means that it cannot be mistaken for single error and miss-corrected. The same reasoning applies to codes that can correct two bit errors. In this case Double Error Correction Triple Error Detection (DEC-DED) codes are used. Those codes have a minimum distance of six. In general, a code that can correct $t$ errors and detect $t + 1$ errors requires a minimum distance of $2 \cdot (t + 1)$. The ability to detect errors that exceed the error correction capability of a code is crucial to avoid Silent Data Corruption (SDC). SDC can cause an unexpected behavior of systems as the memory contents have been modified but the system is not aware of the situation and continues its normal operation [5].

In recent years, Multiple Cell Upsets (MCUs) that corrupt more than one memory bit have become increasingly common [6]. The use of larger memories also poses a challenge in some environments, such as space, where the error rate is high [7]. To cope with those issues, the use of more advanced ECCs to protect memories has been proposed [7],[8],[9],[10]. For example, the use of Bose-Chaudhuri-Hocquenghem (BCH) [11], Euclidean Geometry (EG) [12] and Different Set (DS) [13] codes has been recently proposed for memory protection. Reed-Muller (RM) codes have also been used for memory protection [14]. RM codes are multiple step Majority Logic Decodable (MLD) and can be implemented efficiently. They also have word sizes that
are a power of two and match typical memory word widths. In this paper an enhanced decoding algorithm for Triple Error Correction (TEC) Reed-Muller codes is presented. The new algorithm performs error detection as part of the first step of majority logic decoding. The proposed scheme is analyzed theoretically and by simulation to show how it can be effective in detecting uncorrectable errors.

The rest of the paper is organized as follows, in section 2 a brief overview of RM codes is provided. Then in section 3, the modified decoding algorithm is presented and analyzed. In section 4, the proposed decoder is validated and its implementation cost is discussed. Finally, the conclusions of the paper are summarized in section 5.

2 REED-MULLER CODES

Reed-Muller (RM) codes are a class of multiple error correction codes which are rich in structural properties [15]. For any integers \( r \) and \( m \) such that \( 0 \leq r \leq m \), there exists a binary \( r^{th} \) order RM code with the following parameters:

- Word length \( n = 2^m \).
- Number of data bits \( k = 1 + \binom{m}{1} + \binom{m}{2} + \ldots + \binom{m}{r} \).
- Minimum distance \( d_{\text{min}} = 2^{m-r} \).
- Number of correctable errors \( t_{\text{ML}} = 2^{m-r}-1 \).

RM codes are typically denoted by \( RM(r,m) \). Some examples of codes that can be useful for memory applications are shown in Table 1.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( k )</th>
<th>( r )</th>
<th>( m )</th>
<th>Correctable Errors ( t_{ML} )</th>
<th>Minimum Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>11</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>64</td>
<td>42</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>128</td>
<td>99</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>256</td>
<td>219</td>
<td>5</td>
<td>8</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>512</td>
<td>466</td>
<td>6</td>
<td>9</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

As discussed in the introduction, RM codes are multiple step majority logic decodable. This enables an efficient implementation of the decoder. In particular an \( RM(r,m) \) code can be decoded in \( r+1 \) steps of majority logic decoding [15]. In each of the steps some bits are decoded so that at the end of the process the complete word has been decoded. The proposed decoding algorithm introduces changes in the first step of majority logic decoding. Therefore a basic understanding of how this first step works is needed.

Let us assume an \( RM(r,m) \) code with \( n = 2^m \). Then each bit or position in a code word can be represented as a binary number with \( m \) bits. In the first step of decoding, combinations of \( m-r \) bits are selected. There are \( \binom{m}{m-r} \) such combinations and each one is used to decode a bit. For each of them, a set of XOR equations is formed and the decoded bit is given by the majority value among those equations. The number of equations is given by \( 2^{m-r} \). Let us take one of the combinations of \( m-r \) bits \( \{i_1,i_2,...,i_{m-r}\} \) then, the equations are built by selecting a value for each of those bits. The positions that participate in an equation are those that have those values in bits \( \{i_1,i_2,...,i_{m-r}\} \) and any possible combination in the remaining \( r \) bits. This means that each equation involves \( 2^r \) positions. This is better illustrated with an example. Let us consider the \( RM(3,6) \) code. In this case \( m-r = 3 \) and there are \( \binom{6}{3} = 20 \) combinations of three bits. One such combination is the following group of bits \( \{4,5,6\} \). When bits 4, 5, 6 are zero an equation is obtained that checks positions of the following form 000:xx in binary format. This corresponds to the following positions \( \{1,2,3,4,5,6,7,8\} \).

The rest of the equations are shown in Table 2. Then a majority vote of the equations is done to determine the decoded bit. In the absence of errors, all equations will give the same value. If there are three or less bits in error the errors are corrected as they cannot invert the majority vote. However, when there are four bits in error, there is a tie in the majority and the decision of whether the bit is one or zero cannot be taken reliably [15] therefore an error in that bit is likely to occur. The same process is done for the twenty combinations of three bits, therefore twenty bits are decoded in the first step of majority logic decoding.

Then, the word on which the decoding algorithm operates is modified using the results from the first step [15] and decoding proceeds to the second step of majority logic decoding. This means that errors not corrected in the first step propagate to the subsequent steps and can lead to data corruption [15]. The second step decodes another group of bits and the word on which the decoding algorithm operates is modified again using those bits. The process continues until all steps are completed. A more detailed explanation of the decoding of RM codes is out of the scope of this paper but can be found in [15].

3 MODIFIED MAJORITY LOGIC DECODING

In this section, the modified majority logic decoding algorithm is presented. As discussed before the objective is to detect uncorrectable errors. Let us consider the \( RM(3,6) \) code. In this case, since the minimum code distance is eight, the code can theoretically correct three errors and detect four errors. During the first step of majority logic decoding an error that affects four bits can be detected and identified as uncorrectable by checking if there is a tie in the majority vote for any of the sets of equations. For example, for the set of equations in Table 2 an error that affects positions 1, 9, 17 and 25 will cause a tie in the majority vote.
However an error on bits 1, 2, 9, 17 will not cause a tie in the majority vote. Therefore it is important to be certain that a tie will occur in at least one of the sets of equations to ensure that all possible errors affecting four bits are identified as uncorrectable. This is guaranteed for some values of \( r \) and \( m \) by this theorem: Theorem “For a RM\((r,m)\) code, any error affecting \( t_{MLE} + 1 = 2^{m-r-1} \) bits will result in a tie on at least one set of equations during the first step of majority logic decoding provided that \( m-r \) is equal or smaller than three.”

Proof: The proof starts with the observation that when \( m-r \) is equal or smaller than three then \( m-r \geq 2^{m-r-1} - 1 \). For a \( RM\((r,m)\) code, the sets of equations are obtained by selecting a combination on \( m-r \) bits. If there is any such combination for which the positions in error have all different values then each position in error will be checked by a different equation and there will be a tie on the majority vote. Since \( m-r \) is at least the number of correctable errors, the bits can be selected as follows. First select a bit that is different in the first and second positions in error. Then if for the third position in error that bit is equal to that in position one (two), select a bit that is different in positions one (two) and three. For the third bit, check if the first and second bits already selected are the same in position four and in any of the previous. If so, select a bit that is different in position four and the one that matches the value of the first and second bit. This process continues until all positions have been covered. This is possible since \( m-r \geq 2^{m-r-1} - 1 \). The combination of bits selected in this way will result in a tie in the majority vote. To illustrate the process, an example is discussed in the following. Let us consider the \( RM\((3,6)\) code with errors on bits 1, 2, 9, 17. Then the first bit is selected to be bit 1 such that it is different for position 1 and 2. Since for position 9 that bit is equal to position 1, the second bit is bit 4 which is different in positions 1 and 9. Finally bits 1 and 4 are the same in positions 1 and 17. Therefore bit 5 that is different in positions 1 and 17 is selected as the third bit. The combination of bits \{1, 4, 5\} is different for all the positions in error. Therefore for the set of equations that corresponds to bits \{1, 4, 5\} there will be a tie in the majority vote. This example is illustrated in Figure 1. The minimum distance and the number of correctable errors for the codes for which the theorem applies are shown in Table 3.

The proposed scheme requires a modification of the decoder to check if there is a tie in the majority logic vote. This requires additional circuitry which is the same for all \( RM\((r,m)\) codes as all have eight equations on the first step of majority logic decoding. The modified computation for one of the bits decoded in the first step is shown in Figure 2. The majority vote is a circuit that outputs a one if the majority of its inputs is one and a zero if the majority of its inputs is zero. It can be implemented using sorting networks as proposed in [12]. The tie detection circuit outputs a one if and only if the number of inputs that have a value of one is the same as the number of inputs having a value of zero.

<table>
<thead>
<tr>
<th>m-r</th>
<th>Correctable Errors ((t_{MLE}))</th>
<th>Minimum Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

4 VALIDATION AND IMPLEMENTATION

The proposed approach has been implemented for the \( RM\((3,6)\) code and tested by exhaustively generating all the \((\binom{64}{4}) = 635376\) possible error combinations affecting four bits. The results have shown that the errors are identified as uncorrectable in all cases. This guarantees that when a word suffers four bit errors no Silent Data Corruption (SDC) would occur.

Finally, a logical OR of the tie detection outputs of all the bits decoded in the first step (20 for the \( RM\((3,6)\) code) is computed to detect uncorrectable errors. When an uncorrectable error is detected decoding is stopped and the error condition signaled.
5 CONCLUSIONS

In this paper, an enhanced decoding algorithm for some Reed-Muller codes has been presented. The algorithm is capable of detecting errors that exceed the error correction capability of the code by one. This is useful to reduce silent data corruption in memory applications. The proposed algorithm has been analyzed theoretically and validated by simulation. Finally, the method has been implemented for the $RM(3, 6)$ code showing that it has no impact on decoding time but requires additional circuitry. This means that the memory access time will not increase with the proposed scheme.

Table 4 AREA AND DELAY ESTIMATES FOR MAJORITY LOGIC DECODING

<table>
<thead>
<tr>
<th>Logic Type</th>
<th>Area (µm²)</th>
<th>Power (mW)</th>
<th>Majority Delay (ns)</th>
<th>Tie Detection Delay (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Majority logic</td>
<td>116.1</td>
<td>69.1</td>
<td>0.59</td>
<td>n.a.</td>
</tr>
<tr>
<td>Majority logic and tie detection</td>
<td>400.0</td>
<td>418.3</td>
<td>0.59</td>
<td>0.80</td>
</tr>
</tbody>
</table>

As mentioned before, the tie detection outputs from the twenty sets of equations have to be combined with OR gates. This adds a cost per set of equations of $6.6µm²$ and an additional delay of $0.34ns$. The area cost is negligible and the delay does not impact the decoding time as it can also be done in parallel with the rest of the steps of majority logic decoding. Finally it is worth mentioning that the area and power overheads in percentage over the conventional decoder for the complete decoder will be smaller than that in Table 4. This is because the results in Table 4 are for the majority logic and tie detection only while rest of the components are the same in the proposed and conventional decoder. In summary, the implementation of the proposed technique would increase the area and power consumption of the decoder but would not affect latency.

References