A Review Paper on Memory Fault Models and its Algorithms

Dr. Kendaganna Swamy S
Dept. of Electronics and Instrumentation
R V College of Engineering
Bengaluru, India
kendagannaswamys@rvce.edu.in

Dr. Rajasree P M
Dept. of Electronics and Instrumentation
R V College of Engineering
Bengaluru, India
rajasreepm@rvce.edu.in

Anand M Sharma
Dept. of Electronics and Instrumentation
R V College of Engineering
Bengaluru, India
anandmsharma9@gmail.com

Jnanaprakash J Naik
Dept. of Electronics and Communictation
R V College of Engineering
Bengaluru, India
jnanaprakashjn@gmail.com

***Abstract*—The significance of testing semiconductor memories has surged in the industry due to the higher density of modern memory chips. This paper investigates functional faults prevalent in today's memory technology: coupling faults, address decoder faults, transition faults, stuck-at faults, and neighbourhood pattern-sensitive faults. It delves into techniques to identify these issues, focusing on zero-one, checkerboard, and March pattern tests across chip, array, and board levels. The study explores test algorithms, assessing fault coverage. Overall, it offers insights into challenges posed by dense memory chips and a comprehensive analysis of functional faults. Notably, MARCH algorithms outperform others in fault coverage, power efficiency, area optimization, and time complexity, suggesting their preference for reliable high-performance memory devices in the electronics industry.**

Keywords— Memory Faults, Memory Test Algorithms, BIST (Built in Self-Test), Memory array, March test, MATS, Up Transition, Down Transition, MSCAN, DFT, Fault coverage

I. INTRODUCTION

This review paper explores various memory fault models and their associated detection algorithms. Memory faults pose significant challenges in the reliable operation of electronic systems, and understanding their characteristics and detection techniques is crucial for ensuring system robustness.

Let's analyse an example summary table that demonstrates how various memory sizes are checked using various memory methods, along with their time complexity. The table below demonstrates patterns utilized at a frequency of 100M write or read operations per second for the computations.

Table 1: Example Summary Table.

Memory Size	Time Complexity (n)	Time Complexity (n log n)	Time Complexity (n^1.5)	Time Complexity (n^2)
1 Kilo Byte	0.0001 seconds	0.001 seconds	0.0033 seconds	0.105 seconds
1 Mega Byte	0.102 seconds	2.04 seconds	1.83 minutes	1.27 days
1 Giga Byte	1.75 minutes	52.48 minutes	40.8 days	3659 years

We can infer from the above Table 1 how lengthy memory testing might be if an appropriate test algorithm is not employed. An algorithm spanning several days or years doesn't follow a linear time pattern. Such algorithms are intolerable and are not supported by the semiconductor industry.

The paper investigates several memory fault models, including coupling faults, transition faults, stuck-at faults, and address decoder faults. To address these memory faults, the paper discusses various detection algorithms commonly employed in practice. Stuck-at faults occur when a specific bit in memory remains constantly stuck at either 0 or 1. Transition faults involve errors during state transitions, leading to incorrect data propagation. Coupling faults arise from interactions between adjacent memory cells, potentially causing data corruption. Address decoder faults pertain to issues in memory address decoding, resulting in incorrect memory access [1][3]. The zero-one algorithm, known for its simplicity, aims to detect stuck-at faults by ensuring that both 0 and 1 values are observed during memory read operations. The checkerboard algorithm verifies memory integrity by writing alternating 0’s and 1’s and then reading back to identify any inconsistencies. March algorithms, a family of sophisticated test algorithms, systematically march through memory locations, stimulating specific patterns and detecting faults based on observed responses [5][6].

By thoroughly reviewing these memory fault models and detection algorithms, this paper offers valuable perspectives into the challenges associated with ensuring memory reliability in electronic systems. The findings contribute to the advancement of fault-tolerant designs and enhance the overall dependability of memory subsystems

II. MEMORY FAULT MODELS

A. *Stuck at Fault (SAFs):*

A stuck-at fault (SAF) arises when a cell or line consistently holds a value of either 0 (known as a stuck-at-0 fault as shown in Figure 1) or 1 (known as a stuck-at-1 fault as shown in Figure 2). An effective test for identifying all SAFs ensures that both 0 and 1 can be observed when reading from each cell [2].

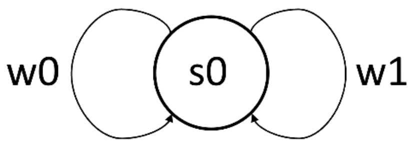

Figure 1: Memory cell having s-a-0 fault.

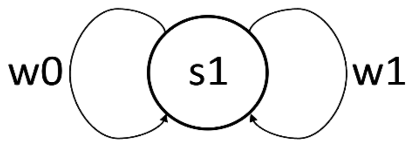

Figure 2: Memory cell having s-a-1 fault.

B. *Transition Fault (TF):*

Transition faults (TFs) are specific faults that can occur in digital circuits when a cell or logic element fails to transition from one state to another during a write operation. In normal operation, signals transition between the 0 and 1 states representing low and high voltage levels. However, transition faults can result in cells getting stuck in a particular state, either 1 or 0, when they were supposed to transition.

There are two types of transition faults: up-transition faults and down-transition faults.

i) In Figure 3 an up-transition fault (< ↑ | 0 >) occurs when a cell fails to transition from 0 to 1 as expected, which leads the output remaining stuck at 0.

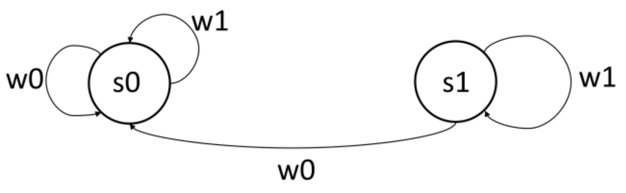

Figure 3: Memory cell having up transition faults.

j) In Figure 4 an down-transition fault (< ↓ | 1>) happens when a cell fails to transition from 1 to 0, which leads the output to remain stuck at 1.

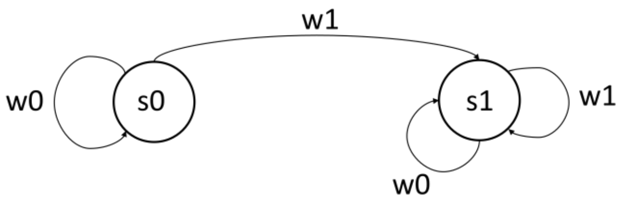

Figure 4: Memory cell having down transition faults.

C. *Address Decoder Faults (ADFs):*

These faults can affect the decoding circuitry, which interprets memory addresses and chooses the proper memory cell for read or write operations, in memory systems [1]. Decoding the memory address bus and producing control signals to choose the proper memory cell for the desired operation are tasks carried out by the address decoder. The memory's address decoder comprises both a row and column decoder. The analysis focuses on considering four types of defects in the address decoder concerning memory testing [11][7].

AF1: It is impossible to access any memory cell with a specific address.

AF2: There is no address that allows access to a specific memory cell.

AF3: Multiple memory cells can be accessed at the same time using certain addresses.

AF4: One particular memory cell can be accessed through multiple addresses.

D. *Coupling Faults (CFs):*

Coupling faults (CFs) manifest as a fault within a cell in a digital circuit, resulting from its interaction or coupling with adjacent cells. They emerge when a cell's function is influenced by neighbouring cells, causing irregular or incorrect operations.

Considering coupling faults, there's a potential for an exponential array of combinations in which a cell can couple with other cells within a circuit. Each coupling combination can result in unique fault behaviour, making it challenging to identify and diagnose these faults. The widely used coupling fault model assumes that any two cells in the circuit can be coupled together, resulting in abnormal behaviour or faulty operation within those two cells. This model is known as the 2-cell coupling fault model, as it focuses on the interaction between pairs of cells [7].

Considering a memory with n cells, the number of 2-cell coupling faults can be determined using the combination formula nC2, also known as "n choose 2". This formula calculates the number of ways to choose 2 cells out of a total of n cells for coupling analysis. The result represents the number of unique coupling combinations and, consequently, the potential number of 2-cell coupling faults present in the circuit. This is represented in Figure 5.

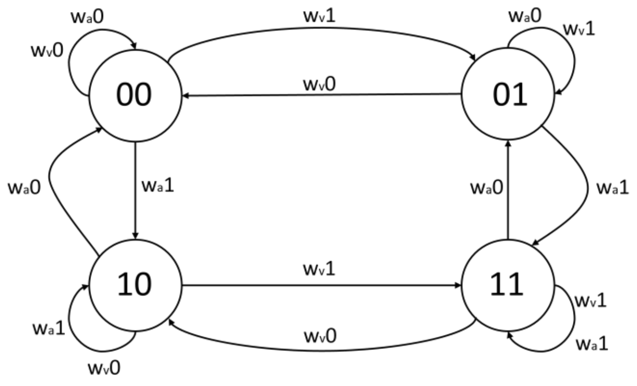

Figure 5: Write operation performed between two good memory cells.

i. **Inversion Coupling Faults (CFin):**

Inversion Coupling Faults (CFin) represent a distinct subtype of coupling fault encountered in digital memory cells. These faults arise from a write operation in an aggressor word, causing an upper (0 to 1) or lower (1 to 0) transition, resulting in an inversion within the cell of a victim word.

Regarding CFin, the aggressor word refers to the word or cell that triggers the coupling effect, while the victim word is the word or cell that experiences the inversion due to the coupling. When an aggressor word undergoes a write operation with a transition from 0 to 1 (rising) or from 1 to 0 (falling), it causes an inversion in the content of the victim cell.

The two types of CFin are rising and falling inversions, each corresponding to a specific transition in the aggressor word.

As represented in Figure 6, the rising inversion is represented as < ↑ | ↕ >, indicating that a change from 0 to 1 in the aggressor cell complements or inverts the content of the victim cell.

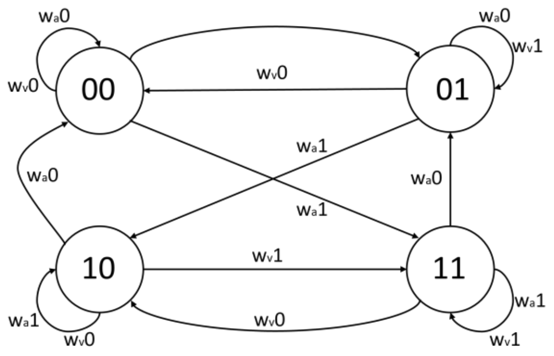

Figure 6: Memory cell having rising inversion coupling faults.

As represented in Figure 7, the falling inversion is represented as < ↓ | ↕ >, indicating that a change from 1 to 0 in the aggressor cell complements the content of the victim cell.

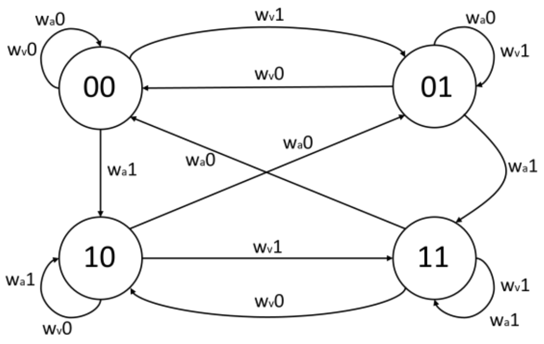

Figure 7: Memory cell having falling inversion coupling faults.

It is important to note that inversion coupling faults are not typically observed in faulty memory cells and are defined mainly for historical reasons. Therefore, they are not included in the linked faults list, which refers to the list of faults identified as actual observable faults in a memory system [12].

ii. Idempotent Coupling Faults (CFid):

Idempotent coupling faults (CFid) represent a unique category of coupling fault found in digital systems, triggered by a write operation on an aggressor word cell that imposes a specific value (0 or 1) onto a victim word cell. These faults, falling within the broader category of coupling faults (CFs), are distinguished by the write operation causing a reversal in the victim cell's content from its prior state [15].

There are four variations of idempotent coupling faults:

1) **Rising 0**: In this scenario, denoted as < ↑ | 0 >, a 0 to 1 transition in an aggressor word cell leads to the content of the victim cell being set to 0.This is represented in Figure 8.

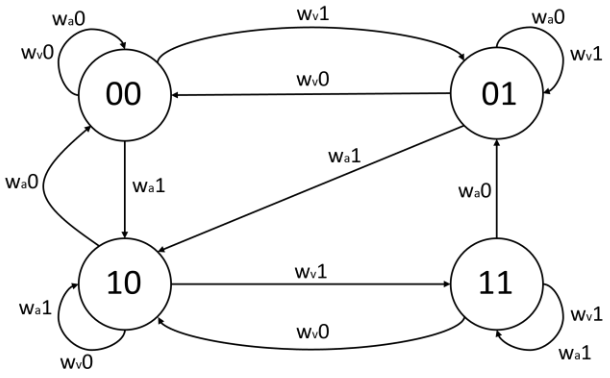

Figure 8: Memory cell having rising 0 idempotent coupling faults.

2) **Rising 1**: Represented as < ↑ | 1 >, this fault occurs when a 0 to 1 transition in an aggressor word cell results in the content of the victim cell being set to 1.This is represented in Figure 9.

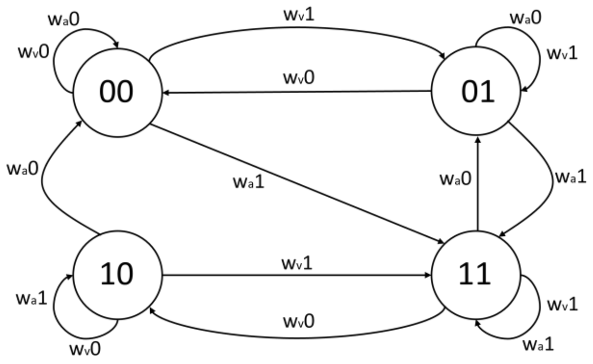

Figure 9 Memory cell having rising 1 idempotent coupling faults.

3) **Falling 0**: This fault is indicated by < ↓ | 0 > and occurs when a 1 to 0 transition in an aggressor word cell causes the content of the victim cell to be set to 0. This is represented in Figure 10.

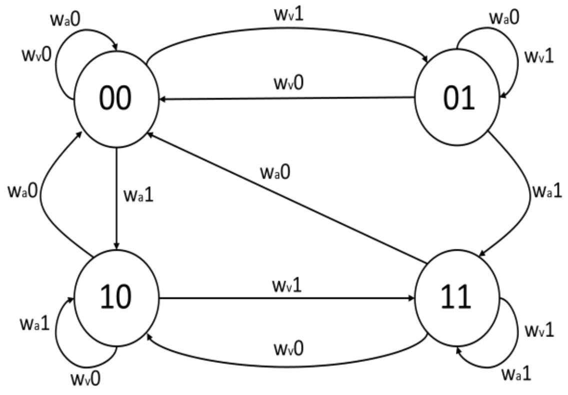

Figure 10: Memory cell having falling 0 idempotent coupling faults.

4) **Falling 1**: Denoted as < ↓ | 1 >, this fault happens when a 1 to 0 transition in an aggressor word cell sets the

content of the victim cell to 1. Figure 11 depicts this information.

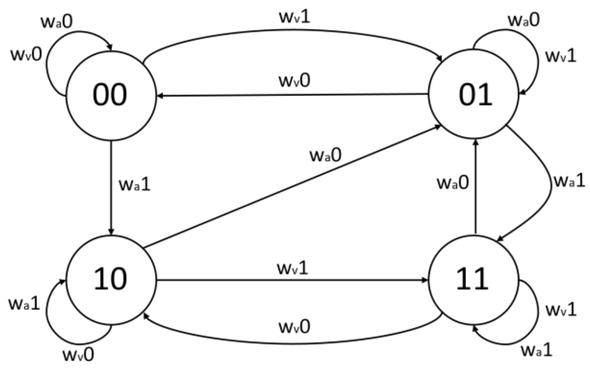

Figure 11: Memory cell having falling 1 idempotent coupling faults.

In each of these fault scenarios, the write operation on the aggressor cell results in imposing a specific value on the victim cell, replacing its previous content. Such idempotent coupling faults may result in circuit malfunctions and data corruption.

Identifying and addressing idempotent coupling faults is essential for ensuring the integrity and reliability of digital systems. Techniques such as fault simulation, testing, and analysis are employed to detect and mitigate these faults during the design, manufacturing, or maintenance stages of the circuit, helping to enhance overall system performance and dependability.

iii. Static Coupling Faults (CFst):

Static coupling faults (CFst) are a distinct subtype of coupling fault, arising when a particular value (0 or 1) in an aggressor word cell imposes or influences a specific value (0 or 1) in a cell belonging to another word, known as the victim word. These faults originate from the interaction or coupling between the aggressor and victim cells.

There are four possible scenarios that describe static coupling faults:

1) **(Cell A containing 0 triggers the cell V to be 0)**: In this scenario, when the value of cell a in the aggressor word is 0, it influences the cell v in the victim word to also have a value of 0. The coupling between these cells causes the content of cell v to be forced to 0. This is represented in Figure 12.

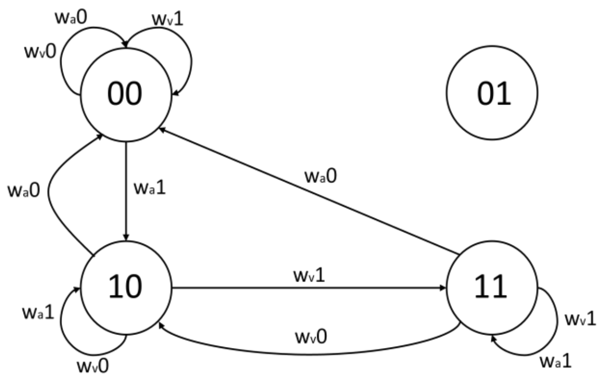

Figure 12: Cell A containing 0 triggers the cell V to be 0.

2) **(Cell A containing 0 triggers the cell V to be 1)**: Here, if cell a in the aggressor word has a value of 0, it causes the content of cell v in the victim word to be forced to 1. The coupling between the cells leads to an undesired change in the content of cell v. This is represented in Figure 13.

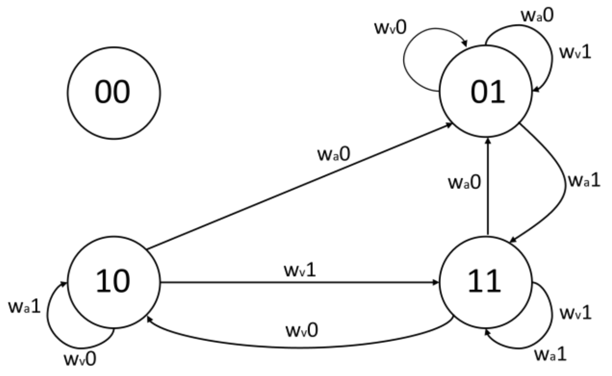

Figure 13: Cell A containing 0 triggers the cell V to be 1.

3) **(Cell A containing 1 triggers the cell V to be 0)**: This scenario occurs when cell a in the aggressor word has a value of 1, resulting in the content of cell v in the victim word being forced to 0. The coupling between these cells causes an unexpected change in the content of cell v. This is represented in Figure 14.

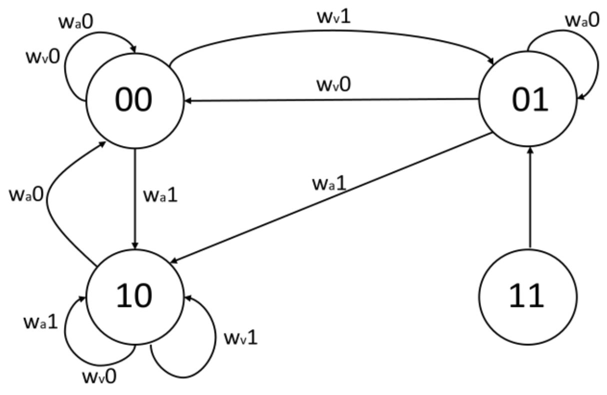

Figure 14: Cell A containing 1 triggers the cell V to be 0.

4) **(Cell A containing 1 triggers the cell V to be 1)**: In this case, when the value of cell a in the aggressor word is 1, it forces the content of cell v in the victim word to be 1. The interaction among these cells causes a modification in the content of cell V, as depicted in Figure 15.

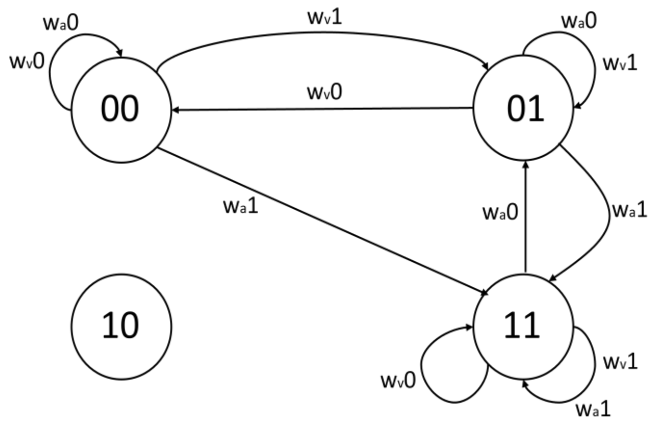

Figure 15: Cell A containing 1 triggers the cell V to be 1.

E. Neighbourhood Pattern Sensitive Coupling Faults:

Neighbourhood pattern sensitive coupling faults represent a distinct category of faults that emerge from the interplay between a cell under examination and the arrangement of neighbouring cells within a digital circuit. In this type of fault, the behaviour of the affected cell, known as the victim cell, undergoes an influence or alteration based on the values or states exhibited by the neighbouring cells. The neighbourhood of a cell refers to the collection of cells in close proximity or directly connected to the cell being tested. These neighbouring cells possess the potential to exert a significant impact on the performance and functionality of the victim cell [10].

In the occurrence of a neighbourhood pattern sensitive coupling fault, the interaction between the examined cell and its neighbouring cells induces the victim cell to display inaccurate behaviour. The distinctive pattern formed by the values or states of neighbouring cells introduces a disturbance that hampers the operation of the victim cell, leading to flawed outputs or compromised functionality.

The coupling effect can manifest in various forms, including the introduction of electrical noise, signal interference, or unintended propagation of signals between cells. In Figure 16, the particular coupling pattern formed by the neighbouring cells serves as the determining factor for the nature of the faulty behaviour exhibited by the victim cell.

Type-1 Neighbourhood Faults: The blue coloured called as base cell. The base cell is the cell which undergoes testing when the four cells around it is in the coupling state. The four cells (pink colour blocks) are neighbourhood cells for the base cell. This fault pattern is shown in Figure 16 and example of this is describes in Figure 17.

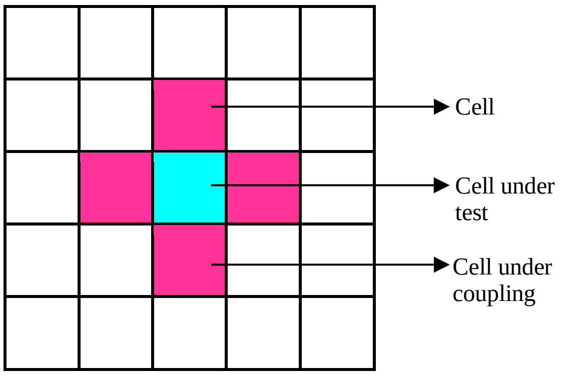

Figure 16: Pattern of Type-1 Neighbourhood Faults.

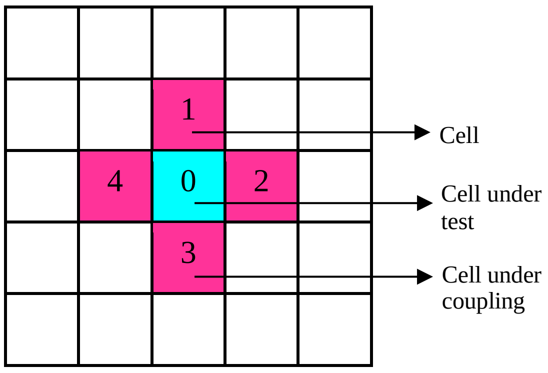

Figure 17: Example of Type-1 Neighbourhood Faults.

Type-2 Neighbourhood Faults: It has eight neighbourhood cells corresponding around the cell under test. These faults exhibit increased complexity in contrast to type-1 neighbourhood faults. The fault pattern is depicted in Figure 18, and an example is outlined in Figure 19.

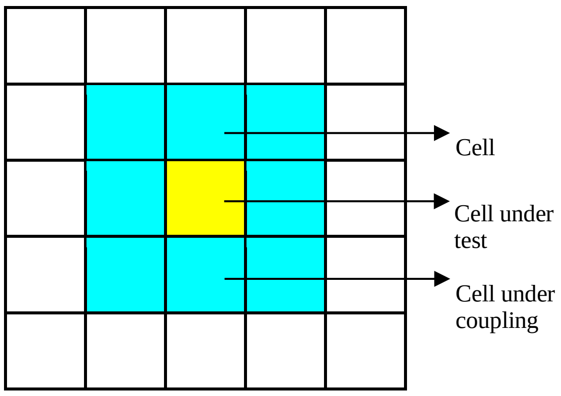

Figure 18: Pattern of Type-2 Neighbourhood Faults

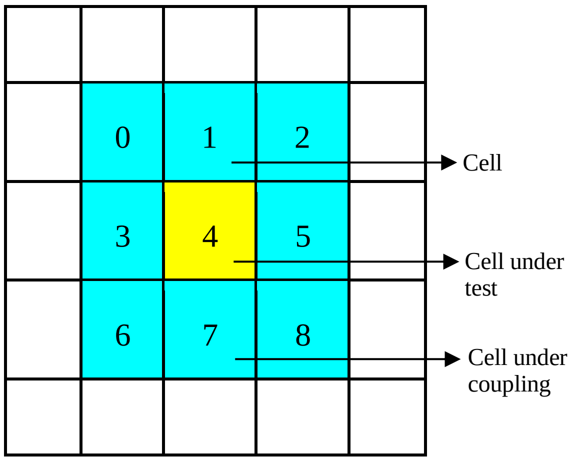

Figure 19: Example of Type-2 Neighbourhood Faults

III. MEMORY FAULT DETECTION ALGORITHMS

A. MSCAN Algorithm

MSCAN Algorithm is also known as the zero-one algorithm. This algorithm has four major steps:

- Write zero to every cell.
- Read zero from every cell.
- Write one to every cell.
- Read one from every cell.

This Algorithm detects all stuck at faults (SAF) and rising transition faults but fails to detect falling transition faults. It also fails to detect all Address decoder faults and Coupling faults. The complexity of the circuit is 4N as 4 operations takes place in each cell.

B. Chekerboard Algorithm

Checkerboard is a commonly used algorithm in memory Built-In Self-Test (BIST) for detecting and locating faults in memory arrays. It is based on the principle of alternating memory patterns to detect faults in the memory array [12]. The checkerboard algorithm works by writing a specific pattern of alternating 1s and 0s to the memory array. The pattern is written in a way that creates a checkerboard-like pattern of alternating 1s and 0s. Once the pattern is written, the algorithm reads back the data and checks for any errors or faults in the memory array [8].

The time complexity of the checkerboard algorithm aligns with that of a zero-one algorithm, both operating at 4N. Primarily employed for identifying faults arising from leakage, cell shorts, and data retention issues, the checkerboard algorithm additionally identifies SAFs (Stuck-at faults) and half the number of TFs (Transition faults).

This Algorithm has 4 major steps

•Write checkerboard with up addressing order.

•Read checkerboard with up addressing order.

•Write inverse checkerboard with up addressing order.

•Read inverse checkerboard with up addressing order

C. Marching Algorithm

March Algorithms are used to detect single bit error. Faults can be manifested as errors. In this algorithm it performs two operations. One is read and other is write operation. The main aim is to read and write the address with a finite sequence. It comprises a sequence of March elements in its test pattern. This algorithm is greatly used in testing of memories. There are some March notations like [17]:

$\Uparrow$: It indicates the accessing of memory location from lower memory address to higher memory address.

$\Downarrow$: It indicates the accessing of memory location from higher memory address to lower memory address.

$\Updownarrow$: It is used to access user defined memory locations in both directional.

r0: It performs read 0 operation in the cell.

r1: It performs read 1 operation in the cell.

w0: It performs write 0 operation in the cell.

W1: It performs write 1 operation in the cell.

This Algorithm has 4 major steps:

Increasing Address

- write 0s with up addressing order (to initialize)
- Read 0s, write 1s with up addressing order
- Read 1s, write 0s with up addressing order

Decreasing address

- Read 0s, write 1s with down addressing order
- Read 1s, write 0s with down addressing order
- Read 0s with down addressing order

i. MATS

MATS, which stands for Modified Algorithmic Test Sequence, is a compact MARCH test used for unlinked SAFs (Stuck-At Faults) in memory cell arrays and read/write logic circuits. By treating the collective reading of multiple cells as an OR function of their contents, this algorithm is capable of detecting all faults in OR-type technology[6]. Moreover, the MATS Algorithm can also be applied to AND-type technology by utilizing the MATS-AND test sequence provided below. With a complexity of 4N, the MATS Algorithm offers superior fault coverage compared to equivalent zero-one and checkerboard tests [10].

$MATS - ORtype$:
$\{$
$\Updownarrow (w0); \Updownarrow (r0, w1); \Updownarrow (r1)$
$\}$
$MATS - ANDtype$:
$\{$
$\Updownarrow (w1); \Updownarrow (r1, w0); \Updownarrow (r0)$
$\}$

ii. MATS+

The MATS+ test sequence detects all SAF's and AF's, its often used instead of MATS when the technology used under test is unknown. The MATS+ algorithm has a test complexity of 5N.

$MATS+$:
$\{$
$\Updownarrow (w0); \Uparrow (r0, w1); \Downarrow (r1, w0)$
$\}$

iii. MATS++

It is any extension of MATS+ algorithm. It has the capability to identify faults such as SAF, AF and TF. In this algorithm the process is complete with no repetition. It has complexity of 6N.

$MATS + +$:
$\{$
$\Updownarrow (w0); \Uparrow (r0, w1); \Downarrow (r1, w0, r0)$
$\}$

iv. MARCH X

It has the capability to identify faults such as AF, TF, SAF and some CF. It has complexity of 6N. There is no repetition in this algorithm. It has simple test sequence to detect four faults. This is represented in Table 2.

$MARCH\ X$:
$\{$
$\Updownarrow (w0); \Uparrow (r0, w1); \Downarrow (r1, w0); \Updownarrow (r0)$
$\}$

v. MARCH Y

It is similar to March X algorithm. So, the complexity is increased to 8N. It can detect faults like SAF, TF, AF and some CF.

$MARCH\ Y$:
$\{$
$\Updownarrow (w0); \Uparrow (r0, w1, r1); \Downarrow (r1, w0, r0); \Updownarrow (r0)$
$\}$

Table 2 Fault coverage of different fault simulation algorithms for various fault models

ALG	**Stuck at Fault**	**Transition Fault**	**Address Decoder Fault**	**Coupling Fault**	**Neighbor hood Pattern Sensitive Fault**
MSCAN	Yes	Rising (< ↑ | 0 >)	No	No	No
Checkerboard	Yes	Half (either < ↑ | 0 > or < ↓ | 1 >)	No	No	No
MATS	Yes	No	Yes	No	No
MATS+	Yes	Yes	Yes	No	No
MATS++	Yes	Yes	Yes	No	No
March X	Yes	Yes	Yes	Yes	No
March Y	Yes	Yes	Yes	Yes	No
March A	Yes	Yes	Yes	Yes	No
March B	Yes	Yes	Yes	Yes	No
March C	Yes	Yes	Yes	Yes	No
March C-	Yes	Yes	Yes	Yes	No

vi. MARCH A

It has complexity of 15N. Here four elements as group in each term. It can detect faults like SAF, TF, AF and some CF. It is irredundant algorithm.

$$
\begin{aligned}
&MARCHA:\\
&\{\\
&\Updownarrow (w0); \Uparrow (r0, w1, w0, w1); \Uparrow (r1, w0, w1);\\
&\Downarrow (r1, w0, w1, w0); \Downarrow (r0, w1, w0)\\
&\}
\end{aligned}
$$

vii. MARCH B

It is similar to March A algorithm. So, the complexity is increased to 17N. It has the capability to identify faults such as AF, TF, SAF and CF.

$$
\begin{aligned}
&MARCHB:\\
&\{\\
&\Updownarrow (w0); \Uparrow (r0, w1, r1, w0, r0, w1); \Uparrow (r1, w0, w1);\\
&\Downarrow (r1, w0, w1, w0); \Downarrow (r0, w1, w0)\\
&\}
\end{aligned}
$$

viii. MARCH C

It has complexity of 15N. It has the capability to identify faults such as AF, TF, SAF and CF. Here there is a repetition process so complexity is more.

$$
\begin{aligned}
&MARCH\ C:\\
&\{\\
&\Updownarrow (w0); \Uparrow (r0, w1); \Uparrow (r1, w0);\\
&\Updownarrow (r0); \Downarrow (r0, w1); \Downarrow (r1, w0); \Updownarrow (r0)\\
&\}
\end{aligned}
$$

ix. MARCH C-

It is the extension of March C algorithm. It has complexity of 10N. It can detect faults like SAF, TF, AF and CF. Here no repetition occurs.

$$
\begin{aligned}
&MARCH\ C-:\\
&\{\\
&\Updownarrow (w0); \Uparrow (r0, w1); \Uparrow (r1, w0);\\
&\Downarrow (r0, w1); \Downarrow (r1, w0); \Updownarrow (r0)\\
&\}
\end{aligned}
$$

Analysis of algorithms and their fault coverage concerning Time complexity, Power dissipation for a 4 KB memory [20], and Area overhead in terms of gate count for a 4 KB system. This is represented in Table 3.

Table 3 Comparison of Power, Area and Timing.

Algorithms	**Time Complexity**	**Fault Coverage**	**Power Dissipation (mW)**	**Area Overhead**
MSCAN	4N	Limited	70 - 80	280 - 300
Checker board	4N	Limited	80 - 90	300 -340
MATS	4N	High	80 - 90	360 - 380
MATS+	5N	High	90 -100	400 - 430
MATS++	6N	High	100 -120	420 - 450
March X	6N	High	720 - 740	480 - 500
March Y	8N	High	750 - 765	510 - 530
March A	15N	High	690 -710	620 - 640
March B	17N	Very High	745 - 765	640 - 660
March C	15N	Very High	700 - 720	580 - 600
March C-	10N	Very High	680 - 700	560 -580

The symbol 'N' represents the duration required for individual read and write operations within the memory.

CONCLUSION

The review paper discusses the predominant utilization of functional testing, encompassing tests such as zero-one, checkerboard, and March patterns, particularly in SRAM and DRAM technologies. These tests try to find different types of faults, such as stuck-at faults or neighbourhood pattern-sensitive faults [11]. From this analysis, it's apparent that the MSCAN algorithm consumes fewer gates, resulting in lower power dissipation. However, it exhibits very poor fault coverage. Conversely, March C- boasts the highest fault coverage but entails increased power dissipation and area overhead. Balancing these factors necessitates sacrificing one parameter for better performance. Compared to traditional methods like MSCAN and Checkerboard, the MATS, MATS+, MarchX, MarchA, MarchY, and MarchB algorithms showcase superior efficiency and fault coverage [13]. Despite ongoing enhancements aimed at bolstering fault coverage in existing algorithms, there remains a critical need for a novel algorithm capable of efficiently detecting a wide array of fault types [14]. As semiconductor memory density escalates, research persistently pursues advanced pattern sequences and alternative strategies such as DFT and BIST to fortify testing capabilities. These efforts are aimed at meeting the evolving challenges posed by advancing semiconductor technologies. Emerging alternatives like MATP, GALPAT, Butterfly, and Signature Analysis using LFSR promise enhanced results, although the trade-off between parameters remains an inevitable consideration.

REFERENCES

[1] T. Q. Bui, L. D. Pham, H. M. Nguyen, V. T. Nguyen, T. C. Le, and T. Hoang, "An effective architecture of memory built-in self-test for wide range of sram," in 2016 International Conference on Advanced Computing and Applications (ACOMP), 2016, pp. 121–124. doi: 10.1109/ACOMP.2016.026.

[2] A. Z. Jidin, R. Hussin, L. Fook, and M. S. Mispan, "A review paper on memory fault models and test algorithms," Bulletin of Electrical Engineering and Informatics, vol. 10, pp. 3083–3093, Dec. 2021. doi: 10.11591/eei.v10i6.3048.

[3] S. Al-Harbi and S. Gupta, "An efficient methodology for generating optimal and uniform march tests," in Proceedings 19th IEEE VLSI Test Symposium. VTS 2001, 2001, pp. 231–237. doi: 10.1109/VTS.2001.923444.

[4] M. Parvathi, N. Vasantha, and K. Parasad, "Modified march c - algorithm for embedded memory testing," International Journal of Electrical and Computer Engineering, vol. 2, pp. 571–576, 2012.

[5] S. Hamdioui, R. Wadsworth, J. Delos Reyes, and A. van de Goor, "Importance of dynamic faults for new sram technologies," in The Eighth IEEE European Test Workshop, 2003. Proceedings., 2003, pp. 29–34. doi: 10.1109/ETW.2003.1231665.

[6] T. Koshy and C. S. Arun, "Diagnostic data detection of faults in ram using different march algorithms with bist scheme," in 2016 International Conference on Emerging Technological Trends (ICETT), 2016, pp. 1–6. doi: 10.1109/ICETT.2016.7873754.

[7] W. T. Hale and G. M. Bollas, "Design of built-in tests for active fault detection and isolation of discrete faults," IEEE Access, vol. 6, pp. 50 959–50 973, 2018. doi: 10.1109/ACCESS.2018.2869269.

[8] G. S. Lakshmi, K. Neelima, and D. Subhas, "A march ns algorithm for detecting all types of single bit errors in memories," Journal of emerging technologies and innovative research, 2019.

[9] A. Singh, G. M. Kumar, and A. Aasti, "Controller architecture for memory bist algorithms," in 2020 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS), 2020, pp. 1–5. doi: 10.1109/SCEECS48394. 2020.43.

[10] K.-L. Cheng, M.-F. Tsai, and C.-W. Wu, "Neighborhood pattern-sensitive fault testing and diagnostics for random-access memories," IEEE Transactions on ComputerAided Design of Integrated Circuits and Systems, vol. 21, no. 11, pp. 1328–1336, 2002. doi: 10.1109/TCAD.2002.804101.

[11] S. Wang, T. Aono, Y. Higami, et al., "Capture-pattern-control to address the fault detection degradation problem of multi-cycle test in logic bist," in 2018 IEEE 27th Asian Test Symposium (ATS), 2018, pp. 155–160. doi: 10.1109/ATS.2018.00038.

[12] A. K. S. Pundir and O. P. Sharma, "Fault tolerant reconfigurable hardware design using bist on sram: A review," in 2017 International Conference on Intelligent Computing and Control (I2C2), 2017, pp. 1–16. doi: 10.1109/I2C2.2017.8321907.

[13] S. Hamdioui, A. van de Goor, and M. Rodgers, "March ss: A test for all static simple ram faults," in Proceedings of the 2002 IEEE International Workshop on Memory Technology, Design and Testing (MTDT2002), 2002, pp. 95–100. doi: 10. 1109/MTDT.2002.1029769.

[14] R. Dekker, F. Beenker, and L. Thijssen, "A realistic fault model and test algorithms for static random access memories," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 9, no. 6, pp. 567–572, 1990. doi: 10.1109/ 43.55188.

[15] Nair, Thatte, and Abraham, "Efficient algorithms for testing semiconductor randomaccess memories," IEEE Transactions on Computers, vol. C-27, no. 6, pp. 572–576, 1978. doi: 10.1109/TC.1978.1675150.

[16] M.-C. V. Marinescu, "Simple and efficient algorithms for functional ram testing," in International Test Conference, 1982.

[17] I. Hwang, S. Kim, Y. Kim, and C. E. Seah, "A survey of fault detection, isolation, and reconfiguration methods," IEEE Transactions on Control Systems Technology, vol. 18, no. 3, pp. 636–653, 2010. doi: 10.1109/TCST.2009.2026285.

[18] J. Rosero, J. Ortega, E. Aldabas, and L. Romeral, "Moving towards a more electric aircraft," IEEE Aerospace and Electronic Systems Magazine, vol. 22, no. 3, pp. 3–9, 2007. doi: 10.1109/MAES.2007.340500.

[19] L. Warrington, J. Jones, and N. Davis, "Modelling of maintenance, within discrete event simulation," in Annual Reliability and Maintainability Symposium. 2002 Proceedings (Cat. No.02CH37318), 2002, pp. 260–265. doi: 10 . 1109 / RAMS . 2002 . 981652.

[20] Lakha, Balwinder & Khosla, Arun & Narang, Sukhleen. (2012). Area Overhead and Power Analysis of March Algorithms for Memory BIST. Procedia Engineering. 30. 930-936. 10.1016/j.proeng.2012.01.94