XAPP715 (v1.0) November 15, 2004

Multiple Bit Error Correction

Author: Simon Tam

Summary

In high-reliability aerospace, avionics, and military applications, single error correction (SEC) and double error detection (DED) may not provide adequate protection against SDRAM memory faults. This makes multiple-error correction (MEC) highly desirable. Although many powerful error control methods including Reed-Solomon are capable of correcting multiple bytes of error, the general drawback with these methods is latency and speed. Most of these codes require at least several dozen cycles to complete the first correction. Additional latency is not appealing to most memory interface applications. Fortunately, Reed-Muller error control codes possess multiple bit error correction capability with relatively low latency and high performance. In this application note, the triple error correcting *Reed-Muller* (RM) is implemented in both the Virtex-II Pro™ and Virtex-4™ Platform FPGA families.

Introduction

In high-reliability applications, memory can sustain multiple soft errors due to single or multiple event upsets caused by environmental factors (cosmic neutrons, alpha particles, etc.). The traditional Hamming code with SEC-DED capability can not address these types of errors. It is possible to use powerful non-binary BCH code such as Reed-Solomon code to address multiple-bit errors. However, it could take at least a couple dozen cycles of latency to complete the first correction and run at a relatively slow speed.

This application note explores the possibility of using *Reed-Muller* (RM) code in memory interface applications to address multiple-bit soft errors. RM code is one of the oldest error correction codes belonging to the Finite Geometry family. Due to its orthogonal structure, it is relatively easy to decode using the *Majority-Logic Decoding* (MLD) method (see Reference Design). The following section is a brief explanation of the construct and decoding of simple RM code. For details and the mathematical proof of the RM code consult Reference Item 2.

An r^{th} order Reed-Muller code RM(r,m) is the set of all binary strings of length $n = 2^m$ associated with the Boolean polynomials $p(x_1; x_2;\ldots; x_m)$ of degree at most r. A Boolean polynomial is a linear combination of Boolean monomials. A Boolean monomial p in the variables $x_1, x_2, \ldots, x_m$ is the expression of the form:

$$p = x_1^{r_1} x_2^{r_2} \ldots x_m^{r_m} \text{ where } r_i \in \{0,1,2..\} \text{ and } 1 \le i \le m.$$

The degree of a monomial is deduced from its reduced form (after rules $x_i x_j = x_j x_i$ and $x_i^2 = x_i$ are applied), and it is equal to the number of variables. This rule extends to polynomials. Example of a polynomial of degree 3:

$$q = x_1 + x_2 + x_1 x_2 + x_1 x_2 x_3$$

For example, the first order RM(1,3) code word size is $2^3 = 8$ with single bit error correction capability. It has up to one in three variables: $\{1, x_1, x_2, x_3\}$ in each monomial as follow:

$q_0 = 1$	$q_1 = 1 + x_1$	$q_2 = 1 + x_2$
$q_3 = 1 + x_3$	$q_4 = 1 + x_1 + x_2$	$q_5 = 1 + x_1 + x_3$
$q_6 = 1 + x_2 + x_3$	$q_7 = 1 + x_1 + x_2 + x_3$	

In the RM encoder, the code word is created by the following matrix multiplication:

$$C = M \bullet G$$

Where M is the original message matrix, G is the generator matrix and C is the resulting code word. The generation of the RM(1,3) code word C is described as:

$$C = M \bullet G = \begin{bmatrix} M_3 & M_2 & M_1 & M_0 \end{bmatrix} \begin{bmatrix} 1 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix}$$

The RM(1,3) generator matrix is:

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The matrix multiplication results in the following encoder equations where $\otimes$ denotes an Exclusive-OR operation.

$$C_7 = M_3$$
$$C_6 = M_3 \otimes M_2$$
$$C_5 = M_3 \otimes M_1$$
$$C_4 = M_3 \otimes M_2 \otimes M_1$$
$$C_3 = M_3 \otimes M_0$$
$$C_2 = M_3 \otimes M_2 \otimes M_0$$
$$C_1 = M_3 \otimes M_1 \otimes M_0$$
$$C_0 = M_3 \otimes M_2 \otimes M_1 \otimes M_0$$

To decode an incoming code word C' back to its original message M', each message bit M'_i is determined based on the majority of the corresponding orthogonal checksums $S_{i,k}$ generated from the incoming code word C'. In this case, there are four checksums for each original message bit M'_2, M'_1, and M'_0.

The orthogonal checksums for decoding the original message are shown in Table 1.

Table 1: **Orthogonal Checksums**

M'_0	M'_1	M'_2
$S_{0,3} = C'_4 \otimes C'_0$	$S_{1,3} = C'_2 \otimes C'_0$	$S_{2,3} = C'_1 \otimes C'_0$
$S_{0,2} = C'_6 \otimes C'_2$	$S_{1,2} = C'_6 \otimes C'_4$	$S_{2,2} = C'_5 \otimes C'_4$
$S_{0,1} = C'_5 \otimes C'_1$	$S_{1,1} = C'_3 \otimes C'_1$	$S_{2,1} = C'_3 \otimes C'_2$
$S_{0,0} = C'_7 \otimes C'_3$	$S_{1,0} = C'_7 \otimes C'_5$	$S_{2,0} = C'_7 \otimes C'_6$

The majority rules are simple, if more than two checksums result in a "1", the original message bit is "1". If more than two checksums result in a "0", the original message bit is "0".

In the case of equal number of checksums resulting in 1s and 0s, the original message bit is undetermined. In other words, it has reached the correcting limit of this code. However, it is important to note, such an event also indicates the presence of quadruple error. The decoder should flag this as a warning. Furthermore, any one group of the checksums can detect quadruple error independently.

To determine M'_3, another partial code word C" needs to be constructed based on the result of M'_2, M'_1, and M'_0. C" is derived from the following equations:

$$C''_7 = 0$$

$$C''_6 = M'_2$$

$$C''_5 = M'_1$$

$$C''_4 = M'_2 \otimes M'_1$$

$$C''_3 = M'_0$$

$$C''_2 = M'_2 \otimes M'_0$$

$$C''_1 = M'_1 \otimes M'_0$$

$$C''_0 = M'_2 \otimes M'_1 \otimes M'_0$$

Once C' is determined, add the original code word with C" forming the checksum S_3:

$$S_3 = C' + C''$$

S_3 is eight bits long. The same majority rule applies. If more than four bits are 1s, M'_3 is "1" and if more than four bits are 0s, M'_3 is "0".

The following is an example of the code in practice. Assume the message is {`0101`}. The resulting code word C is {`01011010`}. Let C_2 be the corrupted bit. The code word becomes {`01011110`}. The orthogonal checksums $S_{i,k}$ are shown in Table 2.

Table 2: **Example Orthogonal Checksums $S_{i,k}$**

M'_0	M'_1	M'_2
$S_{0,3} = 1$	$S_{1,3} = 1$	$S_{2,3} = 1$
$S_{0,2} = 0$	$S_{1,2} = 0$	$S_{2,2} = 1$
$S_{0,1} = 1$	$S_{1,1} = 0$	$S_{2,1} = 0$
$S_{0,0} = 1$	$S_{1,0} = 0$	$S_{2,0} = 1$

Taking the majority of the checksums, the message bits are $M'_0 = 1$, $M'_1 = 0$, and $M'_2 = 1$. Based on this result, C" is {`01011010`}. Add C" with the original code word C' {`01011110`}. S_3 becomes {`00000100`}. Hence, $M'_3 = 0$. In summary, the original message is {`0101`} and the error position is C_3 indicated by S_3.

The individual message bit M'_2, M'_1, M'_0 decoding is done independent from others except for M'_3. In this case, a total of two stages are needed to decode the entire message. The decoding logic is relatively simple. This allows RM code to be fast and low in latency compared to equivalent cyclic code.

Second Order Reed-Muller Code

This reference design utilizes a second order 5th degree RM code to achieve multiple bit error correction. The message width of RM(2,5) code is 16 bits and the code word is 32 bits. There are five variables: X_1, X_2, X_3, X_4, X_5. It can correct at most three random error bits and detect four random error bits. The generator matrix [G] is defined as:

$$G = \begin{bmatrix} 0 \\ G_1 \\ G_2 \end{bmatrix} \qquad G_1 = \begin{bmatrix} 0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \end{bmatrix} \qquad G_2 = \begin{bmatrix} X_1X_2 \\ X_1X_3 \\ X_1X_4 \\ X_1X_5 \\ X_2X_3 \\ X_2X_4 \\ X_2X_5 \\ X_3X_4 \\ X_3X_5 \\ X_4X_5 \end{bmatrix}$$

$$G = \begin{bmatrix}
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\
0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \\
0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&0&0&0&1&1&0&0&1&1 \\
0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1 \\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\
0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1 \\
0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&1 \\
0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \\
0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1 \\
0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1 \\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1
\end{bmatrix}$$

The code word is generated similar to RM(1,3) mentioned previously. For example, code word bit 12 is generated as:

$$C_{12} = M_{15} \otimes M_{14} \otimes M_{13} \otimes M_{10} \otimes M_9 \otimes M_6 \otimes M_3$$

For RM(2,5) code, decoding is accomplished in three stages. The first stage consists of eight checksums for each message bit from M_9 to M_0. As an example, the checksums for M_0 are:

$$\begin{aligned}
S_{0,7} &= C'_{24} \otimes C'_{17} \otimes C'_8 \otimes C'_0 \\
S_{0,6} &= C'_{25} \otimes C'_{17} \otimes C'_9 \otimes C'_1 \\
S_{0,5} &= C'_{26} \otimes C'_{18} \otimes C'_{10} \otimes C'_2 \\
S_{0,4} &= C'_{28} \otimes C'_{20} \otimes C'_{12} \otimes C'_4 \\
S_{0,3} &= C'_{27} \otimes C'_{19} \otimes C'_{11} \otimes C'_3 \\
S_{0,2} &= C'_{29} \otimes C'_{21} \otimes C'_{13} \otimes C'_5 \\
S_{0,1} &= C'_{30} \otimes C'_{22} \otimes C'_{14} \otimes C'_6 \\
S_{0,0} &= C'_{31} \otimes C'_{23} \otimes C'_{15} \otimes C'_7
\end{aligned}$$

Majority vote is taken to decide if the message bit is 0 or 1, similar to the previously described majority-rule method. For example, in first stage, if there are five or more equations yielding "1", then the corresponding message bit is "1". Likewise, the message bit is "0" if five or more equations yield a "0". If there are four equations yielding "1" and four equations yielding "0", it indicates a quadruple error. Hence, the original message can not be correctly decoded and the result is an unknown message bit.

Second stage decoding operates on the intermediate code word C''. It is created from the decoded message bits from the first decoding stage:

$$C'' = C' - [M_9 \ldots M_0]\,[G_2]$$

In this equation, C' is the original incoming code word, G_2 is the lower portion of the generator matrix. Second-stage checksum generator creates checksums based on the partial code word C''. There are sixteen checksums for each message bit from M_{14} to M_{10}. The same majority rules apply.

$$\begin{aligned}
S_{10,15} &= C''_{16} \otimes C''_0 & \qquad S_{10,7} &= C''_{17} \otimes C''_1 \\
S_{10,14} &= C''_{24} \otimes C''_8 & \qquad S_{10,6} &= C''_{25} \otimes C''9 \\
S_{10,13} &= C''_{20} \otimes C''_4 & \qquad S_{10,5} &= C''_{21} \otimes C''_5 \\
S_{10,12} &= C''_{28} \otimes C''_{12} & \qquad S_{10,4} &= C''_{29} \otimes C''_{13} \\
S_{10,11} &= C''_{18} \otimes C''_2 & \qquad S_{10,3} &= C''_{19} \otimes C''_3 \\
S_{10,10} &= C''_{26} \otimes C''_{10} & \qquad S_{10,2} &= C''_{27} \otimes C''_{11} \\
S_{10,9} &= C''_{22} \otimes C''_6 & \qquad S_{10,1} &= C''_{23} \otimes C''_7 \\
S_{10,8} &= C''_{30} \otimes C''_{14} & \qquad S_{10,0} &= C''_{31} \otimes C''_{15}
\end{aligned}$$

The final stage is for decoding M_{15}. It works on the intermediate code word C''' and is derived from:

$$C''' = C'' - [M_{14} \ldots M_{10}]\,[G_1]$$

The final stage does not have a checksum generator. The partial code word C''' is a 32-bit wide vector. Apply similar majority rules on these bits directly to determine the correct state of M15.

Reference Design

The reference design consists of two components: the encoder and the decoder. They work independently as far as each component concerns. They operate based on the RM(2,5) code mentioned in previous section.

Encoder

The encoder takes 16-bit message and encodes into 32-bit code word based on the RM(2,5) matrix multiplication. Figure 1 shows a block diagram of the encoder.

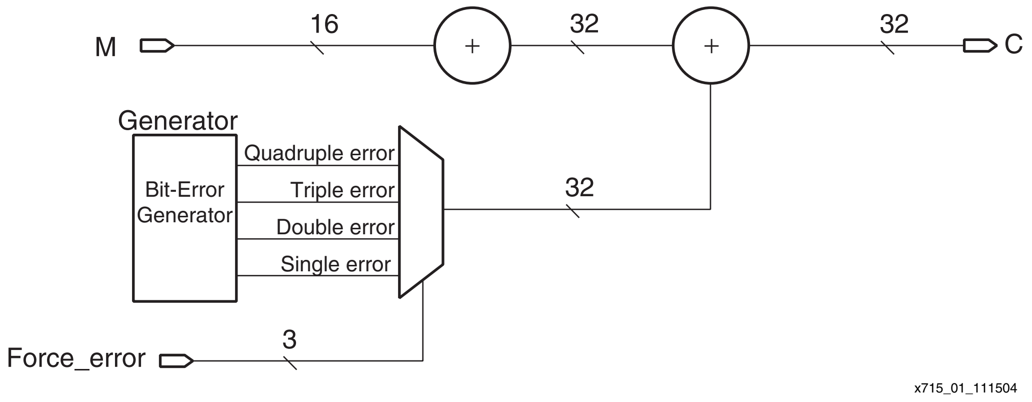

Figure 1: **Encoder Block Diagram**

Error Diagnostics

To test the system, forced-error functions are part of the encoder. Deliberate bit errors can be injected in the code word at the output of the encoder. The FORCE_ERROR pins provide two error diagnostics modes.

- *Normal Operation Mode*
  No bit error imposed on the output of the encoder.
- *Bit Error Mode*
  Depending on the mode-type set by the FORCE_ERROR pins (see Pin Descriptions). Single, double, triple, and quadruple-bit error injection is supported. In bit error mode, one or more consecutive bit(s) is reversed (0 becomes 1 or 0 becomes 1) in the code word on the rising edge of the clock. The sequence moves from low order bits to high order bits. The sequence is repeated as long as the error mode is active.

Decoder

Figure 2 shows the block diagram of the decoder. The decoder has three decoding stages. Each stage is pipelined to maximize performance. It is possible to reduce latency all the way to zero by removing pipelines at the expense of performance. The major components are the Orthogonal Checksum Generator (OCG) and Majority Logic Decoder (MLD). First stage decodes message bit 9 to 0. The second state decodes message bit 14 to 10. The third stage decodes message bit 15. With this method, each subsequent stage operates on the decoded message bits from the previous immediate stage.

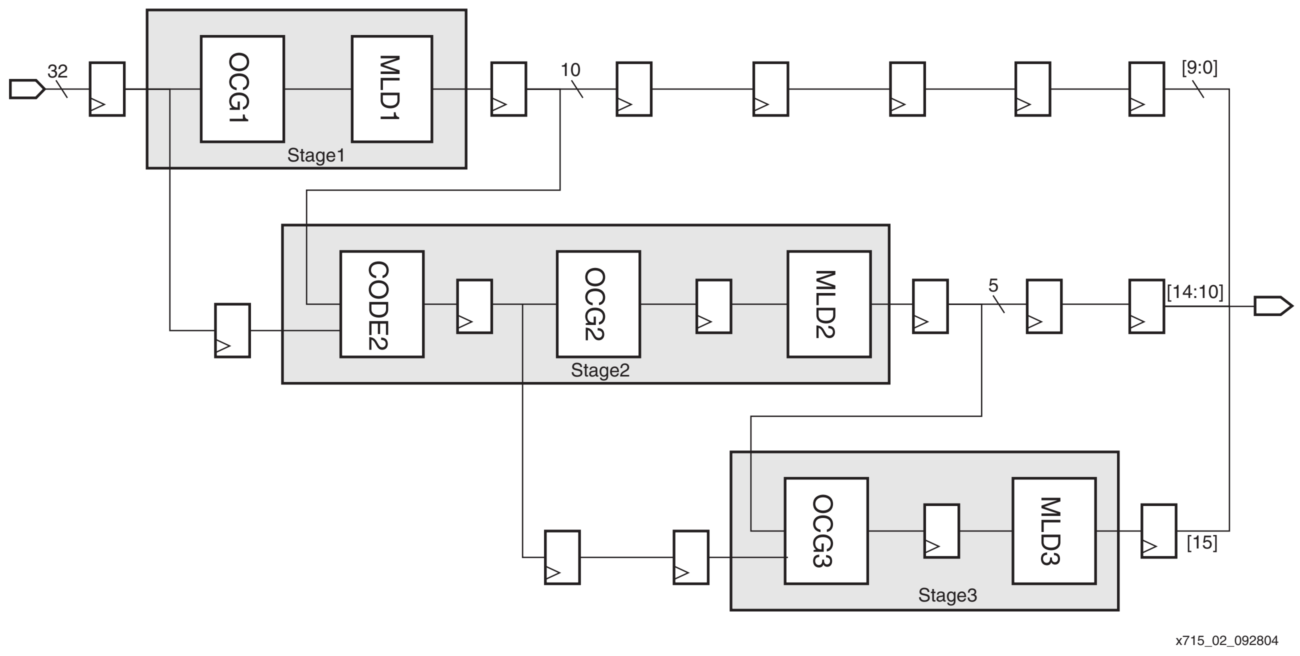

Figure 2: **Decoder Block Diagram**

Design Considerations

Concatenation

Two similar codes can be cascaded to expand the message width. For a 32-bit wide message, two RM(2,5) codes are concatenated making C = {X, Y}, where X and Y are independent RM(2,5) code. Instead of combining two code words side by side {X0,...,X31,Y0,...,Y31}, it is recommended to interleave the code word such that the combined code word is {X0, Y0, X1, Y1,...X31,Y31}. This organization can enhance the correcting capability of certain consecutive bit errors. A concatenated 32-bit reference design is also available.

Use Models

For single-data rate (SDR) memory, the external memory interface width should be the same as the code word width. For double-data rate (DDR) memory applications, the external memory interface width can either be the same as the code word width or the message width (half the code word width). In the later case, half the code word can be accessed with both rising and falling edges at the memory. In both cases, the entire code word is accessed in one cycle on the user side.

Pin Descriptions

Table 3 lists all the encoder and decoder module user interface pins.

Table 3: **Pin Descriptions**

Pin Name	In/Out	Width	Description
CLKIN	In		Clock input
RESET	In		Active High reset
FORCE_ERROR	In	[2:0]	Introduces bit error in the encoded data word for test purposes. `000` – Normal operation `001` – Inject single bit error `010` – Inject double bit error `011` – Inject triple bit error `100` – Inject quadruple bit error
DATA_P	In	[15:0]	Unencoded input data for the encoder
CODE_IN_P	In	[31:0]	Incoming code word for decoder
CODE_OUT_P	Out	[31:0]	Encoded code word generated from the encoder
MESSAGE	Out	[15:0]	Decoded message from the decoder
ERROR	Out	[1:0]	Error status `00` – No error `01` – Error detected and corrected `10` – Quadruple bit error detected. No correction `11` – Invalid bit error detected

Utilization and Performance

Table 4 provides a performance and utilization summary. The design was synthesized using the Xilinx Synthesis Tool (XST). Overall performance varies by design. This summary is of the 16-bit fully-pipelined reference design.

Table 4: **Performance Utilization Summary**

Device	Utilization	Performance
XC2VP7-7	699 slices	184 MHz
XC4VLX15	758 slices	TBD

Conclusion

This application note discusses the basic principle and operation of second order Reed-Muller code. It illustrates the potential use of RM code in correcting multiple errors in high reliability memory system. The reference design is available on the Xilinx web site at:

http://www.xilinx.com/bvdocs/appnotes/xapp715.zip

References

1. Lin, Shu, and Costello, Daniel J. 1983. *Error Control Coding - Fundamentals and Applications*. New Jersey: Prentice Hall.
2. Morelos-Zaragoza, Robert H. 2003. *The Art of Error Correcting Coding*. London: John Wiley & Sons, Ltd.
3. Baylis, John. 1998. *Error Correcting Codes - A Mathematical Introduction*. London: Chapman & Hall/CRC.
4. Cooke, Ben. *Reed-Muller Error Correcting Codes*: MIT Undergraduate Journal of Mathematics Volume 1 1999 http://www-math.mit.edu/phase2/UJM/vol1/COOKE7FF.PDF
5. Gill, John. 2003. *EE387 Reed-Muller code.* Class Handout. http://www.stanford.edu/class/ee387/2003/rm.pdf

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/15/04	1.0	Initial Xilinx release.