



RELIABLE AND LOW DENSITY BASED ERROR CORRECTION IN SRAM CELLS

¹BONTHU UMA DEVI, ²KORIMILLI SIRISHA, ³V.PRASANNA LAKSHMI

¹PG Student, Dept. of ECE, BVC Institute of Technology & Sciences, Bhatlapalem, A.P

²Professor, Dept. of ECE, BVC Institute of Technology & Sciences, Bhatlapalem, A.P

³Associate Professor, Dept. of ECE, BVC Institute of Technology & Sciences, Bhatlapalem, A.P

ABSTRACT: TCAMs are commonly implemented as standalone devices or as an intellectual property block that is integrated on networking application-specific integrated circuits. However, the flexibility of FPGAs makes them attractive for SDN implementations, and most vendors provide development kits for SDN. Those need to support TCAM functionality and, therefore, there is a need to emulate TCAMs using the logic blocks available in the FPGA. In recent years, a number of schemes to emulate TCAMs on processors have been proposed. Some of them take advantage of the large number of memory blocks available inside modern processors to use them to implement TCAMs. A problem when using memories is that they can be affected by soft errors that corrupt the stored bits. The memories can be protected with a parity check to detect errors or with an error correction code to correct them, but this requires additional memory bits per word. In this brief, the protection of the memories used to emulate TCAMs is considered. Further, this project is enhanced using Decimal Matrix codes to reduce complexity and increase efficiency.

KEYWORDS: Multiple Cell Upsets, Content addressable Memory, Ternary, Soft errors, Error checking, Parity, Decimal Matrix Code.

INTRODUCTION: Multiple Cell Upsets (MCUs) are like a single event that induces several bits in an integrated circuit to fail at the same time. It affects mostly Static Random Access Memory (SRAM). The MCUs occur due to radiation particle striking the memory and the neutrons penetrate into the SRAM memory. Due to this, electron hole pair generation will take place resulting in an accumulation of the charges in the memory. When the charges exceed the critical charge limit, then it can flip the logical state in the memory [1]. It is stated that neutron irradiation reduced the single event latch-up and the sensitivity of CMOS SRAM [2]. Soft errors are a major concern for modern electronic circuits and, in particular, for memories [1]. A soft error can change the contents of the bits stored in a memory and cause a system failure. The soft error rate in terrestrial applications is low. For example, in [2], it was estimated that for a

65-nm static random access memory (SRAM) memory, the bit error rate was on the order of 10^{-9} errors per year. That would translate to only one error per year for a system that uses 1 Gbit of memory. However, even such a low error rate is a big concern for critical applications such as communication networks on which the network elements such as routers have to provide a high level of reliability and availability. Therefore, soft errors are an important issue when designing routers or other network elements, and manufacturers take them into account and incorporate error mitigation techniques [3], [4]. For example, error detection and correction codes are commonly used to protect memories [5]. A parity bit can be added to each memory word to detect single-bit errors, or a single-error correction (SEC) code can be used to correct them. These codes require additional bits per word thus, increasing the memory size and also some logic to write and read from the memory. For example, for a 16-bit word, an SEC code requires 5 bits while a parity check requires only one. Ternary content addressable memories (TCAMs) are a special kind of content addressable memories [6] that support do not care bits (commonly denoted as “x”) that match both a zero and a one. TCAMs are widely used in networking applications to perform packet classification [7]. They can be implemented as standalone devices or integrated as part of networking application specific integrated circuits (ASICs) [8]. The TCAM memory cells differ from normal SRAM cells, in which they check the incoming value for a match to the stored value that can be for each bit 0, 1, or x. The results from all the words are then sent to a priority encoder that returns the match with the highest priority. This comparison and selection logic introduces a significant overhead in terms of area and power consumption relative to that of an SRAM memory.

LITERATURE SURVEY: In [9], a technique was proposed to reduce power consumption of matchlines in content addressable memories (CAMs) called selective precharge technique. In selective precharge technique, the matchline is divided into two segments. Firstly, the searching operation is performed in the first segment in which first few bits of a word i.e. a small subset of CAM cells are searched. If there is a matching of data in the first segment only then searching of remaining bits in the second segment will be activated. In [8], an architecture was proposed having low-power, low-cost, and high-reliability features called as fully parallel precomputation-based content addressable memory (PBCAM). This design is based on a precomputation skill that saves power consumption of the CAM by reducing number of comparisons in the second part of the comparison process. In this design, one’s count approach is used for precomputation. Hence, a one’s count parameter extractor was designed using a chain of full adders but it increases delay as data bit length increases. In [7], a technique was proposed to reduce power consumption of matchlines in content addressable memories (CAMs) called pipelining technique. In this technique, the search operation is pipelined by breaking the match-lines into many segments. Since most stored words do not match in their first segments, the search operation is aborted for subsequent segments. Hence, power gets reduced. The power savings of the pipelined MLs is a result of activating only a small portion of the

matchline segments. In [5], a new approach for PBCAM known as a Block-XOR approach was proposed to improve the efficiency of low power precomputation-based CAM (PBCAM) proposed in [8]. In this paper, a Block-XOR parameter extractor for low power PB-CAM was proposed. This paper presented theoretical and practical proofs to verify that this proposed Block-XOR PB-CAM can effectively achieve greater power reduction by reducing the number of comparison operations in the second part of the comparison process. This implies that this approach is more flexible and adaptive for general designs. In addition, the proposed Block-XOR PB-CAM can compute parameter bits in parallel with only three XOR gate delays for any input bit length (constant delay of search operation). Matrix Code (MC) [15] combined hamming and parity code to protect SRAM memory. It performed better than Hamming. MC corrects MCUs per word with lower decoding delay. In MC two bit errors can be detected by hamming, but these errors can be corrected only when two vertical syndrome bits were activated. Hamming code combined with decimal algorithm [16] to detect and correct soft errors provided low delay overhead by the introduction of integer values. Hamming code did error detection and correction by generating parity codes [17].

CONTENT-ADDRESSABLE MEMORY: Content-addressable memory (CAM) is silicon chip architecture that is purpose-built for extremely fast but very specific type of memory lookups. Lookups using a CAM is conceptually similar to associative array logic in data structures but the output are highly simplified. When key is passed to a CAM sub-system it returns the associated value to that key. As a result a “key -> value” pair is created that can be referenced further as an object. The most important feature is that a lookup of an entry in a CAM can be performed in a single clock cycle in the silicon. Compare this with a RAM module that requires multiple clock cycles to make a single memory fetch.

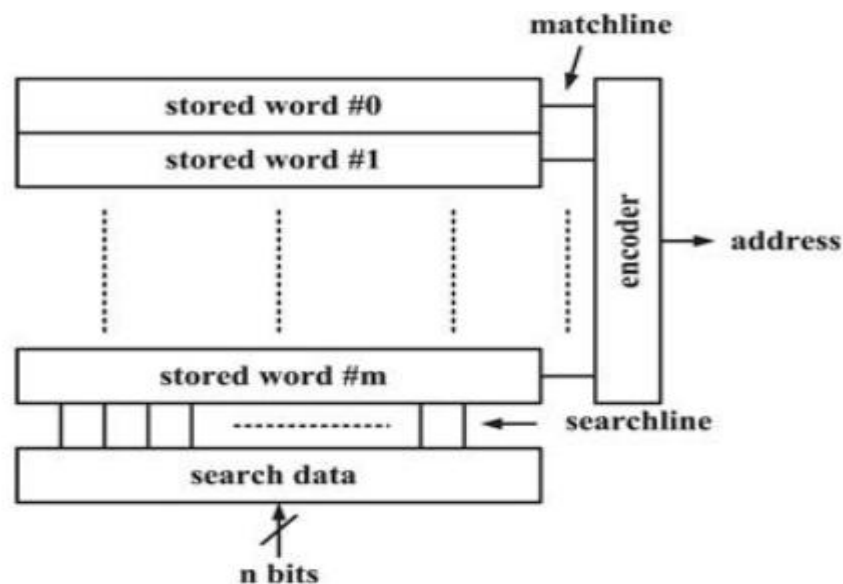


Figure 1: Conceptual View Diagram of CAM

Copyright © 2021ijearst. All rights reserved.

INTERNATIONAL JOURNAL OF ENGINEERING IN ADVANCED RESEARCH
SCIENCE AND TECHNOLOGY

Volume.04, IssueNo.02, November-2021, Pages: 380-391

The conceptual view diagram of CAM is shown in Figure1. It shows that CAM contains m data words in which data is stored. The search word is the n bit input data which is broadcasted onto the search lines to compare it with the table of stored words simultaneously [6]. There is a matchline associated with each stored word which indicates whether the search data is matched with the stored data or not. If the search data is matched with stored data, it is a match case otherwise mismatch case. These matchlines are fed to an encoder. This encoder generates the binary location corresponding to matchline which indicates the match case. If there are more than one matchline that indicates the match case then the priority encoder can be used to generate the matched memory location. The priority encoder gives the matching address location corresponding to highest priority matchline.

ERROR DETECTION AND CORRECTION IN SRAM-BASED TCAMS The scheme proposed to protect the memories used to emulate the TCAM uses a per word parity bit to detect single-bit errors. Then, once an error is detected, the intrinsic redundancy of the memory contents is used to try to correct the error. The implementation of the parity protection is shown in Fig. 2 where p corresponds to the parity bit. It can be seen that in addition to the match signal, an error signal is generated when there is a mismatch between the

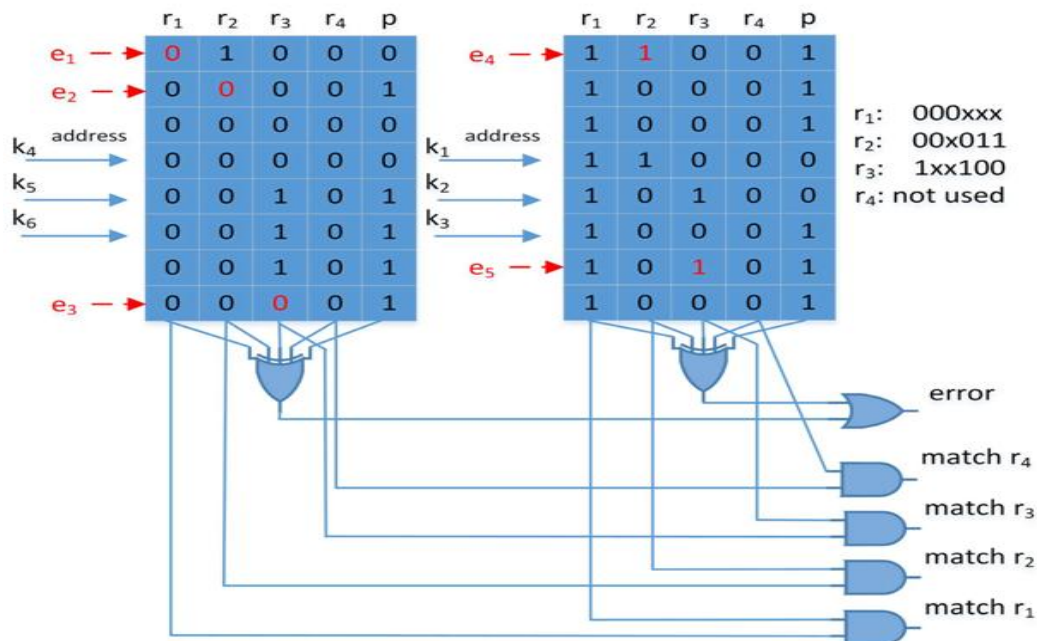


Fig. 2. Examples of single-bit errors on a parity protected

TCAM with 6-bit keys and four rules emulated using two SRAMs, stored parity and the recomputed one. This is a standard parity protection that can detect all single-bit errors [5]. Detecting the error on every access is crucial to avoid incorrect results on search operations. Let us now assume that a single-bit error has occurred on a given word and that it is detected with the parity check. Upon error detection, we can check

the contents of the memory to try to correct the error. A first attempt could be to read all the words in the memory and count the number of positions that have a one for each rule. Let us denote that number as the weight of the rule in that memory. For example, in the leftmost memory of Fig. 2, r_1 would have a weight of 1, r_2 of 2, and r_3 of 4. This can help us identify the erroneous bit as the weight for an error-free rule can only be 0, 1, 2, 4, and 8 for an 8-position memory. To further discuss the error correction process, let us focus on the examples of single-bit errors shown in Fig. 3. For example, e_3 affects r_3 on the leftmost memory by changing its weight from 4 to 3. Since 3 is not a valid value, after detecting the parity error, we would identify that the erroneous bit is that in r_3 and we would correct it. This approach would be effective for rules that have a weight larger than two, i.e., they have two or more “x” bits on the key bits that correspond to that memory. On the other hand, for rules with a lower weight, checking the weight alone may not be enough. Let us now consider a rule with weight two. Then, an error that changes a zero to a one will change the weight to three and the error will be corrected. However, when a one is changed to a zero (as in e_2), then the new weight would be one that is a valid value and the error cannot be corrected. This, however, is less likely to occur as only 2 positions have a one. If we now consider a weight one rule, an error that sets another bit to one would produce a weight of two that is also valid. However, not all weight two combinations are possible. This is clearly seen when looking at e_4 . In that case, the values of r_2 that are one would correspond to key values 000 and 011 and those do not correspond to a valid rule. In general, only positions that correspond to key values that are at distance one from the original value will not be detected. On the other hand, an error that sets to zero the position that was one in a weight one rule can be corrected by checking if the rule has zero weight on the other memories. If that is the case, then the rule is disabled and the bit is not in error. Otherwise, the rule had a weight of one and the error is corrected. Finally, an error in a rule that had a weight of zero can also be corrected by checking the weight of the rule on the other memories. The previous discussion shows that by using the intrinsic redundancy of the memory contents, many single-bit error patterns could be corrected. Let us now quantify the fraction of single-bit error patterns that can be corrected for each weight in a memory of $2b$ positions.

- 1) Weight zero: all patterns can be corrected.
- 2) Weight one: all except those that set a bit to one for a position with an address at distance one, this corresponds to $1 - b/2b$.
- 3) Weight two: all patterns can be corrected except the two that set a position with a one to a zero, this corresponds to $1 - 2/2b$.
- 4) Weight four or larger: all patterns can be corrected.

It can be seen that most of the error patterns are corrected. It can be seen that even for small memories ($b = 5$ corresponds to 32 positions), the error coverage is close to 90% in the worst case. For larger memories, the coverage is over 95% and gets close to 100%. For example, for $b = 9$, the coverage is over 98% in the worst case. This shows the effectiveness of the proposed scheme in correcting single bit errors when the memories are protected with a parity bit. The pseudocode of the proposed correction algorithm. The process starts when a parity error is detected when reading a word from a block memory.

To correct the error, we need to identify the bit (or column) affected by the error. To do so, in the first phase, all the positions in the block are read and the column weights are computed by adding the ones seen in each column. Then, the second phase checks different cases for the column weight to try to identify the erroneous column. If that occurs, the bit of that column in the word that had the parity error is the erroneous bit and it is corrected. In the algorithm, this second phase starts by checking if there is a column that has an illegal weight. As discussed before, the only valid column weights are: 0, 1, $2i$ for $i = 1, 2, \dots, b$. Therefore, if a column has, for example, weight three, then it is the erroneous one. If the erroneous bit is found, it is corrected and the process ends. Otherwise, we proceed to check columns that have zero weight. Those should correspond to TCAM entries that are not used and should have zero weight on all the other memory blocks. Therefore, we check if they have also zero weight on another block. If not, the error has been found and it is corrected. If all the columns with zero weight have also weight zero on another block, we proceed

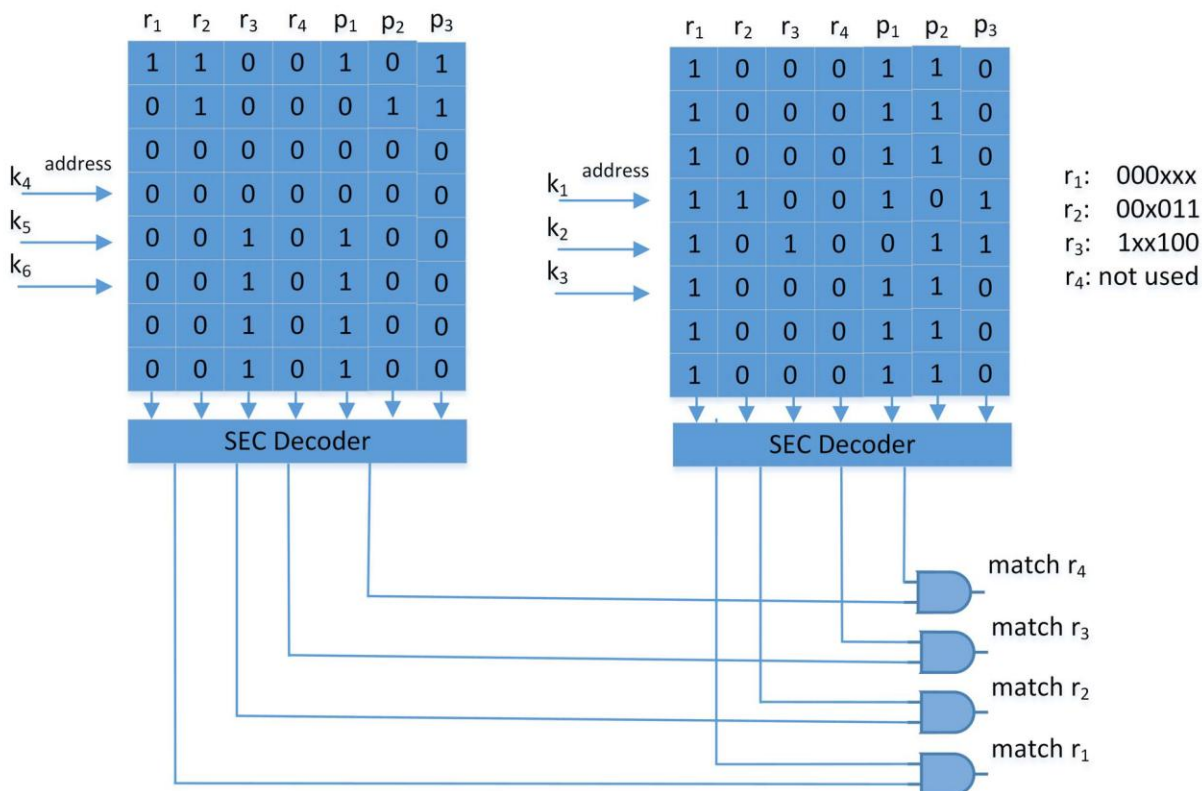


Fig. 3. SEC protected TCAM with 6-bit keys and four rules emulated using two SRAMs.

to check columns of weight one. For that, we check if they have zero weight on another block. If that is the case, that column is the one that suffered the error and we correct it. If not, we proceed to the last step in

which columns of weight two are checked. To do that, the two addresses of the two positions that contain a one are *XORed*. If the result has more than a one, the column has suffered an error and we correct it. If that does not happen, then we have suffered one of the few errors that are not correctable.

PROPOSED TECHNIQUE:

DMC ENCODER: The contribution of this paper is a novel decimal matrix code (DMC) based on divide-symbol is implemented to provide enhanced memory reliability. The implemented DMC utilized decimal algorithm (decimal integer addition and decimal integer subtraction) to identify errors. By using decimal algorithm is that the error detection capability was maximized so that the reliability of memory was enhanced. Besides, the encoder-reuse technique (ERT) was implemented to minimize the area overhead of extra circuits (encoder and decoder) without disturbing the whole encoding and decoding processes, because ERT use DMC encoder itself to be part of the decoder.

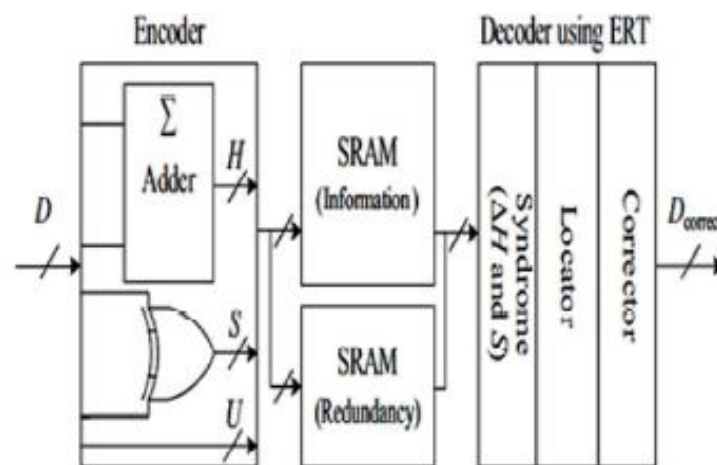


Fig4: DMC Proposed block

In the proposed scheme, the circuit area of DMC is minimized by reusing its encoder. This is called the ERT. The ERT can reduce the area overhead of DMC without disturbing the whole encoding and decoding processes. It can be observed that the DMC encoder is also reused for obtaining the syndrome bits in DMC decoder. Therefore, the whole circuit area of DMC can be minimized as a result of using the existent circuits of encoder. Besides, this figure also shows the proposed decoder with an enable signal En for deciding whether the encoder needs to be a part of the decoder. In other words, the En signal is used for distinguishing the encoder from the decoder, and it is under the control of the write and read signals in memory. Therefore, in the encoding (write) process, the DMC encoder is only an encoder to execute the encoding operations. However, in the decoding (read) process, this encoder is employed for computing the syndrome bits in the decoder. These clearly show how the area overhead of extra circuits can be

substantially reduced. In the proposed DMC, first, the divide-symbol and arrange-matrix ideas are performed, i.e., the N-bit word is divided into k symbols of m bits ($N = k \times m$), and these symbols are arranged in a $k_1 \times k_2$ 2-D matrix ($k = k_1 \times k_2$, where the values of k_1 and k_2 represent the numbers of rows and columns in the logical matrix respectively). Second, the horizontal redundant bits H are produced by performing decimal integer addition of selected symbols per row. Here, each symbol is regarded as a decimal integer. Third, the vertical redundant bits V are obtained by binary operation among the bits per column. It should be noted that both divide-symbol and arrange-matrix are implemented in logical instead of in physical. Therefore, the proposed DMC does not require changing the physical structure of the memory. To explain the proposed DMC scheme, we take a 32-bit word as an example, as shown in Fig.4.2. The cells from D0 to D31 are information bits. This 32-bit word has been divided into eight symbols of 4-bit. $k_1 = 2$ and $k_2 = 4$ have been chosen simultaneously. H0-H19 are horizontal check bits; V0 through V15 are vertical check bits. However, it should be mentioned that the maximum correction capability (i.e., the maximum size of MCUs can be corrected) and the number of redundant bits are different when the different values for k and m are chosen. Therefore, k and m should be carefully adjusted to maximize the correction capability and minimize the number of redundant bits. For example, in this case, when $k = 2 \times 2$ and $m = 8$, only 1-bit error can be corrected and the number of redundant bits is 80. When $k = 4 \times 4$ and $m = 2$, 3-bit errors can be corrected and the number of redundant bits is reduced to 32. However, when $k = 2 \times 4$ and $m = 4$, the maximum correction capability is up to 5 bits and the number of redundant bits is 72. In this paper, in order to enhance the reliability of memory, the error correction capability is first considered, so $k = 2 \times 8$ and $m = 4$ are utilized to construct DMC.

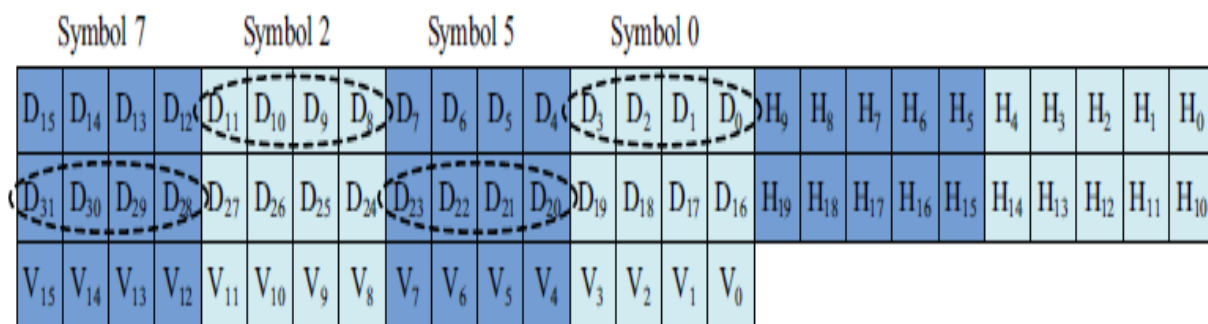


Fig: 5 32 bit word can be divided as 8 symbols with $k=2 \times 4$ and $m=4$.

Horizontal redundant bits are calculated as follows

$$H_9 H_8 H_7 H_6 H_5 = D_7 D_6 D_5 D_4 + D_{15} D_{14} D_{13} D_{12} \quad (1)$$

$$H_4 H_3 H_2 H_1 H_0 = D_3 D_2 D_1 D_0 + D_{11} D_{10} D_9 D_8 \quad (2)$$

Similarly for the horizontal redundant bits H14 H13H12 H11 H10 and H19 H18H17H16 H15, where "+" represents decimal integer addition.

Vertical redundant bits are calculated as follows

$$V0 = D0 \oplus D16 \quad (3)$$

$$V1 = D1 \oplus D17 \quad (4)$$

And similarly for the rest vertical redundant bits. The encoding can be performed by decimal and binary addition operations from (1) to (4). The encoder that computes the redundant bits using multi bit adders and XOR gates is shown in Fig. In this figure, H19 – H0 are horizontal redundant bits, V15 – V0 are vertical redundant bits, and the remaining bits U31 – U0 are the information bits which are directly copied from D31 to D0

RESULT:

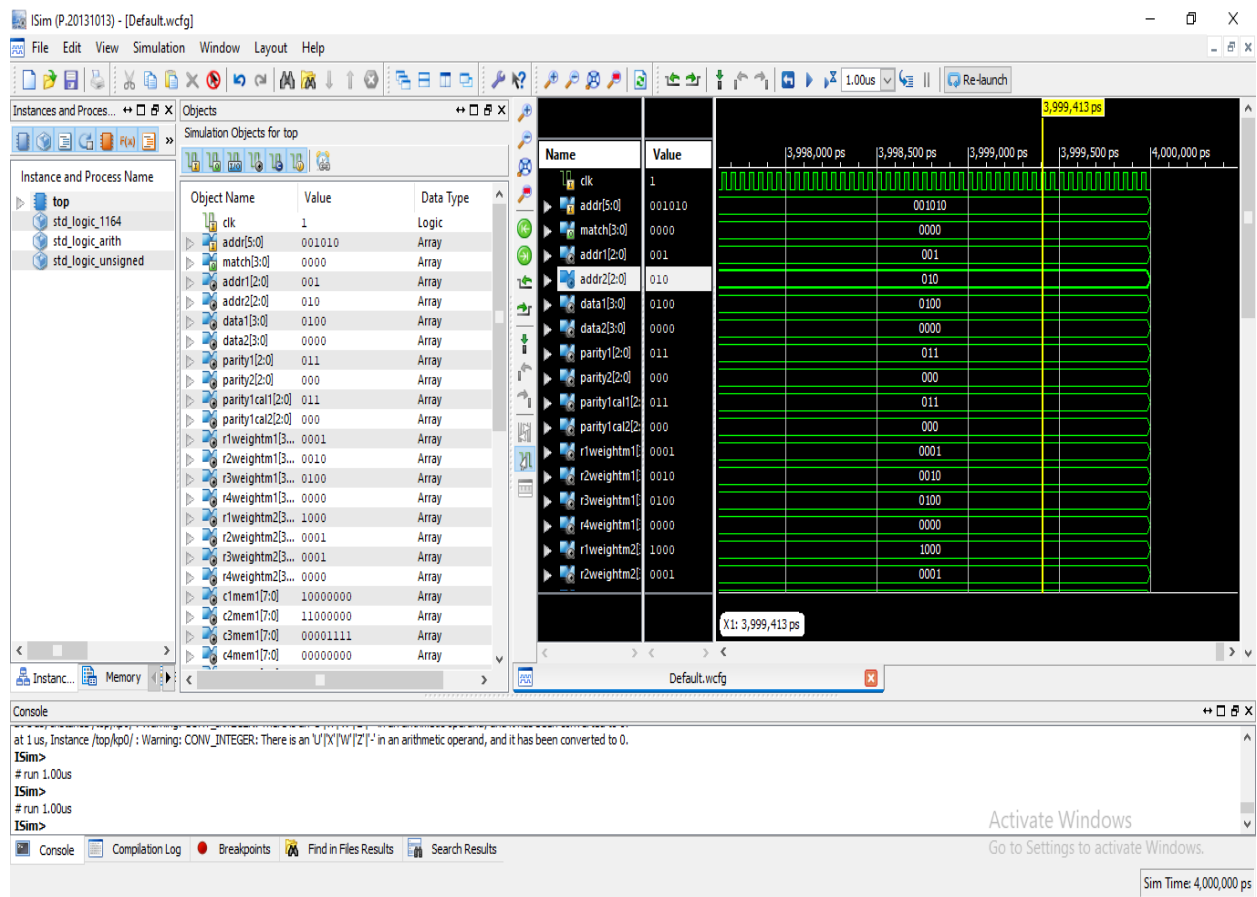


Fig: Error correction using per word parity

Copyright © 2021ijearst. All rights reserved.

INTERNATIONAL JOURNAL OF ENGINEERING IN ADVANCED RESEARCH
SCIENCE AND TECHNOLOGY

Volume.04, IssueNo.02, November-2021, Pages: 380-391

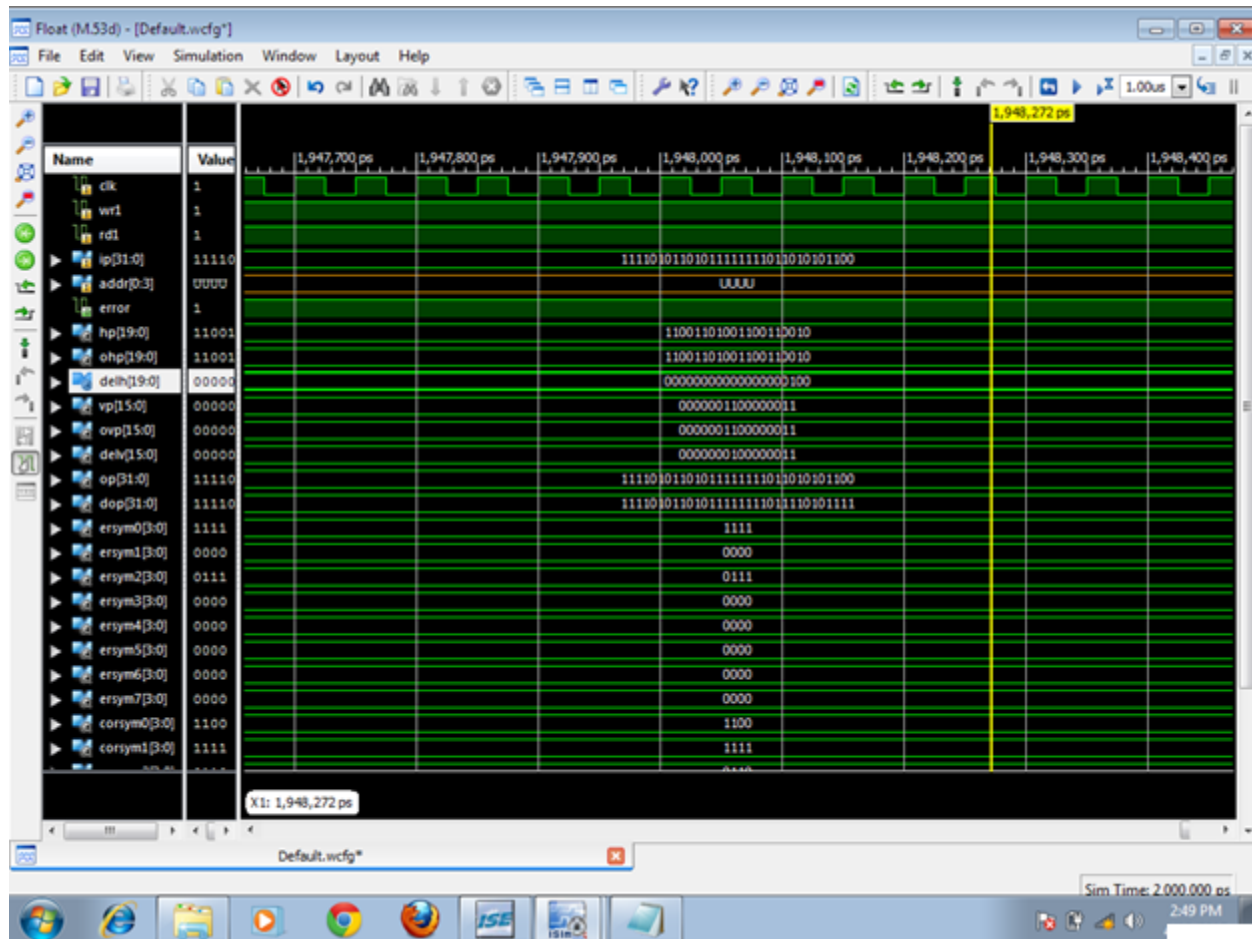


Fig: Error correction using DMC

CONCLUSION: To prevent MCUs from causing data corruption, more complex error correction codes (ECCs) are widely used to protect memory, but the main problem is that they would require higher delay overhead. Recently, matrix codes (MCs) based on Hamming codes have been proposed for memory protection. In this implemented project, novel per-word DMC was proposed to assure the reliability of memory. The protection code utilized decimal algorithm to detect errors, so that more errors were detected and corrected. The obtained results showed that the implemented scheme has a superior protection level against large MCUs in memory. Besides, the implemented decimal error detection technique is an attractive opinion to detect MCUs in CAM because it can be combined with BICS to provide an adequate level of immunity.

REFERENCES:

- [1] DejanGeorgiev, "Low Power Concept for Content Addressable Memory (CAM) Chip Design," International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, Vol.2, Issue 7, July 2013.
- [2] S. Jeeva, S. Bharathi and Dr. C. N. Marimuthu, "Low Power Architecture of Banked Pre-Computation Based Content Addressable Memory", International Conference on Computing, Communication and Information Technology (ICCCIT 2012), p.p. 156-160, June 2012.
- [3] Rafeekha M. J, V. Lakshmi Narasimhan, "Banked Approach of Low Power Design of Pre-Computation Based Content Addressable Memory", International Journal of Modern Engineering Research (IJMER), Vol. 2, Issue 3, p.p. 1424-1429, May-June 2012.
- [4] Subha. M, "The Efficient Architecture Methods for Low Power Content Addressable Memory- Survey", Recent Advances in Networking, VLSI and Signal Processing, p.p. 141-146.
- [5] S. J. Ruan, C.Y. Wu, J. Y. Hsieh, "Low power design of pre-computation based content-addressable memory," IEEE Transactions Very Large Scale Integration (VLSI) Systems, Vol. 16, No. 3, p.p. 331-335, March 2008.
- [6] K. Pagiamtzis and A. Sheikholeslami, "ContentAddressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey," IEEE Journal of Solid-State Circuits, Vol. 41, p.p. 712-727, March 2006.
- [7] K. Pagiamtzis and A. Sheikholeslami, "A low-power content-addressable memory (CAM) using pipelined hierarchical search scheme," IEEE J. Solid-State Circuits, Vol. 39, No. 9, p.p. 1512-1519, Sep. 2004.
- [8] C.-S. Lin, J.-C. Chang, and B.-D. Liu, "A low-power precomputation-based fully parallel content-addressable memory," IEEE J. Solid-State Circuits, Vol. 38, No. 4, p.p. 654-662, Apr. 2003.
- [9] C. A. Zukowski and S.-Y. Wang, "Use of selective precharge for low power content-addressable memories," in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), Vol. 3, p.p. 1788-1791, 1997.