Indonesian Journal of Electrical Engineering and Computer Science
Vol. 43, No. 2, August 2026, pp. 400~412
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v43.i2.pp400-412

A comprehensive review of memory BIST algorithms for SRAM static fault detection

Aiman Zakwan Jidin[1], Razaidi Hussin[2], Mohd Syafiq Mispan[1], Lee Weng Fook[3], Loh Wan Ying[2]
[1]Fakulti Teknologi dan Kejuruteraan Elektronik dan Komputer, Universiti Teknikal Malaysia Melaka, Durian Tunggal, Malaysia
[2]Fakulti Kejuruteraan dan Teknologi Elektronik, Universiti Malaysia Perlis, Arau, Malaysia
[3]Emerald System Sdn Bhd, Bayan Lepas, Malaysia

Article Info

Article history:

Received Sep 6, 2024
Revised May 26, 2026
Accepted Jun 28, 2026

Keywords:

March test algorithm
MBIST
Memory fault coverage
SRAM
Static faults

ABSTRACT

Memory built-in self-test (MBIST) has become an essential design-for-testability technique for ensuring the reliability and quality of embedded memories in modern integrated circuits. The effectiveness of an MBIST implementation is largely determined by its test algorithm, which defines the sequence of memory operations, directly influencing both test complexity and fault coverage. Designing an efficient test algorithm requires balancing low test complexity with comprehensive fault detection. This paper presents a comprehensive review of MBIST test algorithms for static fault detection in static random-access memory (SRAM). The review first summarizes the characteristics of major SRAM static faults and their corresponding detection requirements. It then compares representative test algorithms in terms of test sequence, computational complexity, fault coverage, and design methodology. Furthermore, the evolution of MBIST test algorithm development is discussed, ranging from conventional ad hoc approaches to enhanced algorithms derived from existing March tests. The comparative analysis indicates that an 18N-complexity test algorithm is generally required to achieve complete detection of all unlinked static faults in SRAM, whereas optimized 14N-complexity algorithms provide an effective trade-off between test time and fault coverage. Finally, the review identifies current research challenges, including efficient detection of dynamic and linked memory faults and the development of MBIST algorithms for emerging memory technologies such as magnetic random-access memory (MRAM), highlighting promising directions for future research.

Corresponding Author:

Aiman Zakwan Jidin
Fakulti Teknologi dan Kejuruteraan Elektronik dan Komputer, Universiti Teknikal Malaysia Melaka
76100 Durian Tunggal, Melaka, Malaysia
Email: aimanzakwan@utem.edu.my

1. INTRODUCTION

Memory built-in self-test (BIST) or MBIST is commonly adopted by the semiconductor industry to perform on-chip memory testing [1]. It reduces reliance on expensive external testers and, thus, produces a lower cost and faster memory test, owing to its capability to execute tests and then verify the test responses independently [2]–[5]. It is a vital technique since the chips are mainly occupied by memories nowadays, consuming up to 94% of the total chip area [6], [7]. Subsequently, the quality of the on-chip memories plays a vital role in modern chips' manufacturing yield, which may contribute up to 90% of overall faults in the chip [8]–[10]. A test using MBIST is conducted by performing a series of test operations called the test sequences, defined by the applied test algorithm [11]. It typically consists of a controller to generate the test

patterns to be written to the memory under test and the expected results, which are then compared to the output read from the memory to detect any failure [12]–[14]. Hence, the built-in MBIST circuitry provides faster memory testing than using external testers as it eliminates their test latency [4], [15], [16]. Different test algorithms consist of dissimilar test operation sequences, offering different fault coverage and test complexity.

The fault coverage determines the number of faulty behaviors that may occur in memory, such as static random-access memory (SRAM) during a test, which subsequently defines the quality of the test. Meanwhile, the complexity of a test indicates its length, which directly influences its cost: the higher the complexity, the longer the test length, and the higher the test cost [17], [18]. The complexity of a test algorithm can be either linear or non-linear [11], [19]. It can be expressed as mNp, where m is the number of test operations in the sequence, and N is the size of the memory under test (usually defined in the number of word cells). A test algorithm is said to have a non-linear complexity when p is greater than 1. Two examples of non-linear test algorithms are the Galloping Pattern (GALPAT) and the Walking Pattern (WALPAT) algorithms, with a complexity of 4N2 and 2N2, respectively [11], [12], [20]. On the other hand, a test algorithm's complexity is considered linear when p equals 1. A linear-complexity test algorithm typically produces a significantly shorter test time than a non-linear complexity test algorithm. As shown in Table 1, a test using a linear-complexity algorithm on a 1 MB memory requires only 102.4 ms. In contrast, a test on the same memory may require up to 1 day to be completed when a non-linear complexity algorithm is used [8], [11], [21].

Table 1. The test completion time on a 1-MB memory for various test complexities

Test complexity		
N	$N^{3/2}$	N
102.4 ms	1.8 minutes	1.3 day

For this reason, the industry prefers a linear-complexity test algorithm like the March-series test algorithms [19], [22]. A March test algorithm consists of a certain number of test operations that must be applied on every cell of the memory under test. Its test sequence combines the following four possible operations: write the cell to 0 (w0), write the cell to 1 (w1), read from the cell and expecting a 0 (r0), and read from the cell and expecting a 1 (r1) [11]. Additionally, these test operations may be carried out either in the ascending (⇑) or descending (⇓) memory address order. In the former case, the test operations will first be carried out on the cell with the base address [23]–[25]. A March algorithm is generally broken down into multiple smaller groups called the test elements; all the test operations that belong to a test element must be executed to all memory cells first before moving on to the following test elements.

For instance, the March X algorithm [26] has the following test sequence: ⇳(w0); ⇑(r0, w1); ⇓(r1, w0); ⇳(r0). In this notation, the test elements are separated by semicolons. So, it has 4 test elements, starting from TE_0 to TE_3. In TE_0 (⇳(w0)), all cells will be set to 0 regardless of the memory address direction. Next, in TE_1 (⇑(r0, w1)), all cells (starting from the cell with the base memory address) will be read (expecting a 0 at the output) and set to 1. After that, in TE_2 (⇓(r1, w0)), all cells will be read (expecting a 1 at the output) and set to 1, starting from the cell with the final memory address. Finally, all cells will be read (expecting a 0 at the output) in any address direction in TE_3 (⇳(r0)). These test operations are illustrated in Table 2 using a simple 3-cell memory.

Table 2. An illustration of the March X algorithm test operations

Test element	Test sequence	Illustration
TE_0	⇳(w0)	ascending: ? ? ? → w0 ? ? → 0 w0 ? → 0 0 w0 descending: ? ? w0 → ? w0 0 → w0 0 0
TE_1	⇑(r0, w1)	0 0 0 → r0, w1 0 0 → 1 r0, w1 0 → 1 1 r0, w1
TE_2	⇓(r1, w0)	1 1 1 → 1 1 r1, w0 → 1 r1, w0 0 → r1, w0 0 0
TE_3	⇳(r0)	ascending: 0 0 0 → r0 0 0 → 0 r0 0 → 0 0 r0 descending: 0 0 r0 → 0 r0 0 → r0 0 0

Selecting an appropriate test algorithm for an MBIST controller is one of the most critical design decisions because it directly affects both test quality and production cost. Test algorithms with high complexity (e.g., exceeding 18N) generally achieve excellent fault coverage but require longer execution time, thereby increasing manufacturing cost. Conversely, low-complexity algorithms significantly reduce test time but may fail to detect certain memory fault models, resulting in inadequate test quality [27], [28]. Consequently, an effective MBIST test algorithm must achieve a suitable balance between fault coverage and test complexity.

Over the past several decades, numerous March-based test algorithms have been proposed to improve fault coverage while minimizing test complexity. These algorithms differ in their test sequences, detectable fault models, implementation complexity, and suitability for MBIST architectures. Although several surveys have discussed memory testing, a comprehensive comparison focusing on the evolution of March algorithms, their complexity, fault coverage, and design methodologies remains limited.

Therefore, this paper presents a comprehensive review of March test algorithms for SRAM static fault detection in MBIST applications. First, the characteristics and detection requirements of common static SRAM fault models are discussed. Next, representative March algorithms are systematically compared in terms of their test sequences, computational complexity, and fault coverage. Subsequently, the methods used to develop and optimize these algorithms, including modifications of existing March tests and newly proposed approaches, are reviewed and analyzed. Finally, current research challenges and future research directions are highlighted to support the development of more efficient MBIST algorithms for emerging memory technologies.

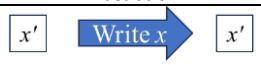
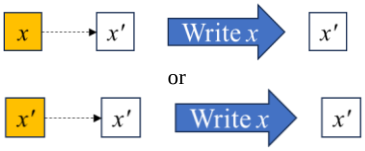
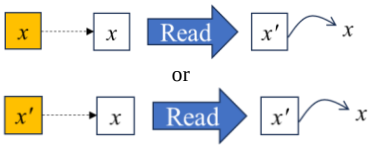
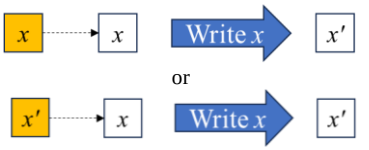
2. STATIC FAULT MODELS IN SRAM

Memory faults are generally classified into static faults and dynamic faults according to the number of test operations required to activate the faulty behavior. A static fault can be sensitized and detected using a single memory operation, whereas a dynamic fault requires a sequence of two or more memory operations for activation and observation [10], [29]. Because static faults constitute the majority of manufacturing-related defects and can be detected using relatively simple test sequences, they have been extensively investigated in the development of MBIST algorithms. Memory faults can also be categorized as unlinked or linked faults. An unlinked fault occurs independently, where the faulty behavior of one memory cell is unaffected by other faults present in the memory array. In contrast, a linked fault arises when two or more faults coexist and interact with one another. Such interactions may alter, mask, or even compensate for the observable behavior of individual faults, making fault detection considerably more challenging [30]. Consequently, test algorithms capable of detecting linked faults generally require more comprehensive test sequences than those designed solely for unlinked faults.

This paper considers 9 types of unlinked static faults, as described in Table 3. Single-cell faults (SCF), TF, RDF, IRF, DRDF, and WDF are SCF whose fault behaviors only affect a single victim cell. In contrast, CFtr, CFdrd, and CFwd are the Double-Cell faults (DCF), where the faulty behavior observed in a victim cell (v) depends on the state or the operations done in its aggressor cell (a) [2], [31]–[35]. In addition, each SCF consists of 2 fault primitives (FP), which indicates its faulty characteristics using the $<S / V / O>$ notation, where S shows the fault's sensitizing operations, V represents the observed faulty state at the v cell, and O the read value to be observed at the output [36]–[38]. O can have '-' as a value if it is irrelevant or is expected to be identical to V. Meanwhile, a DCF has 8 FPs since it depends on two possible states of the a cell (low or high state) and the location of the a cell can be either before ($a<v$) or after ($a>v$) the v cell within the memory. Therefore, the FP notation for a DCF is $<S_a;S_v / V / O>_{a>v}$ or $<S_a;S_v / V / O>_{a<v}$.

The FP of each fault also indicates its detection requirement, as in the necessary test operation sequence to sensitize and detect it. An SAF $<x/x'/->$ can be sensitized by setting all memory cells to x, where $x \in \{0, 1\}$. Then, its faulty behavior can be detected when a read operation from the affected v cell returns its complement x'. A TF $<x'$w$x/x'/->$ sensitization and detection is almost identical to SAF $<x/x'/->$, but all cells' states must be initialized to x' before performing the wx operation. RDF $<$r$x/x'/x'>$ and IRF $<$r$x/x/x'>$ can be sensitized and detected by reading from all memory cells that are set to x-state. They have identical detection requirements as the SAF $<x/x'/->$ [34]. Meanwhile, a DRDF $<$r$x/x'/x>$ can be detected by performing the read operation twice consecutively on every memory cell; the first read will sensitize its faulty behavior while the second will detect it at the output. Finally, a WDF $<x$w$x/x'/->$ can be sensitized by writing an x logic to memory cells containing x; a read operation that follows can detect its faulty behavior.

Table 3. An SRAM's unlinked static fault models' descriptions

Fault	FP	SCF/DCF	Description	Illustration
Stuck-at fault (SAF)	<0/1/->, <1/0/->	SCF	Regardless of the new written value, a *v* cell's state remains low or high.	
Transition fault (TF)	<0w1/0/->, <1w0/1/->	SCF	A *v* cell fails to change a state (from low to high or high to low).	
Read Destructive fault (RDF)	<r0/1/1>, <r1/0/0>	SCF	A read from a *v* cell unwantedly changes its state and returns incorrect output.	
Incorrect read fault (IRF)	<r0/0/1>, <r1/1/0>	SCF	A read from a *v* cell returns incorrect output.	
Deceptive read destructive fault (DRDF)	<r0/1/0>, <r1/0/1>	SCF	A read from a *v* cell unwantedly changes its state but returns the correct output.	
Write disturb fault (WDF)	<0w0/1/->, <1w1/0/->	SCF	A non-transition write to a *v* cell unexpectedly changes its state.	
Transition coupling fault (CFtr)	<0;0w1/0/->_{a>v}, <0;0w1/0/->_{a<v}, <1;0w1/0/->_{a>v}, <1;0w1/0/->_{a<v}, <0;1w0/1/->_{a>v}, <0;1w0/1/->_{a<v}, <1;1w0/1/->_{a>v}, <1;1w0/1/->_{a<v}	DCF	A *v* cell fails to change a state (from low to high or high to low) every time its *a* cell is in a specific state.	
Deceptive read destructive coupling fault (CFdrd)	<0;r0/1/0>_{a>v}, <0;r0/1/0>_{a<v}, <1;r0/1/0>_{a>v}, <1;r0/1/0>_{a<v}, <0;r1/0/1>_{a>v}, <0;r1/0/1>_{a<v}, <1;r1/0/1>_{a>v}, <1;r1/0/1>_{a<v}	DCF	A read from a *v* cell unwantedly changes its state but returns the correct output every time its *a* cell is in a specific state.	
Write disturb coupling fault (CFwd)	<0;0w0/1/->_{a>v}, <0;0w0/1/->_{a<v}, <1;0w0/1/->_{a>v}, <1;0w0/1/->_{a<v}, <0;1w1/0/->_{a>v}, <0;1w1/0/->_{a<v}, <1;1w1/0/->_{a>v}, <1;1w1/0/->_{a<v}	DCF	A non-transition write to a *v* cell unexpectedly changes its state every time its *a* cell is in a specific state.	

In contrast, since a DCF involves two coupling cells, their detection requirements are more complex than SCFs. CFtr, CFdrd, and CFwd occurrences can be sensitized and detected by following the same rules as TF, DRDF, and WDF, respectively. However, the identical test sequences must be carried out in both address directions to cover both $a>v$ and $a<v$ cases. Table 4 summarizes the required test sequences to sensitize and detect these faults [27], [34]. In this table, the necessary test operations to detect RDF and IRF are represented by SAF since their detection requirements are similar. $F(x)$ represents any operation(s) to cause the memory cells to be in the x-state.

Consider the following test sequence: ⇳(w0); ⇑(r0, w1, w1); ⇓(r1, w0, w0); ⇳(r0). This test sequence shall allow the MBIST operation to detect the following faults:

- SAF, RDF, and IRF since it includes necessary the read operations (r0 in TE_1 and TE_3, and r1 in TE_2).
- WDF since it performs non-transition write operations (w1w1 in TE_1 and w0w0 in TE_2), which are immediately followed by a read operation.
- Half of CFtr (CFtr <0;0w1/0/->$_{a>v}$, CFtr <1;0w1/0/->$_{a<v}$, CFtr <0;1w0/1/->$_{a>v}$, and CFtr <1;1w0/1/->$_{a>v}$).
- Half of CFwd (CFwd <0;1w1/0/->$_{a>v}$, CFwd <1;1w1/0/->$_{a<v}$, CFwd <0;0w0/1/->$_{a>v}$, and CFwd <1;0w0/1/->$_{a<v}$).
- It cannot detect DRDF and CFdrd since it has no consecutive double-read operations.

Table 4. Necessary test operation sequences to detect each unlinked static fault

Fault	Description
SAF	⇳(...,w*x*, r*x*, …); or ⇳(...,w*x*); ⇳(r*x*, …);
TF	⇳(*F*(*x'*), w*x*, r*x*, …); or ⇳(*F*(*x'*), w*x*); ⇳(r*x*, …);
DRDF	⇳(*F*(*x*), r*x*, r*x*, …); or ⇳(*F*(*x*), r*x*); ⇳(r*x*, …); or ⇳(..., *F*(*x*)); ⇳(r*x*, r*x*, …);
WDF	⇳(..., *F*(*x*), w*x*, r*x*, …); or ⇳(..., *F*(*x*), w*x*); ⇳(r*x*, …); or ⇳(..., *F*(*x*)); ⇳(w*x*, r*x*, …); or ⇳(..., *F*(*x*)); ⇳(w*x*); ⇳(r*x*, …);
CFtr	To detect CFtr <*x*;w*x'*/*x*/->$_{a>v}$: ⇳(*F*(*x*)); ⇑(*F*(*x*), w*x'*, r*x'*, *F*(*x'*)); or ⇳(*F*(*x*)); ⇑(*F*(*x*), w*x'*); ⇳(r*x'*, …); or ⇳(*F*(*x*)); ⇳(w*x'*, r*x'*, *F*(*x'*), *F*(*x*)); Required test sequences for CFtr <*x*;w*x'*/*x*/->$_{a<v}$ detection are obtained by reversing the address orders in the test sequence. To detect CFtr <*x'*;w*x'*/*x*/->$_{a>v}$: ⇳(*F*(*x*); ⇓(*F*(*x*), w*x'*, r*x'*, *F*(*x'*)); or ⇳(*F*(*x*); ⇓(*F*(*x*), w*x'*); ⇳(r*x'*, …); or ⇳(*F*(*x'*)); ⇳(*F*(*x*), w*x'*, r*x'*, *F*(*x'*)); or ⇳(*F*(*x'*)); ⇳(*F*(*x*), w*x'*); ⇳(r*x'*, …); Required test sequences for CFtr <*x'*;w*x'*/*x*/->$_{a<v}$ detection are obtained by reversing the address orders in the test sequence.
CFdrd	To detect CFdrd <*x*;r*x*/*x'*/*x*>$_{a>v}$: ⇳(*F*(*x*); ⇑(*F*(*x*), r*x*, r*x*, *F*(*x'*)); or ⇳(*F*(*x*); ⇑(*F*(*x*), r*x*, r*x*, *F*(*x*)); or ⇳(*F*(*x'*); ⇓(*F*(*x*), r*x*, r*x*, *F*(*x*)); or ⇳(*F*(*x'*); ⇓(*F*(*x*), r*x*); ⇳(r*x*, …); or ⇳(*F*(*x*); ⇳(*F*(*x*), r*x*, r*x*, *F*(*x'*)); or ⇳(*F*(*x*); ⇳(*F*(*x*), r*x*); ⇳(r*x*, …); or ⇳(*F*(*x*)); ⇳(r*x*, r*x*, *F*(*x'*), *F*(*x*)); or ⇳(*F*(*x*)); ⇳(*F*(*x'*), w*x*, r*x*, r*x*, *F*(*x*)); or ⇳(*F*(*x*)); ⇳(*F*(*x'*), w*x*, r*x*); ⇳(r*x*, …); Required test sequences for CFdrd <*x*;r*x*/*x'*/*x*>$_{a<v}$ detection are obtained by reversing the address orders in the test sequence. To detect CFdrd <*x'*;r*x*/*x'*/*x*>$_{a>v}$: ⇳(*F*(*x*); ⇓(*F*(*x*), r*x*, r*x*, *F*(*x'*)); or ⇳(*F*(*x'*); ⇑(*F*(*x*), r*x*, r*x*, *F*(*x*)); or ⇳(*F*(*x'*); ⇑(*F*(*x*), r*x*); ⇳(r*x*, …); or ⇳(*F*(*x'*); ⇳(*F*(*x*), r*x*, r*x*, *F*(*x'*)); Required test sequences for CFdrd <*x'*;r*x*/*x'*/*x*>$_{a<v}$ detection are obtained by reversing the address orders in the test sequence.
CFwd	To detect CFwd <*x*;w*x*/*x'*/->$_{a>v}$: ⇳(*F*(*x*)); ⇑(*F*(*x*), w*x*, r*x*, *F*(*x'*)); or ⇳(*F*(*x'*)); ⇑(*F*(*x*), w*x*, r*x*, *F*(*x*)); or ⇳(*F*(*x'*)); ⇑(*F*(*x*), w*x*); ⇳(r*x*, …); or ⇳(*F*(*x*)); ⇳(w*x*, r*x*, *F*(*x'*), *F*(*x*)); or ⇳(*F*(*x*)); ⇳(*F*(*x'*), w*x*, r*x*, *F*(*x*)); or ⇳(*F*(*x*)); ⇳(*F*(*x'*), w*x*); ⇳(r*x*, …); or ⇳(*F*(*x*)); ⇳(w*x*, r*x*, *F*(*x*)); or ⇳(*F*(*x*)); ⇳(w*x*); ⇳(r*x*, …); Required test sequences for CFwd <*x*;w*x*/*x'*/->$_{a<v}$ detection are obtained by reversing the address orders in the test sequence. To detect CFwd <*x'*;w*x*/*x'*/->$_{a>v}$: ⇳(*F*(*x*)); ⇓(*F*(*x*), w*x*, r*x*, *F*(*x'*)); or ⇳(*F*(*x'*)); ⇓(*F*(*x*), w*x*, r*x*, *F*(*x*)); or ⇳(*F*(*x'*)); ⇓(*F*(*x*), w*x*); ⇳(r*x*, …); or ⇳(*F*(*x'*)); ⇓(*F*(*x*), w*x*, r*x*, *F*(*x'*)); Required test sequences for CFwd <*x'*;w*x*/*x'*/->$_{a<v}$ detection are obtained by reversing the address orders in the test sequence.

3. REVIEW OF MBIST TEST ALGORITHMS

As mentioned in section 1, a test algorithm is indispensable in the MBIST implementation to define the test operation sequence to be executed onto the memory under test. Therefore, it influences two main criteria in a memory test: fault coverage and test complexity. A fault coverage (FC) defines the test quality in terms of the number of possible faults to be detected during a test. It can be calculated by dividing the number of detectable faults by the total faults, as written in (1). In this section, the coverage of each fault is calculated in the following manners: the FC of each SCF is calculated by dividing the number of detectable FPs over 2 (since each SCF consists of 2 FPs), while the FC of each DCF is obtained by dividing the number of detectable FPs over 8. Meanwhile, the test complexity determines the test duration; the higher the complexity, the longer it takes for a test to complete. A linear-complexity test algorithm's complexity equals the total number of test operations within its test sequence multiplied by the size of the memory or *N*. For example, the complexity of the March X algorithm [26] equals 6*N* since it consists of 6 read or write operations in its test sequence: ⇳(w0); ⇑(r0, w1); ⇓(r1, w0); ⇳(r0).

$$FC = \frac{no\ of\ detectable\ faults}{total\ faults} \quad (1)$$

Based on the literature, the classical zero-one is the algorithm with the lowest complexity (4*N*), with the ⇓(w0); ⇑(r0); ⇑(w1); ⇓(r1) test sequence. Despite its low complexity, it has the poorest fault coverage among all existing test algorithms, where it can only detect SAF, RDF, IRF, half of TF, and 25% of CFtr. Meanwhile, the Checkerboard algorithm offers minor improvement in TF and some CFs detections.

Figure 1 illustrates the data background patterns used by the Zero-One and Checkerboard algorithms. The Zero-One algorithm uses a solid data background, as shown in Figure 1(a). Meanwhile, the Checkerboard algorithm uses a checkerboard data background, where logic 0s and 1s are alternated between consecutive cells, as shown in Figure 1(b).

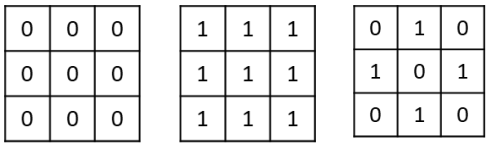


Figure 1. Data background patterns used in the Zero-One and Checkerboard algorithms: (a) solid data background and (b) checkerboard data background

The March X algorithm [26] and the MATS++ algorithm (test sequence: ⇳(w0); ⇑(r0, w1); ⇓(r1, w0, r0)) [39] have 6*N* complexity. Unlike the Zero-One and Checkerboard algorithms, both of these algorithms provide full coverage of TF and 50% coverage of CFtr [30]. The March C algorithm, with 11*N* complexity, has the following test sequence: ⇳(w0); ⇑(r0, w1); ⇑(r1, w0); ⇳(r0); ⇓(r0, w1); ⇓(r1, w0); ⇳(r0) [40]. Later, a read operation within the test sequence was considered unnecessary and hence removed to create a new March C- algorithm with 10*N* complexity and the following test sequence: ⇳(w0); ⇑(r0, w1); ⇑(r1, w0); ⇓(r0, w1); ⇓(r1, w0); ⇳(r0) [41]. For years, both the March C and March C- algorithms have proven to provide complete coverage of SAF, TF, RDF, IRF, CFtr, and conventional CFs such as the Static CF (CFst), Idempotent CF (Cfid), and Inversion CF (Cfin). However, due to technology scaling towards Very-Deep Submicron and nanometer processes, newer faults like DRDF, WDF, CFdrd, and CFwd have emerged and become more relevant in modern memories [42]. Consequently, these test algorithms are no longer adequate in memory testing to detect these new faults [43].

Many March test algorithms with complexities higher than 10*N* were created and can detect some of these faults, either fully or partially. The March CL algorithm, with 12*N* complexity, has the following test sequence: ⇳(w0); ⇑(r0, w1); ⇳(r1); ⇑(r1, w0); ⇓(r0, w1); ⇳(r1); ⇓(r1, w0); ⇳(r0) [44]. It can fully detect CFtr and partially detect DRDF and CFdrd, but it cannot be used to detect WDF and CFwd. Meanwhile, the March LR algorithm, with 14*N* complexity, was initially proposed to cover several linked and unlinked static faults [45]. It has the following test sequence: ⇳(w0); ⇓(r0, w1); ⇑(r1, w0, r0, w1); ⇑(r1, w0); ⇑(r0, w1, r1, w0); ⇑(r0). A researcher claims it fully covers all unlinked static SCFs and DCFs [15]. However, it is unable to detect these new faults (WDF, DRDF, CFwd, and CFdrd); its test sequence includes neither double read operations to enable DRDF and CFdrd detections nor the non-transition operations to detect WDF and CFwd [46]. Several 14*N*-complexity test algorithms exist and are preferred by the industry to balance fault coverage and test complexity [33]. The March SR algorithm, with ⇳(w0); ⇑(r0, w1, r1, w0); ⇑(r0, r0); ⇑(w1); ⇓(r1, w0, r0, w1); ⇓(r1, r1) test sequence, provides complete coverage of DRDF but only half of CFdrd [47]. A work was carried out in [33] to modify the data background bits used in the March SR test sequence to establish the modified March SR algorithm with the following test sequence: ⇳(w0); ⇑(r0, w0, r0, w1); ⇑(r1, r1); ⇑(w1); ⇓(r1, w0, r0, w0); ⇓(r0, r0). The newly modified test sequence completely detects all SCF, 50% CFtr and CFwd, and 62.5% coverage of CFdrd. Meanwhile, the March C+ algorithm was an improvement from the March C- algorithm by adding 4 read operations into the latter's test sequence to become ⇳(w0); ⇑(r0, w1, r1); ⇑(r1, w0, r0); ⇓(r0, w1, r1); ⇓(r1, w0, r0); ⇳(r0) [48]. Compared to the latter, the former offers complete detection of DRDF and CFdrd. Similar to the March SR algorithm, it does not provide any detection of WDF and CFwd. The March AZ2 algorithm, with 14*N* complexity, has the following test sequence: ⇳(w0); ⇓(w0, r0); ⇑(r0, w1, w1, r1); ⇑(r1, w0); ⇓(r0, w1, w1, r1); ⇑(r1). It balances test fault coverage and complexity by covering 100% of all SCFs and 75% of all DCFs [27]. Meanwhile, the March AZ1 algorithm, with 13*N* complexity, is a lower complexity alternative to the March AZ2 algorithm, offering a slightly lower CFtr coverage (62.5%) compared to the latter [27]. It has the following test sequence: ⇳(w0); ⇓(w1); ⇑(w1, r1, r1, w0); ⇑(w0, r0); ⇑(r0, w1, w1, r1); ⇑(r1).

The March-sift algorithm is a 17*N*-complexity test algorithm that is composed by ⇳(w0); ⇑(r0, w1); ⇓(r1, w0, r0); ⇑(r0, w1); ⇑(r1, w0); ⇓(r0, w0, r0); ⇑(r0, w1, r1); ⇳(r0) test sequence [31]. Despite its test complexity that is higher than a 14*N*-complexity like the March AZ2 algorithm, it offers a lower overall fault coverage since it can only detect half of the WDF, 5 out of 8 FPs of CFdrd (63%), and 2 out of 8 FPs of CFwd (25%). Through observation, its test sequence includes a non-transition write operation on logic 0 (0w0) at its TE_5. Consequently, it is unable to detect WDF <1w1/0/->, CFwd <X;1w1/0/->$_{a>v}$, and CFwd

<X;1w1/0/->$_{a<v}$ (where X represents any logic value). Meanwhile, an 18*N*-complexity test algorithm called the March-ee was established with the following test sequence: ⇑(w0); ⇑(r0, w1, r1); ⇑(r1, w0, r0); ⇑(r0, w1); ⇓(r1, w0, r0); ⇑(r0, w0); ⇓(r0, w1, r1); ⇑(r1) [49]. It provides a better overall fault coverage than the March-sift algorithm, but it can only detect 50% of the WDF and 25% of the CFwd due to the same weakness found in the latter. Another 18*N*-complexity algorithm called the March LV, with ⇕(w0); ⇑(r0, w1, w1, r1); ⇑(r1, w0, w0, r0); ⇓(r0, r0, w1, r1); ⇓(r1, r1, w0, r0); ⇓(r0) test sequence, provides better fault coverage through a full detection of WDF and 50% detection of CFwd [32].

The March MSS algorithm, also with 18*N* complexity, has been proven to include a minimal test sequence to completely cover all unlinked static faults in SRAM: ⇕(w0); ⇑(r0, r0, w1, w1); ⇑(r1, r1, w0, w0); ⇓(r0, r0, w1, w1); ⇓(r1, r1, w0, w0); ⇕(r0) [34]. It certainly provides faster memory testing than other test algorithms like:

- the March CS, a 20*N*-complexity algorithm with ⇕(w0); ⇑(w0, r0, w1, r1); ⇕(w1); ⇑(w1, r1, w0, r0); ⇓(w0, r0, w1, r1); ⇓(w1, r1, w0, r0); ⇕(w0, r0) test sequence [50],
- the March SS, a 22*N*-complexity algorithm with ⇕(w0); ⇑(r0, r0, w0, r0, w1); ⇑(r1, r1, w1, r1, w0); ⇓(r0, r0, w0, r0, w1); ⇓(r1, r1, w1, r1, w0); ⇕(w0) test sequence [51],
- the improved March C+, a 22*N*-complexity algorithm with ⇕(w0); ⇑(w0, r0, w1, w1, r1); ⇑(w1, r1, w0, w0, r0); ⇓(r0, w0, w1, w1, r1); ⇓(r1, w1, w0, w0, r0); ⇕(r0) test sequence [52],
- the improved March LR, a 22*N*-complexity algorithm with ⇕(w0); ⇓(r0, w1); ⇑(r1, r1, w0, w0, r0, r0, w1, w1); ⇑(r1, w0); ⇑(r0, r0, w1, w1, r1, r1, w0, w0); ⇑(r0) test sequence [46], and
- the March RAW, a 26*N*-complexity algorithm with ⇕(w0); ⇑(r0, w0, r0, r0, w1, r1); ⇑(r1, w1, r1, r1, w0, r0); ⇓(r0, w0, r0, r0, w1, r1); ⇓(r1, w1, r1, r1, w0, r0); ⇕(r0) test sequence [53].

A reduced version of the March RAW algorithm, called the March RAW1 algorithm, is also available with the following test sequence: ⇕(w0); ⇕(w0, r0); ⇕(r0); ⇕(w1, r1); ⇕(r1); ⇕(w1, r1); ⇕(r1); ⇕(w0, r0); ⇕(r0) [53]. With only 13*N* complexity, it only focuses on detecting all SCFs.

Table 5 summarizes all test algorithms reviewed in this section: F designates that a fault is 100% or fully covered, H specifies a 50% coverage, while '-' represents the non-detectability of a fault. This table shows that a minimum of 18N complexity is required for a test algorithm to fully cover all unlinked static faults. In contrast, it also demonstrates that test algorithms with complexities below 10*N* have poor fault coverages and cannot detect those new faults introduced by the nanometer process technology. In between, some test algorithms provide good coverage in many faults but are notably unable to detect WDF and CFwd. For example, the March C+ algorithm offers excellent detections of all faults except WDF and CFwd. Meanwhile, the March AZ2 algorithm, with 14*N* complexity, delivers the best coverage of all targeted faults among all below 18*N*-complexity test algorithms (with 83.3% of the overall fault coverage), thus offering the best balance between the memory testing's complexity and fault coverage.

Table 5. MBIST test algorithms' complexities and fault coverages

Test algorithm	Complexity	Fault coverage									
		SAF	TF	RDF	IRF	DRDF	WDF	CFtr	CFdrd	CFwd	Total
Zero-One	4*N*	F	H	F	F	-	-	25%	-	-	25%
MATS++	6*N*	F	F	F	F	-	-	H	-	-	33%
March X	6*N*	F	F	F	F	-	-	H	-	-	33%
March C-	10*N*	F	F	F	F	-	-	F	-	-	44%
March CL	12*N*	F	F	F	F	H	-	F	H	-	58%
March AZ1	13*N*	F	F	F	F	F	F	63%	75%	75%	81%
March LR	14*N*	F	F	F	F	-	-	F	-	-	44%
March SR	14*N*	F	F	F	F	F	-	F	H	-	61%
Modified March SR	14*N*	F	F	F	F	F	F	H	63%	H	69%
March C+	14*N*	F	F	F	F	F	-	F	F	-	72%
March AZ2	14*N*	F	F	F	F	F	F	75%	75%	75%	83%
March-sift	17*N*	F	F	F	F	F	H	75%	63%	25%	67%
March-ee	18*N*	F	F	F	F	F	H	F	F	25%	81%
March LV	18*N*	F	F	F	F	F	F	F	F	H	89%
March MSS	18*N*	F	F	F	F	F	F	F	F	F	100%
March CS	20*N*	F	F	F	F	F	F	F	F	F	100%
March SS	22*N*	F	F	F	F	F	F	F	F	F	100%
Improved March C+	22*N*	F	F	F	F	F	F	F	F	F	100%
Improved March LR	22*N*	F	F	F	F	F	F	F	F	F	100%
March RAW	26*N*	F	F	F	F	F	F	F	F	F	100%

4. REVIEW OF TEST ALGORITHM GENERATION METHODS

Based on the literature review, various ways can be utilized to create a new test algorithm. In memory testing's early days, classical test algorithms like the Zero-One, Checkerboard, GALPAT, and WALPAT were considered the ad-hoc tests since no formal fault models were established at that period [42]. As mentioned earlier, Zero-One and Checkerboard algorithms produce short memory testing with very poor fault coverages, whereas GALPAT and WALPAT algorithms offer higher fault coverage but require an extensive test duration due to their non-linear complexity, which causes the test cost to arise as well.

Many March test algorithms, e.g., the March C and MATS++, were created based on the test algorithm generation by simulation (TAGS) technique, owing to the establishment of memory fault models in the 1980s [42]. This technique creates multiple templates containing all possible combinations of read and write operations at various stages; it starts at the first stage with a 1*N*-complexity template containing a write operation. After that, the complexities of the templates will gradually increase at each stage. Each template's fault coverage is determined at each stage through analyses using a fault simulator like the RAM simulator for error screening (RAMSES). Additionally, all analyzed templates that do not show any fault coverage improvement from the previous stage are removed from further analysis. The process is repeated until a template containing the test sequence providing 100% coverage of the targeted faults is produced or the maximum test length is reached [11]. The TAGS process flow is depicted in Figure 2.

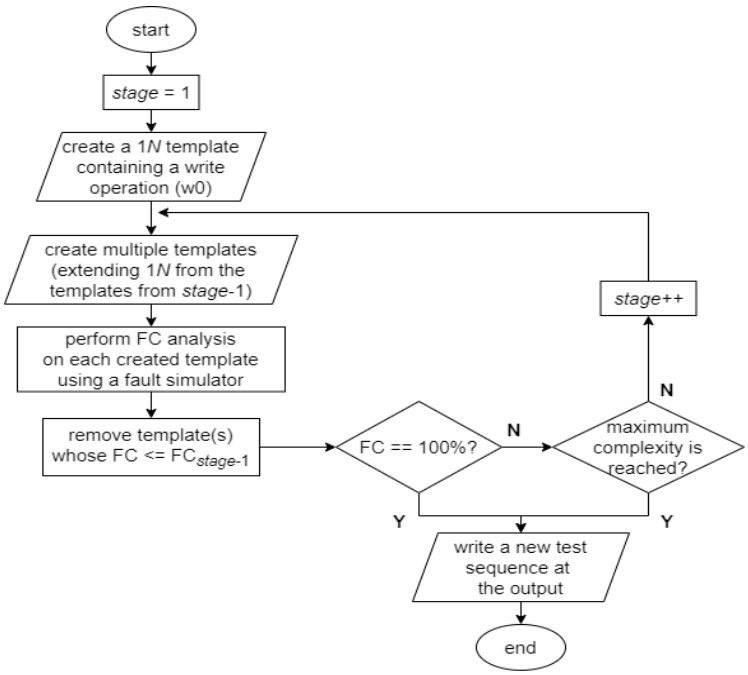


Figure 2. A new March test algorithm generation flow using TAGS

Another way to create a new test algorithm is the FP-based generation method, which was successfully used to establish the March SS and the March MSS algorithms [34], [42], [51]. In this method, all required test operation sequences to detect intended faults are identified based on their FPs. From here, a new test sequence is developed to provide optimum detection of these faults. For example, six propositions were established when creating the March MSS algorithm to guarantee its complete detection of all unlinked static faults in SRAM. For instance, to ensure its coverage of DRDF and CFdrd, its test sequence must include at least two (r0, r0) sequences and two (r1, r1) sequences, one of each must have an address order that is the opposite of the other: it must contain at least one ⇑(…, r*x*, r*x*, …) and one ⇓(…, r*x*, r*x*, …). An automation tool was used to develop 49 new test sequences containing 18 read or write operations, among which the March MSS algorithm test sequence was recognized to provide the best fault coverage, detecting all unlinked static faults in SRAM [34].

Furthermore, many test algorithms were generated from improving another existing test algorithm. On one hand, many of these algorithms were created by extending their initial test sequences through additional test operations. For example, the March C+ algorithm resulted from adding four read operations into the March C- algorithms, as illustrated in Figure 3, thus enabling the DRDF and CFdrd detections [48]. Meanwhile, the improved March LR was created by adding 8 new test operations into the March LR

algorithm's test sequence, resulting in the complete detection of DRDF, CFdrd, WDF, and CFwd, which are undetectable by the March LR algorithm [46]. Despite enhancing the fault coverage, this method has a drawback: it undoubtedly increases the test complexity since extra test operations are involved in detecting more faults.

On the other hand, a work proposed in [33] optimizes the data background used in the March SR algorithm's test sequence while keeping its complexity unchanged, as shown in Figure 4. Compared to the March SR's test sequence, the modified March SR's test sequence includes non-transition write operations in its TE_1 and TE_3 to enable the detections of WDF and CFwd, which are undetectable by the former.

However, it has several limitations: it focuses only on enabling SCF detection and can enable WDF and CFwd detections only when the input test sequence includes double-read operations [54]. Therefore, the work done in [27] improved it by adding several more steps to rearrange the test operations and optimizing the test addresses to enable and remove redundancies in DCF detections. It resulted in the March AZ2 algorithm, with 14*N* complexity, derived by modifying the existing March LR algorithm's test sequence, as depicted in Figure 5. After adjusting the latter's data background, the address orders are optimized by reversing its TE_4's address direction (from ⇑ to ⇓) to eliminate duplicated test element and DCF detection redundancies. Finally, several test operations are rearranged to enable WDF and CFwd detections. The March AZ2 algorithm delivers the best coverage of the targeted faults among all test algorithms with a complexity lower than 18*N*.

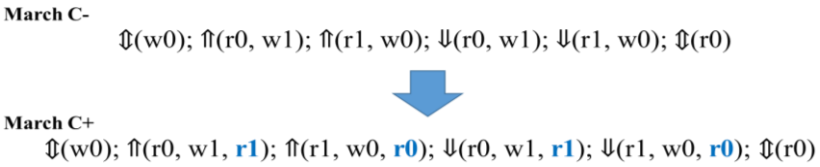


Figure 3. The March C+ creation by adding four read operations into the March C- algorithm's test sequence

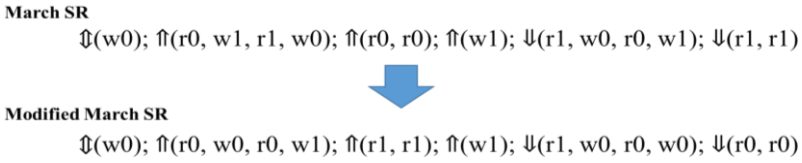


Figure 4. The modified March SR algorithm that is derived from the March SR algorithm's data background optimization

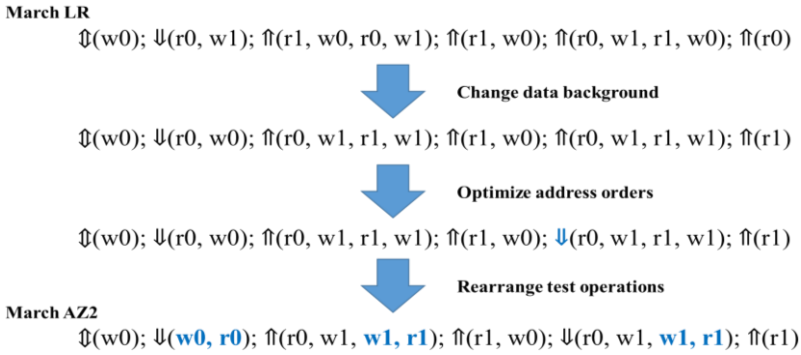


Figure 5. The creation of the March AZ2 algorithm from the modification of the March LR algorithm

Despite the significant progress in balancing test complexity and static fault coverage, considerable opportunities remain to improve the efficiency and capability of memory testing. Existing March algorithms, such as March AZ2 (14N) and March MSS (18N), provide excellent coverage of unlinked static faults but are generally insufficient for detecting more complex fault types, including linked static faults and dynamic faults. According to the literature, comprehensive dynamic fault detection typically requires substantially higher test complexity, as demonstrated by algorithms such as March SL24 (24N) [55], March FRDD (38N)

[2], and March LSD (75N) [55]. Consequently, future research should focus on developing low-complexity test algorithms capable of detecting dynamic faults while maintaining high fault coverage and short test execution time. Promising directions include optimizing existing March algorithms through redundancy reduction, operation reordering, or hybrid test strategies, as demonstrated in recent studies [56], [57].

In addition to improving SRAM testing, further research is needed to establish fault models and MBIST algorithms for emerging memory technologies. Advanced non-volatile memories, such as Magnetoresistive RAM (MRAM), resistive RAM (RRAM), phase-change memory (PCM), and Ferroelectric RAM (FeRAM), exhibit operating principles and failure mechanisms that differ fundamentally from those of conventional SRAM. For example, MRAM stores information using magnetic tunnel junctions instead of charge, resulting in unique fault behaviors that cannot be adequately represented by conventional SRAM fault models. As a result, the development of comprehensive fault models, efficient test algorithms, and dedicated MBIST architectures for these emerging memories remains an important research direction. Furthermore, integrating adaptive testing techniques, machine learning, or artificial intelligence into MBIST design may provide additional opportunities to optimize test generation, improve fault diagnosis, and reduce overall testing cost for future high-density memory systems.

5. CONCLUSION

This paper presented a comprehensive review of MBIST algorithms for detecting unlinked static faults in SRAM. The review examined the characteristics and detection requirements of major static fault models and analyzed representative March test algorithms in terms of their test sequences, computational complexity, and fault coverage. The comparison demonstrates that the effectiveness of an MBIST implementation is largely determined by the selected test algorithm, which directly influences test quality, execution time, and manufacturing cost. The reviewed literature indicates that low-complexity March algorithms (10N or below) provide fast testing but are unable to detect several important static fault models. Conversely, complete detection of all unlinked static faults generally requires a minimum test complexity of 18N, as achieved by algorithms such as March MSS. To bridge the gap between testing efficiency and fault coverage, numerous optimized March algorithms with complexities ranging from 10N to 18N have been proposed. Among these, the March AZ2 algorithm provides one of the most effective trade-offs by achieving 14N complexity while detecting all state coupling faults (SCFs) and approximately 75% of dynamic coupling faults (DCFs), making it an attractive candidate for practical MBIST implementations.

Although significant progress has been achieved in SRAM testing, several research challenges remain. Future studies should focus on developing low-complexity algorithms capable of detecting both static and dynamic fault models without substantially increasing test time. In addition, the emergence of advanced memory technologies, including MRAM, RRAM, PCM, and FeRAM, necessitates the development of new fault models and corresponding MBIST algorithms tailored to their unique failure mechanisms. Furthermore, integrating intelligent optimization techniques and adaptive test strategies into MBIST architectures represents a promising direction for achieving higher fault coverage, shorter test duration, and improved testing efficiency in next-generation memory systems.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the financial support provided by the Ministry of Higher Education Malaysia (MOHE) through the Fundamental Research Grant Scheme (FRGS) under Grant No. FRGS/1/2024/TK07/UTEM/02/17. The authors also express their sincere appreciation to Universiti Teknikal Malaysia Melaka (UTeM) and Universiti Malaysia Perlis (UniMAP) for their continuous support and facilities provided throughout this research.

REFERENCES

[1] A. A. Wojciechowski, K. Marcinek, and W. A. Pleskacz, "Configurable MBIST processor for embedded memories testing," in *2019 MIXDES - 26th International Conference "Mixed Design of Integrated Circuits and Systems,"* IEEE, Jun. 2019, pp. 341–344, doi: 10.23919/MIXDES.2019.8787161.

[2] R. Wang, Z. Huang, G. Cai, and Z. Yu, "A built-in self-test circuit based on March FRDD algorithm for FinFET memory," in *Advances in Transdisciplinary Engineering*, vol. 35, 2023, pp. 437–447, doi: 10.3233/ATDE230069.

[3] T. S. Nguan Kong *et al.*, "An efficient March (5n) FSM-based memory built-in self test (MBIST) architecture," in *2021 IEEE Regional Symposium on Micro and Nanoelectronics (RSM)*, IEEE, Aug. 2021, pp. 76–79, doi: 10.1109/RSM52397.2021.9511602.

[4] S. M. U. Qutaiba Khasawneh, "Reducing the production cost of semiconductor chips using (parallel and concurrent) testing and real-time monitoring," Southern Methodist University, 2019. [Online]. Available: https://scholar.smu.edu/engineering_electrical_etds/32

[5] S. N. Bagewadi, S. Shadab, and J. Roopa, "Fast BIST mechanism for faster validation of memory array," in *2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, IEEE, May 2019, pp. 61–65, doi: 10.1109/RTEICT46194.2019.9016882.
[6] I. Mrozek, *Multi-run Memory Tests for Pattern Sensitive Faults*. Cham: Springer International Publishing, 2019, doi: 10.1007/978-3-319-91204-2.
[7] O. S. Nisha and K. Sivasankar, "Architecture for an efficient MBIST using modified March-y algorithms to achieve optimized communication delay and computational speed," *International Journal of Pervasive Computing and Communications*, vol. 17, no. 1, pp. 135–147, Feb. 2021, doi: 10.1108/IJPCC-05-2020-0032.
[8] S. Kapavarpu and S. Cheedarla, "An efficient and low power sram testing using clock gating," *International Research Journal of Engineering and Technology*, vol. 6, no. 7, pp. 1322–1326, 2019.
[9] R. Zeli, R. Silveira, and Q. Qureshi, "SoC memory test optimization using NXP MTR solutions," *LATS 2019 - 20th IEEE Latin American Test Symposium*, 2019, doi: 10.1109/LATW.2019.8704566.
[10] G. Renuka, "FPGA implementation of memory bists using single interface," in *AIP Conference Proceedings*, 2022, p. 030031, doi: 10.1063/5.0083691.
[11] L. T. Wang, C. W. Wu, and X. Wen, "Preface," in *VLSI Test Principles and Architectures*, Elsevier, 2006, pp. xxi–xxiii, doi: 10.1016/B978-012370597-6/50000-7.
[12] D. M. Pavani and Y. Prasad, "Design of programmable memory BIST using stochastic sobol sequence pattern generator," in *AIP Conference Proceedings*, 2023, p. 030019, doi: 10.1063/5.0125204.
[13] N. I. Singh and P. V. Joshi, "Performance analysis of BIST algorithms," *Indian Journal of Science and Technology*, vol. 12, no. 36, pp. 1–3, Sep. 2019, doi: 10.17485/ijst/2019/v12i36/147752.
[14] K. L. V. R. Kumari, M. A. Rani, and N. Balaji, "FPGA implementation of memory design and testing," in *2017 IEEE 7th International Advance Computing Conference (IACC)*, IEEE, Jan. 2017, pp. 552–555, doi: 10.1109/IACC.2017.0119.
[15] R. Manasa, R. Verma, and D. Koppad, "Implementation of BIST technology using March-LR algorithm," in *2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, IEEE, May 2019, pp. 1208–1212, doi: 10.1109/RTEICT46194.2019.9016784.
[16] Y. Lu, "BIST implementation access through a reconfigurable network," Lund University, 2019.
[17] A. A. A. Wahab, S. S. N. Alhady, W. A. F. W. Othman, H. Husin, and N. Q. M. Adnan, "Optimizing RAM testing method for test time saving using automatic test equipment," in *Lecture Notes in Electrical Engineering*, vol. 829 LNEE, 2022, pp. 260–266, doi: 10.1007/978-981-16-8129-5_41.
[18] 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)*, IEEE, Feb. 2020, pp. 1–5, doi: 10.1109/SCEECS48394.2020.43.
[19] V. S. Chakravarthi, "A practical approach to VLSI system on chip (SoC) design: a comprehensive guide," *A Practical Approach to VLSI System on Chip (SoC) Design: A Comprehensive Guide*, pp. 1–312, 2019, doi: 10.1007/978-3-030-23049-4.
[20] Y. Bai *et al.*, "Optimization of memory March X test algorithm based on nuclear safety level DCS system platform," in *Environmental Science and Engineering*, 2023, pp. 73–82, doi: 10.1007/978-3-031-30233-6_7.
[21] A. A. A. Wahab, S. S. N. Alhady, W. A. F. W. Othman, H. Husin, and N. Q. M. Adnan, "Optimizing RAM testing method for test time saving using automatic test equipment," *Lecture Notes in Electrical Engineering*, vol. 829 LNEE, pp. 260–266, 2022, doi: 10.1007/978-981-16-8129-5_41.
[22] I. Mrozek, N. A. Shevchenko, and V. N. Yarmolik, "Universal address sequence generator for memory built-in self-test," *Fundamenta Informaticae*, vol. 188, no. 1, pp. 41–61, 2022, doi: 10.3233/FI-222141.
[23] G. P. Acharya, M. A. Rani, G. G. Kumar, and L. Poluboyina, "Adaptation of March-SS algorithm to word-oriented memory built-in self-test and repair," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 26, no. 1, pp. 96–104, 2022, doi: 10.11591/ijeecs.v26.i1.pp96-104.
[24] Khushi and K. Singh, "Performance Analysis of March M & B algorithms for memory built-in self-test (BIST)," in *2022 IEEE World Conference on Applied Intelligence and Computing (AIC)*, IEEE, Jun. 2022, pp. 78–84, doi: 10.1109/AIC55036.2022.9848869.
[25] A. J. van de Goor, S. Hamdioui, and H. Kukner, "Generic, orthogonal and low-cost March Element based memory BIST," in *2011 IEEE International Test Conference*, IEEE, Sep. 2011, pp. 1–10, doi: 10.1109/TEST.2011.6139148.
[26] S. M. Al-Harbi and S. K. Gupta, "A methodology for transforming memory tests for in-system testing of direct mapped cache tags," in *Proceedings. 16th IEEE VLSI Test Symposium (Cat. No.98TB100231)*, IEEE Comput. Soc, 1998, pp. 394–400, doi: 10.1109/VTEST.1998.670897.
[27] A. Z. Jidin *et al.*, "Generation of new low-complexity March algorithms for optimum faults detection in SRAM," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 8, pp. 2738–2751, Aug. 2023, doi: 10.1109/TCAD.2022.3229281.
[28] A. Paul and P. Rony Antony, "Optimized microcode BIST architecture for multiple memory cores in SoCs," in *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, IEEE, May 2018, pp. 910–914, doi: 10.1109/RTEICT42901.2018.9012276.
[29] Y. Xiao, S. Lu, E. Wang, R. Zhu, and Z. Dai, "Test primitive: a straightforward method to decouple March," *arXiv preprint arXiv:2309.03214*, 2023.
[30] R. D. Adams, *High Performance Memory Testing: Design Principles, Fault Modeling and Self-Test*, vol. 22A. in Frontiers in Electronic Testing, vol. 22A. Boston: Kluwer Academic Publishers, 2003, doi: 10.1007/b101876.
[31] S. Alnatheer and M. A. Ahmed, "Optimal method for test and repair memories using redundancy mechanism for SoC," *Micromachines*, vol. 12, no. 7, p. 811, Jul. 2021, doi: 10.3390/mi12070811.
[32] Z. Cai, Y. Wang, S. Liu, K. Lv, and Z. Wang, "A novel BIST algorithm for low-voltage SRAM," in *2019 IEEE International Test Conference in Asia (ITC-Asia)*, IEEE, Sep. 2019, pp. 133–138, doi: 10.1109/ITC-Asia.2019.00036.
[33] N. A. Zakaria, W. Z. W. Hasan, I. A. Halin, R. M. Sidek, and X. Wen, "Fault detection with optimum march test algorithm," in *2012 Third International Conference on Intelligent Systems Modelling and Simulation*, IEEE, Feb. 2012, pp. 700–704, doi: 10.1109/ISMS.2012.88.
[34] G. Harutunyan, V. A. Vardanian, and Y. Zorian, "Minimal March tests for unlinked static faults in random access memories," in *23rd IEEE VLSI Test Symposium (VTS'05)*, IEEE Comput. Soc, 2005, pp. 53–59, doi: 10.1109/VTS.2005.56.
[35] Siemens, "Tessent MemoryBIST User's Manual For Use with Tessent Shell," 2020.
[36] Z. Cai, H. Yu, H. Yang, Z. Wang, and Y. Guo, "Design of novel dynamic March algorithm based on memory built-in self-test," *Journal of Electronics and Information Technology*, vol. 45, no. 9, pp. 3420–3429, 2023, doi: 10.11999/JEIT221032.

[37] G. Cardoso Medeiros, M. Fieback, L. Wu, M. Taouil, L. M. Bolzani Poehls, and S. Hamdioui, "Hard-to-Detect Fault Analysis in FinFET SRAMs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 6, pp. 1271–1284, Jun. 2021, doi: 10.1109/TVLSI.2021.3071940.
[38] G. Harutyunyan, S. Shoukourian, and Y. Zorian, "Fault awareness for memory BIST architecture shaped by multidimensional prediction mechanism," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 3, pp. 562–575, Mar. 2019, doi: 10.1109/TCAD.2018.2818688.
[39] Jin-Fu Li and Cheng-Wen Wu, "Memory fault diagnosis by syndrome compression," in *Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001*, IEEE Comput. Soc, 2001, pp. 97–101, doi: 10.1109/DATE.2001.915007.
[40] M. Marinescu, "Simple and efficient algorithms for functional ram testing.," *Digest of Papers - International Test Conference*, pp. 236–239, 1982.
[41] A. J. Van De Goor, "Using march tests to test SRAMs," *IEEE Design & Test of Computers*, vol. 10, no. 1, pp. 8–14, Mar. 1993, doi: 10.1109/54.199799.
[42] S. Hamdioui, "Testing embedded memories: a survey," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7721 LNCS, 2013, pp. 32–42, doi: 10.1007/978-3-642-36046-6_4.
[43] N. A. Zakaria, "Multiple and solid data background scheme for testing static single cell faults on SRAM memories," Universiti Putra Malaysia, 2013.
[44] V. A. Vardanian and Y. Zorian, "A March-based fault location algorithm for static random access memories," in *Proceedings of the Eighth IEEE International On-Line Testing Workshop (IOLTW 2002)*, IEEE, 2002, pp. 256–261, doi: 10.1109/OLT.2002.1030228.
[45] A. J. van de Goor, G. N. Gaydadjiev, V. G. Mikitjuk, and V. N. Yarmolik, "March LR: a test for realistic linked faults," in *Proceedings of 14th VLSI Test Symposium*, IEEE Comput. Soc. Press, 1996, pp. 272–280, doi: 10.1109/VTEST.1996.510868.
[46] X. Ning, H. Yang, M. Zhang, Y. Wang, Y. Zhao, and S. Qiao, "Multi-type SRAM test structure with an improved March LR algorithm," in *2022 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, IEEE, Nov. 2022, pp. 578–582, doi: 10.1109/APCCAS55924.2022.10090328.
[47] S. Hamdioui and A. J. Van De Goor, "An experimental analysis of spot defects in SRAMs: realistic fault models and tests," in *Proceedings of the Ninth Asian Test Symposium*, IEEE Comput. Soc, 2000, pp. 131–138, doi: 10.1109/ATS.2000.893615.
[48] Z. Zhi-chao, H. Li-gang, and W. Wu-chen, "SRAM BIST design based on March C+ Algorithm," *Modern Electronics Technique*, vol. 34, no. 10, pp. 149–151, 2011.
[49] M. A. Ahmed and A. M. Abuagoub, "MBIST controller based on March-ee algorithm," *Journal of Circuits, Systems and Computers*, vol. 30, no. 09, p. 2150160, Jul. 2021, doi: 10.1142/S0218126621501607.
[50] J. Zeng, Z. Wang, J. Yuan, W. Peng, and Y. Zeng, "An improved SRAM fault built-in-self-test algorithm," *Hunan Daxue Xuebao/Journal of Hunan University Natural Sciences*, vol. 46, no. 4, pp. 97–101, 2019, doi: 10.16339/j.cnki.hdxbzkb.2019.04.014.
[51] S. Hamdioui, A. J. Van De Goor, and M. Rodgers, "March SS: a test for all static simple RAM faults," *Records of the IEEE International Workshop on Memory Technology, Design and Testing*, vol. 2002-January, pp. 95–100, 2002, doi: 10.1109/MTDT.2002.1029769.
[52] Y. Wang, Q. Zheng, and Y. Yuan, "The improvement of March C+ algorithm for embedded memory test," in *Communications in Computer and Information Science*, vol. 592, 2016, pp. 31–37, doi: 10.1007/978-3-662-49283-3_4.
[53] S. Hamdioui, Z. Al-Ars, and A. J. van de Goor, "Testing static and dynamic faults in random access memories," in *Proceedings 20th IEEE VLSI Test Symposium (VTS 2002)*, IEEE Comput. Soc, 2002, pp. 395–400, doi: 10.1109/VTS.2002.1011170.
[54] A. Z. Jidin, R. Hussin, M. S. Mispan, and L. W. Fook, "Novel March test algorithm optimization strategy for improving unlinked faults detection," in *2021 IEEE Asia Pacific Conference on Circuit and Systems (APCCAS)*, IEEE, Nov. 2021, pp. 117–120, doi: 10.1109/APCCAS51387.2021.9687791.
[55] G. Harutyunyan, S. Shoukourian, V. Vardanian, and Y. Zorian, "A new method for march test algorithm generation and its application for fault detection in RAMs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 6, pp. 941–949, 2012, doi: 10.1109/TCAD.2012.2184107.
[56] L. W. Ying, R. Hussin, N. Ahmad, L. W. Fook, and A. Z. Jidin, "Modified March MSS for unlinked dynamic faults detection," *2022 IEEE 20th Student Conference on Research and Development, SCOReD 2022*, pp. 68–72, 2022, doi: 10.1109/SCOReD57082.2022.9974097.
[57] W. Y. Loh, W. F. Lee, R. Hussin, A. Zakwan Jidin, N. Ahmad, and N. A. Zakaria, "Novel March WY approach for dynamic fault detection in memory BIST," *Proceedings - 2023 16th IEEE International Symposium on Embedded Multicore/Many-Core Systems-on-Chip, MCSoC 2023*, pp. 516–521, 2023, doi: 10.1109/MCSoC60832.2023.00082.

BIOGRAPHIES OF AUTHORS

Aiman Zakwan Jidin recently completed his Ph.D. in Electronic Engineering at Universiti Malaysia Perlis, Malaysia. His research topic focuses on creating a new low-complexity memory testing algorithm for optimum static fault coverage in SRAM. Previously, he obtained his M.Eng. in Electronic and Microelectronic Systems from ESIEE Engineering Paris, France, in 2011, before working as an FPGA IP Core Design Engineer at Altera Corporation Malaysia (now part of Intel). He is a full-time lecturer and researcher in Electronic and Computer Engineering at Universiti Teknikal Malaysia Melaka (UTeM). His research interests include DFT, VLSI, and FPGA system design. He can be contacted at email: aimanzakwan@utem.edu.my.

Razaidi Hussin received a Ph.D., degree in Electronic and Electrical Engineering from the University of Glasgow, the UK, in 2017 with a focus on oxide-reliability issues in complementary metal-oxide-semiconductor nanoscale devices. He joined Universiti Malaysia Perlis (previously known as KUKUM) in 2002. He is a full-time senior lecturer at the Faculty of Electronic Engineering Technology, Universiti Malaysia Perlis. He can be contacted at email: shidee@unimap.edu.my.

Mohd Syafiq Mispan received B.Eng. Electrical (Electronics) and M.Eng. Electrical (Computer and Microelectronic System) from Universiti Teknologi Malaysia, Malaysia, in 2007 and 2010 respectively. He had experience working in the semiconductor industry from 2007 until 2014 before pursuing his Ph.D. He obtained his Ph.D. in Electronics and Electrical Engineering from the University of Southampton, the United Kingdom, in 2018. He is currently a senior lecturer in the Faculty of Electronic and Computer Engineering and Technology, Universiti Teknikal Malaysia Melaka. His research interests include hardware security, CMOS reliability, VLSI design, and electronic systems design. He can be contacted at email: syafiq.mispan@utem.edu.my.

Lee Weng Fook is a technical director at Emerald Systems Design Center with 26 years of IC design experience. Lee has vast experience in designing with Verilog and VHDL, RTL coding, and logic synthesis for ASIC/FPGA/SOC. Lee specializes in synthesizing and tweaking synthesis for performance and low power, leading to enhanced methodology to address advanced DFT techniques for VDSM technology, development, and deployment of low-power standard cell libraries. Lee has led the development of new architectures and micro-architectures for efficient PMSM motion control ASIC and has developed architectures for AI classification algorithms implementation in ASIC. Lee has published 4 IC design books and is also the inventor and co-inventor of 14 design patents granted by the US patent and trademark office. He can be contacted at email: seanlee@emersysdesign.com.

Loh Wan Ying received the B.Eng., degree (Hons.) in electronics from Universiti Tunku Abdul Rahman, Petaling Jaya, Malaysia. She is currently pursuing the Ph.D., degree with Universiti Malaysia Perlis, Arau, Malaysia. She has a few years of experience in designing with Verilog and VHDL, RTL ASIC/IP coding, and FPGA synthesis. She has experience working with RISC V, AXI bus, and FPGA SoC. She has experience in artificial intelligence front-end design and full-chip verification. Her research interest focuses on improving fault detection efficiency by optimizing memory testing algorithms. She can be contacted at email: wyloh@studentmail.unimap.edu.my.