International Journal of Electronics and Communication Engineering.
ISSN 0974-2166 Volume 9, Number 1 (2016), pp. 55-62
© International Research Publication House
http://www.irphouse.com

Error Correction Using Extended Orthogonal Latin Square Codes

P. Vamsi Krishna
M. TECH, ECE Department, JNTUACE, Ananthapuramu, India.,

Smt D. Lalitha Kumari
Assistant Professor, ECE Department, JNTUACE, Ananthapuramu, India.

Abstract

To protect memories against errors, error correction codes (ECCs) are used. As frequency of occurring multiple errors are common, we need to go for advanced ECCs. Among advanced ECCs, Orthogonal Latin Squares (OLS) codes have gained renewed interest for memory protection due to their modularity and the simplicity of the decoding algorithm that enables low delay implementations. An important issue is that when ECCs is used, the encoder and decoder circuits can also suffer errors. In this brief, multiple errors correction technique using Extended OLS codes is implemented. The use of more complex codes that can correct more errors is limited by their impact on delay and power, which can limit their applicability to memory designs. To overcome those issues, the use of codes that are one-step majority logic decodable (OS-MLD) has been evolved recently. OS-MLD codes can be decoded with low latency. Therefore they are used to protect memories.

Keywords: Error correction codes (ECCs), Orthogonal Latin square (OLS), One step majority logic decodable (OS-MLD), Memory, Parity.

1. INTRODUCTION

Reliability is a major concern in advanced electronic circuits. To ensure errors do not affect the circuit functionality a number of mitigation techniques can be used. Among them, Error Correction Codes (ECC) are used to protect memories and registers in electronic circuits. The general idea for achieving error correction is to add some redundancy which means to add some extra data to a message, which receiver can use to check uniformity of the delivered message, and to pick up data determined to be corrupt. Error correction scheme may be systematic or it may be non-systematic. In the system of the module non-systematic code, an encoded is achieved by transformation of the message which has least possibility of number of bits present in the message which is being converted. Another classification is the type of systematic module unique data is sent by the transmitter which is attached by a fixed number of parity data like check bits that obtained from the data bits. The receiver applies the same algorithm when only detection of the error is required to the received data bits which is then compared with its output with the receive check bits if the values does not match, there we conclude that an error has occurred at some point in the process of transmission. Error correcting codes are regularly used in lower-layer communication, as well as for reliable storage in media such as CDs, DVDs, hard disks and RAM.

Provision against soft errors that apparent they as the bit-flips in memory is the main motto of error detection and correction. Several techniques are used present to mitigate upsets in memories. For example, the Bose – Chaudhuri– Hocquenghem codes, Reed–Solomon codes, punctured difference set codes, and matrix codes has been used to contact with MCUs in memories. But the above codes mentioned requires more area, power, and delay overheads since the encoding and decoding circuits are more complex in these complicated codes. Reed-Muller code is another protection code that is able to detect and correct additional error besides a Hamming code. But the major drawback of this protection code is the more area it requires and the power penalties. Hamming Codes are mostly used to correct Single Error Upsets (SEU's) in memory due to their ability to correct single errors through reduced area and performance overhead. Although it is brilliant for correction of single errors in a data word, but they cannot correct two bit errors caused by single event upset. An extension of the basic SEC-DED Hamming Code has been proposed to form a special class of codes known as Hsiao Codes to increase the speed, cost and reliability of the decoding logic. One more class of SEC-DED codes known as Single-error-correcting, double error-detecting, Single-byte-error-detecting SEC-DED-SBD codes be proposed to detect any number of errors disturbing a single byte. These codes are additional suitable than the conventional SEC-DED codes for protecting the byte-organized memories. Though they operate through lesser overhead and are good for multiple error detection, they cannot correct multiple errors. There are additional codes such as the single-byte-error-correcting, double-byte-error detecting (SBC-DBD) codes, double-error-correcting, triple error-detecting (DEC-TED) codes that can correct multiple errors. The Single-error-correcting, Double-error-detecting and Double-adjacent-error-correcting (SEC-DED-DAEC) code provides a low cost ECC methodology to correct adjacent errors as proposed. The only drawback through this code is the possibility of miss-correction for a small subset of many errors.

2. LITERATURE SURVEY

Most prior work in memory ECC has focused on low failure rates present at normal operating voltages, and has not focused on the problem of persistent failures in cache memory operating at ultralow voltage where defect rates are very high.

Error correction codes (ECCs) have been used to protect memories for many years. There are wide ranges of codes that used or proposed for the applications in the memory. The codes that can correct one bit per word called the Single error correction are commonly used known as SEC. More sophisticated studies are carried on the codes that correct the two adjacent errors or the two errors. Further the use of more complex codes that corrects more errors is limited by their impact on delay and power, which limits their applicability to the design of memory. To surmount the issues, the use of codes that are one step majority logic decodable (OS -MLD) has been proposed recently. OS-MLD codes can be decoded with low latency and so they are used for the protection of memories. Another type of code that is OS-MLD is orthogonal Latin squares called the OLS codes. While OLS codes require more redundancy than conventional ECC, the one-step majority encoding and decoding process is very fast and can be scaled up for handling large numbers of errors as opposed to BCH codes, which while providing the desired level of reliability requires multi-cycles for decoding. The post-manufacturing customization approach proposed in this paper can be used to reduce the number of check bits and hence the amount of redundancy required in the memory while still providing the desired level of reliability. Note that the proposed approach does not reduce the hardware requirements for the OLS ECC as the whole code needs to be implemented on-chip since the location of the defects is not known until post-manufacturing test is performed.

3. ORTHOGONAL LATIN SQUARE CODES

The concept of Latin squares and their applications are well known. A Latin square of size m is an m * m matrix that has permutations of the digits 0, 1,… and m-1 in both its rows and columns . There can be more than one Latin square for each value of m. In that case, two latin squares are said to be orthogonal if every ordered pair of elements appears only once when they are superimposed. Orthogonal Latin Squares (OLS) codes are derived from Orthogonal Latin squares. These codes have $k=m^2$ data bits and 2tm check bits where ‘t’ is the number of errors that the codes can correct.OLS codes can be decoded using OS-MLD. OS-MLD is a simple procedure in which each bit is decoded by simply taking the majority value of the set of the recomputed parity check equations, in which it participates. This is shown in Fig. 1 for a given data bit di . The reasoning behind OS-MLD is that when an error occurs in bit di, the recomputed parity checks in which it participates will take a value of one.

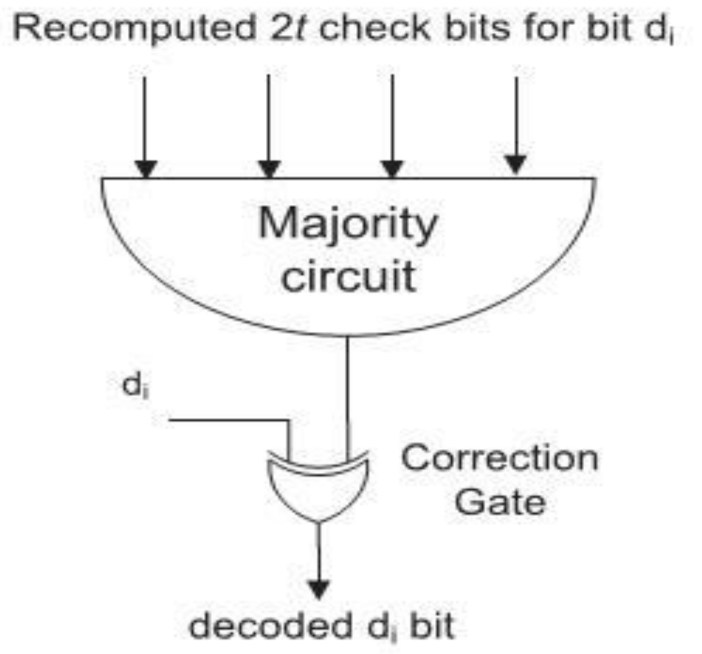


Fig-1: Illustration of OS-MLD decoding for OLS codes

Therefore, a majority of ones in those recomputed checks is an indication that the bit is in error and therefore it needs to be corrected. However, it may also occur that errors in other bits different from di provoke a majority of ones that would cause miss-correction. For a few codes, their properties ensure that this miss-correction cannot occur, and therefore OS-MLD can be used. For a Double Error Correction (DEC) code t=2 and therefore 4m check bits are used. This means that to obtain a code that can correct t+1 errors, simply 2m check bits are added to the code that can correct t errors. The modular property enables the selection of the error correction capability for a given word size. As mentioned in the introduction, OLS codes can be decoded using One Step Majority Logic Decoding (OS-MLD) as each data bit participates in exactly 2t check bits and each other bit participates in at most one of those check bits. This enables a simple correction when the number of bits in error is 't' or less. The 2t check bits are recomputed and a majority vote is taken, if a value of one is obtained, that bit is error and must be corrected. Otherwise it is correct. As long as number of errors is t or less ensures the error correction as the remaining 't-1' errors can, in the worst case it effect t-1 check bits so that still a majority of t+1 triggers the correction of an erroneous bit. For an OLS code that can correct t errors using OS-MLD, t+1 errors can cause miss-corrections. This occurs for example if the errors affect t+1 parity bits in which bit di participates as this bit will be miss-corrected. The same occurs when the number of errors is larger than t+1. Each of the 2t check bits in which a data bit participates is taken from a group of m parity bits. Those groups are bits 1 to m, m+1 to 2m, 2m+1 to 3m and 3m+1 to 4m.

$$\left[\begin{array}{c|c}
1111000000000000 & 1000000000000000 \\
0000111100000000 & 0100000000000000 \\
0000000011110000 & 0010000000000000 \\
0000000000001111 & 0001000000000000 \\
1000100010001000 & 0000100000000000 \\
0100010001000100 & 0000010000000000 \\
0010001000100010 & 0000001000000000 \\
0001000100010001 & 0000000100000000 \\
1000010000100001 & 0000000010000000 \\
0100100000010010 & 0000000001000000 \\
0010000110000100 & 0000000000100000 \\
0001001001001000 & 0000000000010000 \\
1000001000010100 & 0000000000001000 \\
0100000100101000 & 0000000000000100 \\
0010100001000001 & 0000000000000010 \\
0001010010000010 & 0000000000000001
\end{array}\right]$$

Fig-2: Parity check matrix for OLS code with k = 16 and t = 2.

The parity matrix for OLS codes is built from their properties. The matrix is capable of correcting two errors. By the fact that in direction of the modular structure it might be able to correct many errors. The OLS codes have check bits of number “2tm” in which “t” stands for number of errors that code can corrects.

4. IMPLEMENTING METHOD

The implementing method is based on the observation that by construction, the groups formed by the m parity bits in each M_i matrix have at most a one in every column of parity matrix. For the example in Fig. 2, those groups correspond to bits (or rows) 1–4, 5–8, 9–12 and 13–16. Therefore, any combination of four bits from one of those groups will at most share a one with the existing columns in parity matrix. For example, the combination formed by bits 1, 2, 3, and 4 shares only bit 1 with columns 1, 2, 3, and 4. This is the condition needed to enable OS-MLD. Therefore, combinations of four bits taken all from one of those groups can be used to add data bit columns to the parity matrix. For the code with k=16 and t=2 shown in Fig. 2, we have m=4. Hence, one combination can be formed in each group by setting all the positions in the group to one. This is shown in Fig. 3, where the columns added are highlighted. In this case, the data bit block is extended from k= 16 to k=20 bits.

$$\left[\begin{array}{c|c|c}
1111000000000000 & 1000 & 1000000000000000 \\
0000111100000000 & 1000 & 0100000000000000 \\
0000000011110000 & 1000 & 0010000000000000 \\
0000000000001111 & 1000 & 0001000000000000 \\
1000100010001000 & 0100 & 0000100000000000 \\
0100010001000100 & 0100 & 0000010000000000 \\
0010001000100010 & 0100 & 0000001000000000 \\
0001000100010001 & 0100 & 0000000100000000 \\
1000010000100001 & 0010 & 0000000010000000 \\
0100100000010010 & 0010 & 0000000001000000 \\
0010000110000100 & 0010 & 0000000000100000 \\
0001001001001000 & 0010 & 0000000000010000 \\
1000001000010100 & 0001 & 0000000000001000 \\
0100000100101000 & 0001 & 0000000000000100 \\
0010100001000001 & 0001 & 0000000000000010 \\
0001010010000010 & 0001 & 0000000000000001
\end{array}\right]$$

Fig- 3: Parity check matrix for the extended OLS code with k= 20 and t= 2.

The implementing method first divides the parity check bits in groups of m bits given by the M_i matrices. Then, the second step is for each group to find the combinations of 2t bits such that any pair of them share at most one bit. This second step can be seen as that of constructing an OS-MLD code with m parity check bits. Obviously, to keep the OS-MLD property for the extended code, the combinations formed for each group have to share at most one bit with the combinations formed in the other $2t - 1$ groups. When m is small, such combinations are found easily. When m is larger, several combinations can be formed in each group. This occurs, for example, when $m = 8$. In this case, the OLS code has a data block size $k = 64$. With eight positions in each group, now two combinations of four of them that share at most one position can be formed. This means that the extended code will have eight (4×2) additional data bits. As the size of the OLS code becomes larger, the number of combinations in a group also grows. For the case $m = 16$ and $k= 256$, each group has 16 elements. Interestingly enough, there are 20 combinations of four elements that share at most one element. In fact, those combinations are obtained using the extended OLS code shown in Fig. 3 in each of the groups. Therefore, in this case, $4 \times 20 = 80$ data bits can be added in the extended code. The parameters of the extended codes are shown in Table I, where $n - k = 2tm$ is the number of parity bits. The data block size for the original OLS codes (k_{OLS}) is also shown for reference.

The method can be applied to the general case of an OLS code with k = m^2 that can correct t errors. Such a code has 2tm parity bits that as before are divided in groups of m bits. The code can be extended by selecting combinations of 2t parity bits taken from each of the groups. These combinations can be added to the code as long as any pair of the new combinations share at most one bit. When m is small, a set of such combinations with maximum size can be easily found.
However, as m grows, finding such a set is far from trivial (as mentioned before, solving that problem is equivalent to designing an OS-MLD code with m parity bits that can correct t errors). An upper bound on the number of possible combinations can be derived by observing that any pair of bits can appear only in one combination. Because each combination has 2t bits, there are (2t 2) pairs in each combination. The number of possible pairs in each group of m bits is m^2. Therefore, the number of combinations N_G in a group of m bits has to be such that

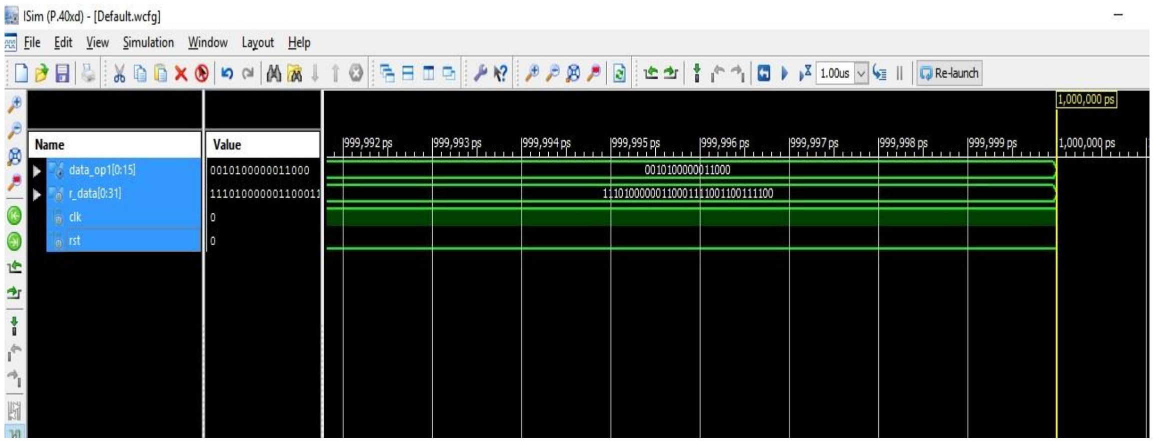
$$\binom{m}{2} \geq \binom{2t}{2} * N_G$$

which can be simplified as$\frac{m^2-m}{4t^2-2t} \geq N_G$

5. Results

The Xilinx ISE software used to perform compilation and synthesis. A test bench is created to execute the simulation. The decoder is stimulated using the Xilinx ISE software. The simulation results for the decoder using OLS code with k=16, t=2, r_data=32 bits (16 data bits and 16 parity bits) and the decoder using extended OLS code with k=20, t=2, r_data=36 (20 data bits and 16 parity bits) is shown as follows, where k- data bits, t- number of errors that extended OLS code can correct, r_data - input of decoder and data_op1 - output.

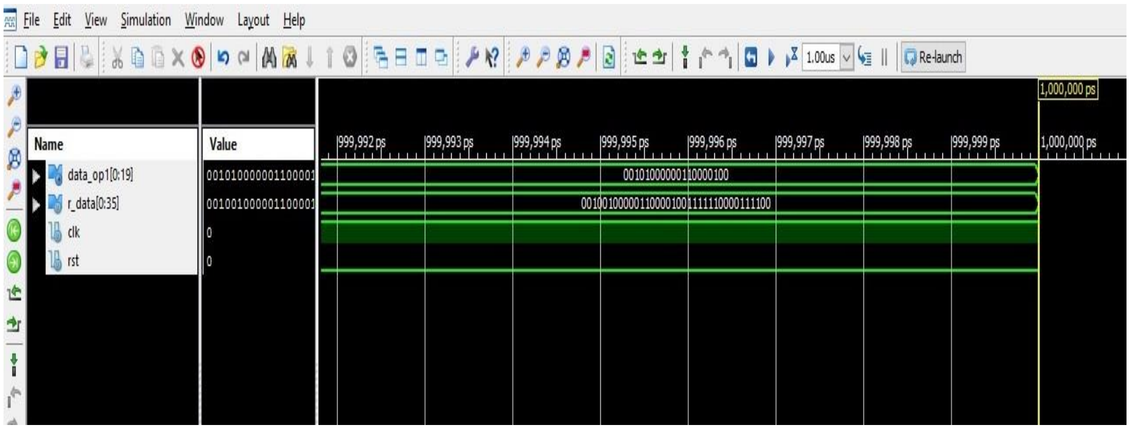
Simulation data: d= 0010_1000_0001_1000. The error is introduced in d_0, d_1 bits. The corrected simulation output seen in fig 4 with k = 16 bits and t = 2 errors.


ISim (P.40xd) - [Default.wcfg]

Name	Value
data_op1[0:15]	0010100000011000
r_data[0:31]	11101000000110001
clk	0
rst	0

Fig- 4: For k = 16 bits OLS codes.

For d = 0010_1000_0001_1000_0100. The error is introduced in d_4 ,d_5 bits. The simulation output with corrected output seen in fig 5 with k= 20 bits and t = 2 errors.



```
Name              Value
data_op1[0:19]    001010000001100001     00101000000110000100
r_data[0:35]      001001000001100001     001001000001100001001111110000111100
clk               0
rst               0
```

Fig- 5: For k= 20 bits extended OLS codes.

CONCLUSION

In this brief, error correction technique for extended OLS codes is implemented. The extended codes have the same number of parity bits as the original OLS codes but a larger number of data bits. The data bits in extended OLS codes are 20 bits whereas parity bits are 16 bits. The number of errors extended OLS codes can correct is 2 adjacent errors. Therefore, the relative overhead is smaller. The derived codes can be decoded using OS-MLD as the original OLS codes. The decoding area and delay are also similar. Therefore, the new codes can be an interesting option to reduce the number of parity bits required to implement multiple bit error correction in memories or caches.

REFERENCES

[1] Pedro Reviriego, Salvatore Pontarelli, Alfonso Sanchez-Macian and Jaun Antonio Maestro"A Method to Extend Orthogonal Latin square Codes" Vol. 22,No.7,July 2014.

[2] R. Datta and N.A. Touba, "Generating burst-error correcting codes from orthogonal Latin square Codes" IEEE int. symp. DFT, 2011.

[3] A. Dutta and N. A. Touba, "Multiple bit upset tolerant memory using a selective cycle avoidance based SEC-DED-DAEC code," in Proc. 25th IEEE VLSI Test Symp. May 2007.

[4] R. Naseer and J. Draper, "DEC ECC design to improve Memory reliability in sub-100 nm Technologies," in Proc. IEEE ICECS, Sep. 2008.