An Integrated ECC and BISR Scheme for Error Correction in Memory

Shabana P B[1], Anu C Kunjachan[2] , Swetha Krishnan[3]

[1] PG Student [VLSI], Dept. of ECE, Viswajyothy College Of Engineering & Technology, Vazhakulam ,Kerala, India
[2] Assistant Professor, Dept. of ECE, Viswajyothy College Of Engineering & Technology, Vazhakulam ,Kerala, India
[3] PG Student [VLSI], Dept. of ECE, Viswajyothy College Of Engineering & Technology, Vazhakulam ,Kerala, India

---***---

Abstract - ***With the fast development of submicron technology, the size and density of semiconductor memory grows rapidly. However, keeping a high level of yield and reliability for memory products is more and more difficult. Both the Built in self-repair (BISR) and ECC techniques have been widely used for enhancing the yield and reliability of memory chips. Specifically, the BISR and ECC techniques are conventionally used to repair or correct the hard faults and soft errors, respectively. Error Correction Code used in this paper is a novel decimal matrix code (DMC) based on divide-symbol is proposed to enhance memory reliability. The proposed DMC utilizes decimal algorithm to obtain the maximum error detection capability. Moreover, the encoder-reuse technique (ERT) is proposed to minimize the area overhead of extra circuits without disturbing the whole encoding and decoding processes. DMC Code is a very powerful technique to correct MCUs in memory. Built in self-repair circuits are used to detect and replace faulty elements by spare elements. In this paper, an integrated ECC and BISR scheme for memory reliability enhancement is proposed.***

Key Words: *ECC, BISR, DMC, ERT etc…*

1. INTRODUCTION

Embedded memories play an important role in the semi-conductor market because the system-on chip market is booming and almost every system chip contains some type of embedded memory. High-density, low-voltage levels, small feature size and small noise margins make the memory chips increasingly susceptible to faults. The failure modes of memory chips are generally classified into two main classes, i.e., those lead to hard faults and those to soft errors. Both are assumed to cause the memory chip to fail, so they affect the memory chip's reliability.

Soft errors are caused by high energy neutrons and alpha particles hitting the silicon bulk resulting in the production of large number of electron-hole pairs. The accumulated charge may be sufficient to flip the value stored in a cell thus causing bit inversion, resulting in soft error. Hence the effects of radiation are bit-flips occurring in the information stored in memory elements. Due to the relentless shrinkage in the device dimensions, the particles that were once considered negligible are now proving to be significant enough to cause upsets. Such errors are identified as soft errors since, although they corrupt the value stored in the cell, they do not permanently damage the hardware. The sources of hard fault are random defects, systematic defects, and parametric defects. For hard errors, like manufacturing defects, off-line repair schemes use built-in self-test (BIST) and built-in self-repair (BISR) circuits to detect and replace faulty elements by spare elements. Redundancy types such as spare words, spare rows, or spare columns are commonly used. For soft errors, on-line concurrent repair schemes, such as error correction codes (ECC), correct the soft errors, which are caused by the variation of operating environment or radioactive alpha particles.

1.1 Block diagram of the proposed Integrated ECC and BISR Scheme

Block diagram of Integrated BISR and ECC Scheme is shown in figure 1. It consists of memory, ECC, BIST, Redundancy analyzer, Reconfiguration mechanism and Spare elements. Data can be read and written into the memory. While reading data from memory, error signal raises if the data is corrupted. Then the Error Correction Code is activated. Errors can be detected and corrected using ECC. Hard errors can be corrected but, can't be eliminated using ECC. If hard fault is detected, BIST module activates. Fault address is detected by BIST. Redundancy analyzer identifies must repair rows and columns and searches for a repair solution. Reconfiguration mechanism replaces the must repair rows or columns with spare elements. Thus, we can eliminate hard errors from the memory. By integrating both ECC and BISR scheme, we can eliminate both soft errors and hard errors from the memory.

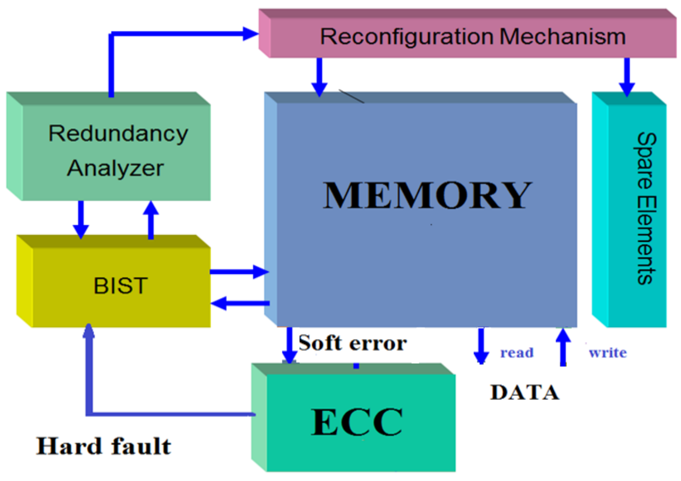


Fig -1: Block diagram of Integrated ECC and BISR

1.2. Error Correction Codes (ECCs)

Soft errors are the major issue in the reliability of memories. Soft error will not damage the hardware; they only damage the data that is being processed. If detected, soft errors are corrected by rewriting corrected data in the place of erroneous data.

To prevent soft errors from causing corruption in the data stored error correction codes are used such as matrix code, hamming etc. When ECC is used, data are encoded when written in the memory and data are decoded when read from the memory. Thus the encoding and decoding process possess a vital impact on the memory access time and complexity. Multiple cell upsets have become the reliability concern in some application apart from single cell upset. The general idea for achieving error detection and correction is to add some redundant bits. Errors in memory can be detected and corrected using these redundant bits.

Soft errors can be corrected using Error Correction Code. DMC (Decimal Matrix Code) [2] is used in this paper to correct soft errors occurring in the memory. Decimal matrix code (DMC) is based on divide-symbol is proposed to provide enhanced memory reliability. The proposed DMC utilizes decimal algorithm (decimal integer addition and decimal integer subtraction) to detect errors.

The advantage of using decimal algorithm is that the error detection capability is maximized so that the reliability of memory is enhanced. Besides, the encoder-reuse technique (ERT) is proposed to minimize the area overhead of extra circuits (encoder and decoder) without disturbing the whole encoding and decoding processes, because ERT uses DMC encoder itself to be part of the decoder.

The proposed schematic of fault-tolerant memory is depicted in Figure 2. The block diagram of DMC consists of encoder, memory and decoder. Input data is given to the input of the encoder. Encoder consists of adders and xor gates. Decoder consists of syndrome generator, error locator and error corrector. Corrected data is obtained from the output of the Decoder.

First, during the encoding (write) process, information bits D are fed to the DMC encoder, and then the horizontal redundant bits H and vertical redundant bits V are obtained from the DMC encoder. When the encoding process is completed, the obtained DMC code word is stored in the memory.

If MCUs occur in the memory, these errors can be corrected in the decoding (read) process. Decoder consists of syndrome generator, error locator and error corrector. Due to the advantage of decimal algorithm, the proposed DMC has higher fault-tolerant capability with lower performance overheads.

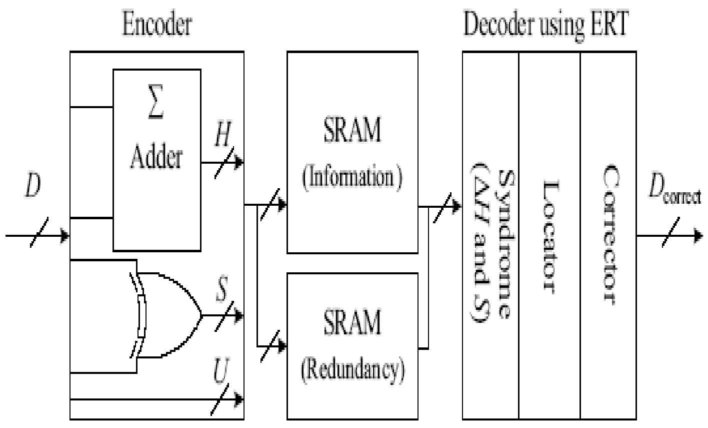


Figure-2: Proposed schematic of fault-tolerant memory protected with DMC.

In the fault-tolerant memory, the ERT technique is proposed to reduce the area overhead of extra circuits and will be introduced in the following sections. In proposed method using Decimal matrix code, information bits are fed to the DMC Encoder and corrected data is obtained from DMC Decoder.

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 ×m), and these symbols are arranged in a k1 ×k2 2-D matrix (k = k1 ×k2, where the values of k1 and k2 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. Proposed DMC Encoder [2] is shown in the fig. 3.

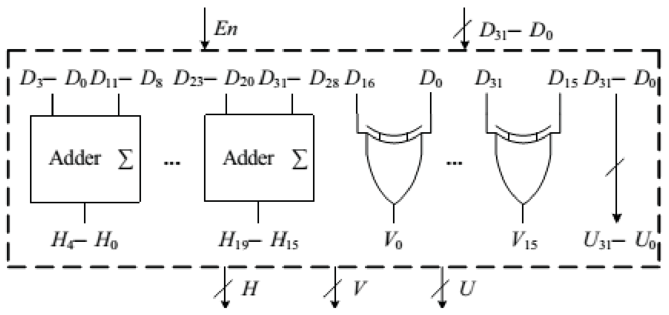

En
$D_{31}-D_0$
D_3-D_0 $D_{11}-D_8$ $D_{23}-D_{20}$ $D_{31}-D_{28}$ D_{16} D_0 D_{31} D_{15} $D_{31}-D_0$
Adder Σ
Adder Σ
H_4-H_0 $H_{19}-H_{15}$ V_0 V_{15} $U_{31}-U_0$
H
V
U

Fig -3: 32-bit DMC encoder structure using multi bit adders and XOR gates.

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-bitword as an example, as shown in Fig.4. The cells from D0 to D31 are information bits. This 32-bit word has been divided into eight symbols of 4-bit. k1 = 2 and k2 = 4 have been chosen simultaneously. H0–H19 are horizontal check bits; V0 through V15 are vertical check bits.

For the vertical redundant bits V, we have

V0 = D0 ⊕D16 (3)
V1 = D1 ⊕D17 (4)

Similarly, for the rest vertical redundant bits also. The remaining bits U31 –U0 are the information bits which are directly copied from D31 to D0. The proposed DMC decoder [2] is depicted in Fig. 5.

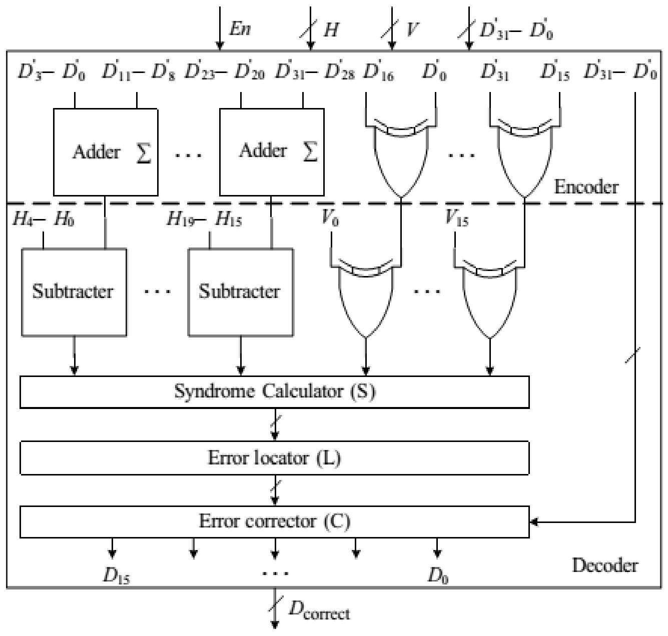
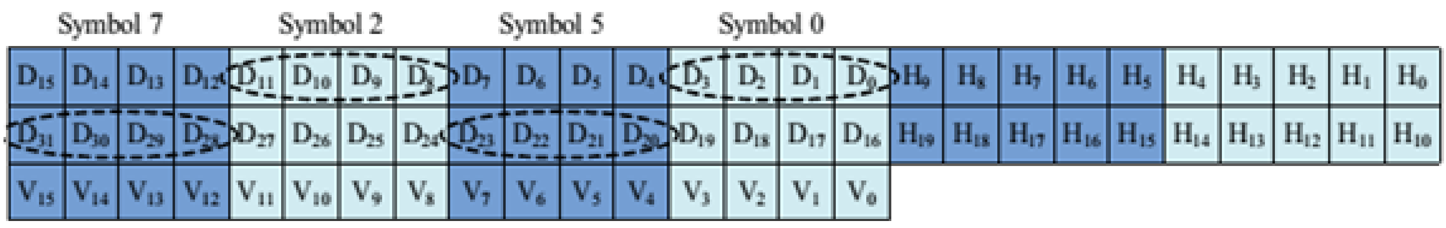

En
H
V
$D'_{31}-D'_0$
$D'_3-D'_0$ $D'_{11}-D'_8$ $D'_{23}-D'_{20}$ $D'_{31}-D'_{28}$ D'_{16} D'_0 D'_{31} D'_{15} $D'_{31}-D'_0$
Adder Σ
Adder Σ
Encoder
H_4-H_0 $H_{19}-H_{15}$ V_0 V_{15}
Subtracter
Subtracter
Syndrome Calculator (S)
Error locator (L)
Error corrector (C)
D_{15} D_0
Decoder
$D_{correct}$

Fig -5: 32-bit DMC decoder structure using ERT.



D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	H_9	H_8	H_7	H_6	H_5	H_4	H_3	H_2	H_1	H_0
D_{31}	D_{30}	D_{29}	D_{28}	D_{27}	D_{26}	D_{25}	D_{24}	D_{23}	D_{22}	D_{21}	D_{20}	D_{19}	D_{18}	D_{17}	D_{16}	H_{19}	H_{18}	H_{17}	H_{16}	H_{15}	H_{14}	H_{13}	H_{12}	H_{11}	H_{10}
V_{15}	V_{14}	V_{13}	V_{12}	V_{11}	V_{10}	V_9	V_8	V_7	V_6	V_5	V_4	V_3	V_2	V_1	V_0										

Fig -4: 32-bits DMC logical organization (k = 2×4&m = 4).

In order to enhance the reliability of memory, the error correction capability is first considered, so k = 2 × 4 and m = 4 are utilized to construct DMC. The horizontal redundant bits H can be obtained by decimal integer addition as follows:

H4H3H2H1H0 = D3D2D1D0 + D11D10D9D8 (1)
H9H8H7H6H5 = D7D6D5D4 + D15D14D13D12 (2)

Similarly for the horizontal redundant bits H14H13H12H11H10 and H19H18H17H16H15, where "+"represents decimal integer addition.

To obtain a word being corrected, the decoding process is required. For example, first, the received redundant bits H4H3H2H1H0' and V0' –V3' are generated by the received information bits D'. Second, the horizontal syndrome bits Del(H4H3H2H1H0) and the vertical syndrome bits S3 –S0 can be calculated as follows:

Del(H4H3H2H1H0)=H4H3H2H1H0'−H4H3H2H1H0 (5)
S0 = V0'⊕V0 (6)

The sign "−"represents decimal integer subtraction.

When Del(H4H3H2H1H0 and S3 −S0 are equal to zero, the stored code word has original information bits in symbol 0 where no errors occur. When Del(H4H3H2H1H0) and S3 −S0 are nonzero, the induced errors (the number of errors is 4 in this case) are detected and located in symbol 0, and then these errors can be corrected by

$$D0correct= D0 \oplus S0 \qquad (7)$$

The proposed DMC decoder is depicted in Fig. 5, which is made up of the following sub modules, and each executes a specific task in the decoding process: syndrome calculator, error locator, and error corrector. It can be observed from this figure that the redundant bits must be recomputed from the received information bits D' and compared to the original set of redundant bits in order to obtain the syndrome bits Del(H) and S. Then error locator uses Del(H) and S to detect and locate which bits some errors occur in. Finally, in the error corrector, these errors can be corrected by inverting the values of error bits.

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. From fig. 5, 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.

1.3. Built in Self Repair (BISR)

On-chip infrastructure for Built in Self Repair (BISR) consist of Built in Self-Test (BIST) module, Built in Redundancy Analyzer (BIRA) module and Reconfiguration mechanism. Fig 6 shows an on-chip infrastructure for Built in Self Repair (BISR).

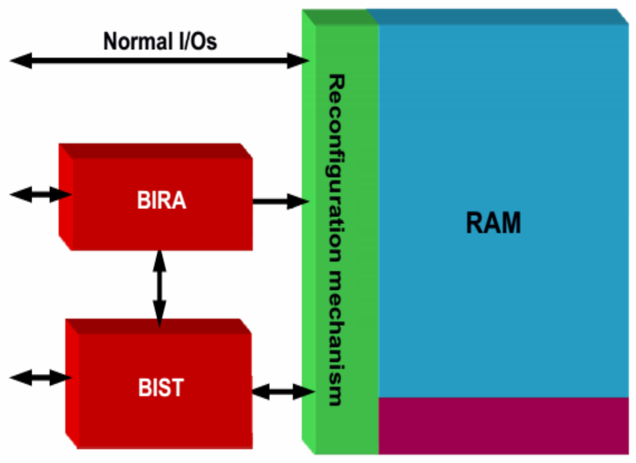


Fig -6: Block diagram of BISR

There are 2 phases of operation for a BISR Scheme. They are Must repair analysis phase and Final repair phase. Must repair analysis phase identifies the must repair rows and columns. It is performed concurrently with the test. Final analysis phase searches for a repair solution. It is done after the test is completed.

Fig 7 shows an on-chip infrastructure of BISR [1] for bit-oriented memories. Repair analyzer requires only a single test and provides the optimal repair rate. This infrastructure does not depend on BIST engines, and assume that an arbitrary BIST engine tests a memory array and provides fault addresses whenever detected.

This infrastructure consists of must-repair analysis and final analysis. The must-repair analysis identifies must-repair rows and columns, and the final analysis searches a repair solution. The must-repair analysis is performed concurrently with the test, while the final analysis is done after the test is completed.

The must repair analyzer (MRA) [1] consists of a pair of CAMs for fault addresses, called the fault-list, and a pair of CAMs for a repair solution, called the solution record. The must repair analyzer (MRA) is shown in Fig. 7.

In the fault-list, each CAM has one extra valid bit for each word, and the valid bits are initialized to “0” in the beginning. Since the CAMs assert “1” at the valid bit position for write and match operation, only written entries can be matched.

During the test, if the BIST engine detects a fault, it sends the fault address to the MRA on the fly through BIST_R_DUTAddr and BIST_C_DUTAddr, and continues the test. The row (column) fault address is compared against row (column) CAM entries, and the number of matched entries is efficiently counted by a parallel counter.

If the number of the matched entries equals in the row (column) CAM, the row (column) indicated by the fault address satisfies the must-repair condition and R_MustRepair (C_MustRepair) signal is asserted. If the fault address triggers neither the row nor column must-repair condition, MRA writes the row and column address in the row and column CAMs, respectively.

If a particular row or column is identified as must-repair, the row or column address must be part of the solution. Thus the MRA writes the row or column address in the solution record. The L registers are used as valid bits for the solution record and also determine the next available CAM entry. Since a must-repair row and a must-repair column can be identified by a fault at the same time, the MRA should be able to write a pair of row and column addresses simultaneously.

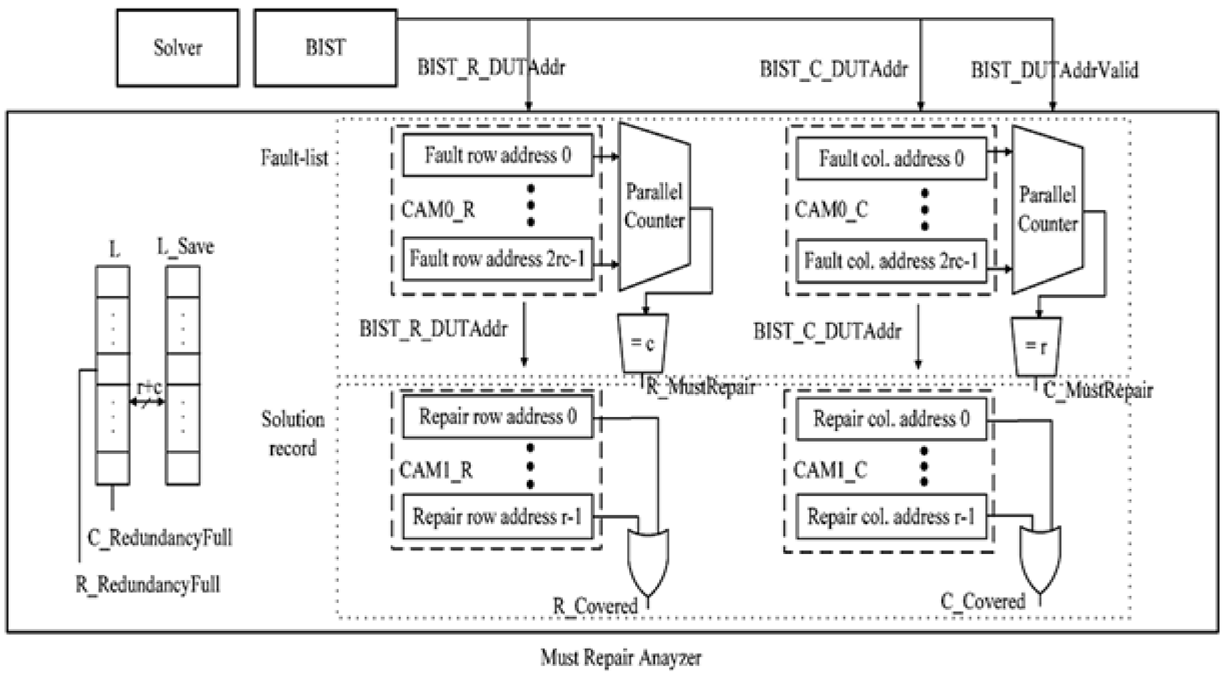


Fig -7: Proposed on-chip must repair analyzer

Once a row or column address is stored as part of solution by the must-repair condition, then all solution candidates considered by the SOLVER include the address, and faults on the address do not affect the final analysis any more. Therefore, such faults do not need to be stored, and we can collect all necessary information for the final analysis during a single test. Once the test is completed (thus the must-repair analysis is done), BIST_Done signal is asserted and the final analysis is started. In the final analysis, the SOLVER module controls the MRA. The operation of the SOLVER and the MRA in the final analysis phase is illustrated in Fig.8. The SOLVER will generate repair strategies one by one and will check whether each repair strategy can fix all the faults captured in the fault-list.

The Repair Strategy module comprises a -bit register and stores the repair strategy being tested currently. The first repair strategy is generated depending on the numbers of must-repair rows and columns, or Used_Must_Repair_Rows and Used_Must_Repair_Cols. The SOLVER [1] generates the first repair strategy and the MRA reads each fault address in the fault-list in order until there exists no more fault address or the RESTART signal is arrived. The MRA check if each fault is covered by the current solution, stored in the solution record, and asserts R_Covered or C_Covered. If both signals are low, the fault should be covered by a new repair row or column.

The SOLVER determines whether a repair row or column is used for the uncovered fault, and asserts R_Insert or C_Insert. If R_Insert (C_Insert) is high, the fault row (column) address is written in the row (column) CAM of the solution record. If the CAM is full, the memory array cannot be repaired by the first repair strategy, and the SOLVER generates the next repair strategy and asserts the RESTART signal. The next repair strategy is generated by the Solver. When the RESTART signal becomes high, the MRA restores the initial state, and the next repair strategy starts being evaluated. In this way, the SOLVER explores the solution space and can find a solution if one exists.

In this implementation, the cost is defined as the number of used spare elements. The SOLVER has a register to store the cost of the current repair strategy, or Used_Repair_Elements. The SOLVER also has registers to store the repair strategy with minimum cost so far and the minimum cost, or Repair_Strategy_Opt and Used_Repair_Opt. The current cost is compared against the minimum cost so far, which generates the Better signal.

If the Better signal goes down during the evaluation of the current repair strategy, the SOLVER immediately asserts the RESTART signal and moves on to the next repair strategy. If the Better signal stays at “1” until the end of the evaluation, the SOLVER saves the current repair strategy and its cost. Since the SOLVER continues to search for a better solution even after finding a solution, the MRA may not have the optimal solution after the last repair strategy is evaluated.

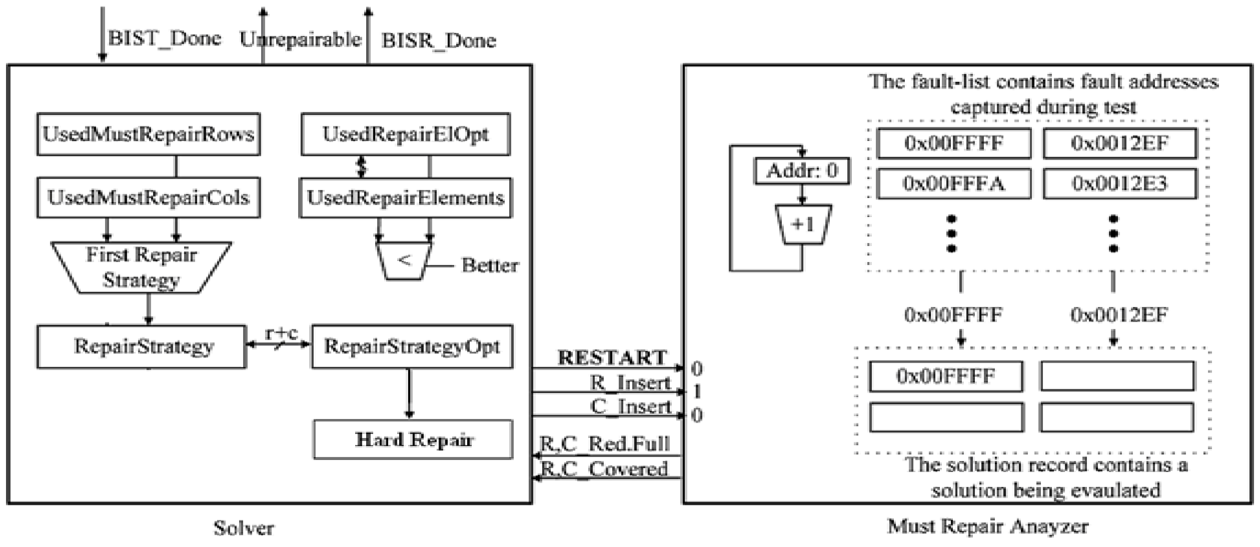


Fig -8: Solver details and MRA operation

In the recovery phase, the optimal repair strategy, stored in the Repair_Strategy_Opt, goes into the Repair Strategy register and the strategy is evaluated again and the final analysis ends up with the optimal solution. Hard repair is done using this optimal solution. Using the optimal repair strategy, corrupted rows or columns of the memory is replaced with spare rows or spare columns using the reconfiguration mechanism. Thus the hard errors are eliminated from the memory. Built in Self Repair scheme is not used for correcting soft errors since the soft errors will not cause damages in the memory.

2. Phases of the Integrated ECC and BISR Scheme

The Integrated ECC and Redundancy Repair Scheme is divided into three phases, i.e., the Fault Free phase, Error Identification phase and the Hard Repair phase.

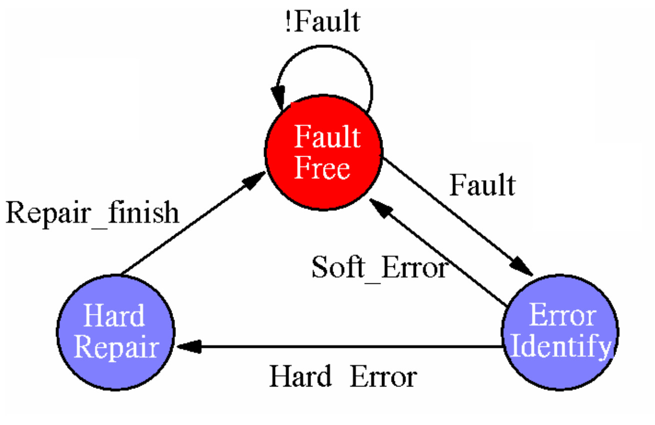


Fig -9: Phase diagram of the Integrated ECC and BISR

The phase (state) transitions are shown in Fig. 9. Fault free memory is the first phase. In that phase, fault is continuously monitored. If a fault is detected, then the control moves to the next phase where error is identified. In this phase, soft errors are corrected by Error Correction Code. After soft repair, control moves to the first phase. If the error is a hard error, then control moves to the third phase. After hard repair, control moves to the first phase and remains in that phase until a fault is detected. Hard repair is done using BISR. In this method, corrupted rows or columns of the memory is replaced with spare rows and columns.

When the hard repair is completed, control moves to the first phase. During fault-free normal operation, the fault tolerance mechanism is not activated, and the system stays in the Fault Free state. When a fault is detected by ECC, the system will immediately execute an Error Identification procedure to determine whether the fault is a hard fault or a soft error.

After error identification the system may transit either to the Fault Free state or the Hard Repair state, depending on the error type identified. Assume that during the Error Identification phase, no other errors may occur. This assumption is reasonable because the error rate is normally very low compared with the system clock speed.

The simulation results for the error correction using Integrated ECC and BISR Scheme is obtained as shown in figure 10. There are 4 modes of operation. They are write mode, read mode and update mode. When rst=1, reset operation is done. When rst=1, BISR activates.

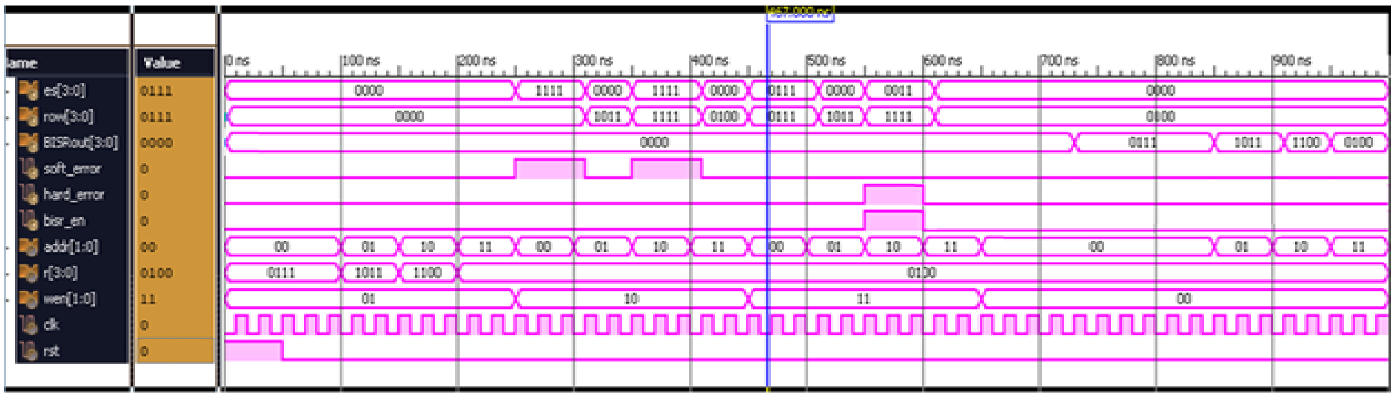


Fig -10: Wave form of the Integrated ECC and BISR

When enable signal (wen) is 01, write mode is activated. Data is written into the memory. Now the memory is in fault free phase. When enable signal (wen) is 10, read mode is activated. Wrong data is read from the memory due to error. Control moves to the error identification phase. Wrong data is obtained from the first and third row, error signal raises while reading that rows. Soft error signal rises to show that soft error occurs in the memory. ECC activates and soft error correction is done in the memory. Hard errors can be corrected but can't be eliminated using ECC. When enable (wen) is 11, data is again read from the memory. Location of error can be obtained from the es signal. While reading the 3rd row, hard error signal raises and wrong data is obtained.

When the hard fault is detected, BIST module activates. Control moves to the third phase and BISR module activates. Fault address is detected using BIST. Redundancy analyzer identifies must repair rows and columns and searches for a repair solution. Reconfiguration mechanism replaces the repair rows or columns with spare elements. Thus, we can eliminate hard error from the memory. When enable signal (wen) is 00, update mode is activated. Corrected data is obtained from the output of BISR. Thus both hard errors and soft errors are eliminated from the memory.

3. CONCLUSIONS

Integrated ECC and BISR Scheme a new scheme to identify hard faults and soft errors, and progressively repair the hard faults by available redundancy in normal operation mode. This new scheme integrates the on-line ECC method and an off-line BISR scheme. The integrated scheme indeed improves the reliability of a memory chip, as it combines the benefits of redundancy repair and ECC. The approach also provides a cost-effective way to solve the problem of growing parametric defects in deep-submicron SOC products, and is good for other products demanding high reliability.

The drawback of the proposed Integrated ECC and BISR Scheme is that more storage space is required to store the spare rows, spare columns and redundant bits to maintain higher reliability of memory. A reasonable combination of k and m should be chosen to maximize memory reliability and minimize the number of redundant bits in actual implementation. Therefore, future work will be conducted for the reduction of the redundant bits, spare elements and the maintenance of the reliability of the proposed technique.

REFERENCES

[1] Jaeyong Chung, Joonsung Park, Jacob A. Abraham, "A Built-In Repair Analyzer With Optimal Repair Rate for Word-Oriented Memories", IEEE Trans. On VLSI Systems, Vol. 21, no. 2, February 2013.
[2] Jing Guo, Liyi Xiao, Zhigang Mao , Qiang Zhao , "Enhanced Memory Reliability Against Multiple Cell Upsets Using Decimal Matrix Code", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 22, no. 1, January 2014.
[3] P. Oehler, A. Bosio, G. D. Natale, and S. Hellebrand, "A modular memory BIST for optimized memory repair," in Proc. Int. On-Line Test. Symp., pp. 171–172,2008.
[4] Costas Argyrides, Dhiraj K Pradhaan, Taskin Kocak, "Matrix-based codes for Reliable and Cost Efficient Memory Chips," IEEE Trans. On VLSI Systems, vol. no. 19, no. 3, March 2011.
[5] Reviriego, M. Flanagan, J. A. Maestro, "A (64,45) triple error correction code for memory applications," IEEE Trans. Device Mater.Rel.,vol.12, no.1, pp.101–106, Mar. 2012.