Journal of Harbin Engineering University
ISSN: 1006-7043

Vol 45 No. 6
June 2024

March_Select_17n: Novel BIST Algorithm and Architecture

M. Parvathi
BVRIT Hyderabad College of Engineering for Women, Hyderabad, India
parvathi.m@bvrithyderabad.edu.in

Abstract

Due to technology trends, variations in VLSI chip designs led to the test methods evolving from the existing ones to become highly efficient, commendable, and adaptable to the new test designs. Especially in embedded SoCs, built-in self-test (BIST) has traditionally become a necessary part in addition to the core logic that provides at-speed testing using a course of algorithms assigned for intended fault coverage. However, in recent test methods like March AB, which cover all static and dynamic faults with the help of larger test sequences due to increased core memory size, this led to increased testing time rather than functioning time. Aiming to reduce test time and test length complexity, a new March algorithm, March_Select_17n is proposed in this paper that works on the selection of essential March elements from the given set of algorithms, resulting in redundancy reduction. Initially, the March Select-46n algorithm is proposed to address the fault coverage issues. Further, by considering a few high fault coverage March algorithms through which each term from an individual algorithm will be eliminated on the basis of repetitive term after comparing it with the term that comes from previously applied algorithms in the pipeline. Hence forth, achieved the final optimized March_Select_17n sequence. For the implementation of this new March_Select_17n algorithm, a novel BIST architecture setup is proposed incorporating new modules like compare and filter. In addition, it aims to cover undefined faults too, which are not seen in any of the existing literature as of now. The simulation results observed with test speed improvement of 86% compared with other existing algorithms that overcome the test latency problem.

Keywords: March algorithms, undefined faults, compare and filter, primitive selection, test latency, ML regression &classification.

1. Introduction

In this digital world, embedded systems are playing a vital role. Every high-end embedded system suffers from hardware and software complexities. Hardware complexity in terms of insufficient memory capability, and software complexity in terms of mal function due to memory faults. Due to advancement in technologies to the level of sub-nano meter, the capacity of memory is also grown accordingly, but at the same time fault occurrences in memory also increased. SRAM in general the right choice as memory due to its speed in operation and reliability of the content. However, as the complexity of memory design increases, the chances of defects may cause violation of original functionality. The traditional memory testing follows simple automatic test procedures, and can be replaced with new chip if required. But, in the case of embedded memory, it needs special testing procedures, as it built around the core. In-built testing through test module is an essential tool usually used for embedded memory testing because of its ease of use, efficiency in speed testing, especially shorter test time, and cost effectiveness. BIST for memory testing is MBIST, which comprises memory as a "circuit under test (CUT) and a "test pattern generator (TPG) as essential components. The additional parts that make the testing complete are the comparator and controller. On the other hand, BIST can also be used for logic testing under the name LBIST, which uses random logic with additional components like a signature analyzer and a scan chain.

BIST testing can be done in two modes: online mode and offline mode. Online mode of testing involves the real-time running operation of the core logic, whereas offline mode of testing involves the test running for a specific duration during which the test circuit is away from its normal

operation. In either of the modes, the test involves a test pattern through which the expected output will be analyzed to decide whether the CUT is faulty or not. The test patterns or test vectors are the crucial deciding factors in identifying the fault within the CUT. Higher end test methods rely on automatic test pattern generators in order to improve the ease in the test process and accuracy as well. The selection of test patterns is based on the CUT under consideration. For example, if CUT is memory that affected with a stuck fault, corresponding algorithms that generates the necessary test elements are essential in elevating the underlying stuck fault; if the chosen set of patterns are deficient in identifying the defect, it is required either to discard the chosen algorithm or to incorporate another suitable algorithm or test pattern. Hence, while designing the BIST architecture itself, it is essential to choose appropriate algorithm according to the fault model under consideration. Today, the recent literature reveals that the existing memory models have become large in capacity and complex using various design technologies [1, 2, 3, 4, 5, 6] that were developed to address various issues such as static noise margin, low supply voltage, read and write stability, etc. In all these memories, if one parameter is violated during system operations, it leads to erroneous behavior that causes the system to be faulty. Hence technological based advanced test methods are always essential to ensure the correctness in the functionality of the system prior to the use. Testing embedded SRAMs is not a new concept, but how well the testing is done is important. The selected algorithm or test pattern should be efficient enough in detecting even undetectable, realistic faults in any of the circumstances within the system. Few such efficient methods are inductive fault analysis (IFA) [7], realistic fault model approach [8] in both the cases, the test algorithms deal with layout information that reframes the test cases and failure model mapping. While mapping the failure models with test cases, some of the fault models may perfectly map with defect models. But some may leave unknown leading to undetectable fault/s. This makes the test inadequate during the process variation under sub-micron memory models. The March primitives are an essential measure of the test model, selecting appropriate sensitizing input and reading the corresponding output from the cell. A reduced functional fault model is more suitable for applying these March-based algorithms [9]. Each March algorithm comprises March elements that indicate the operation to be performed on memory. Memory related operations, like read or write, will be used as March primitives in each algorithm. The only concern in these kinds of March algorithms is how well the number of faults can be covered within the limited number of applied March elements. One should limit the number of March elements to be applied to achieve less test latency while at the same time achieving high fault coverage.

Very early March algorithms focused on linked faults and evolved into March_A and March_B. But, these are limited in detecting all linked faults. March_LA and March_LR are used for a limited set of interconnected faults within the memory that have high fault coverage. However, one should compromise with the large testing complexity in order to meet the high fault coverage, which is noted as 43n for linked faults. However, it is ever essential due to the inadequacy of March algorithms in having large fault coverage, which led to the identification of newer algorithms with more complexity, such as March SL of 41n. March AB [10, 11, and 12] is in a similar line developed for detecting linked faults as March SL, as but with less complexity than 19n.

In literature, "Read after Write"-based March test algorithms are observed with test lengths of 13 n (RAW1) and 26 n (RAW2). These are used for the detection of dynamic faults in both single- and two-cell SRAMs. These algorithms elevated the importance of the read operation followed by the write operation. These kinds of fault models are also considered unrealistic. March AB1 and AB2 of lengths 11N and 22N, respectively [13], are used to address the problem of dynamic faults in single cell and two cell models. Further, to improve the fault coverage, the test complexity was raised up to 100n for the detection of dynamic faults within the two-cell architecture using two fault-sensitizing operations in terms of victim or aggressor cells. Though the test improved in fault coverage, yet the

test length is more that causes much delay while applying for large capacity memories.

The BIST controller is one of the most efficient online and offline test modes for built-in memory test methods. Several BIST architectures, like microcode-based [14, 15] and efficient algorithms for 4N–22N, are implemented [16]. Since the BIST is simple in its implementation methods, other algorithms have been developed with improved fault coverage. Memory testing using March C- and modified March C- is observed as popular test algorithms identified for covering faults like stuck at faults (SAF), transition faults (TF), and coupling faults (CF) [17, 18]. But they are limited in covering dynamic faults.

Dynamic faults is one of the categories considered in literature with its complex test scenario that needs more than two operations, like sensitization and observing for read and write effects. This dynamic nature of the fault is observed with sub-threshold SRAM architectures [19]. Dynamic faults for sub threshold region are represented in primitive concepts. The advantage of representing in primitive concept is the probability of assigning each operation from the functional fault model is made easy. Defect manifestation is taken as input for fault models where process variation is considered a major problem [20]. The behavior of SRAM under low voltage regimes is crucial and leads to an increase in the number of fault models compared to contemporary existing fault models [21, 22, 23]. In bigger SoC applications, spares such as supplementary row and column architectures are frequently used for testing and repair [24, 25, 26]. Instead of focusing on test stimuli performance, the application of large, complex test algorithms in such massive memory banks makes testing laborious and time-consuming.

Having gone through these many references and their work, the major problems commonly observed in all these fault models and corresponding algorithms are test latency, test complexity, hardware complexity, and fault coverage. It will be very difficult to achieve all these without being satisfied with one solution. Hence, the objectives of our work is to achieve an algorithm for testing of SRAM cell faults with less test latency and with less complexity. Hence, we have proposed a new March algorithm “March 46n”, but by using the filtering method, the complexity is further reduced to the “March_Select_17n” algorithm without satisfying the test performance.

The forgoing sections of the paper is organized in a way such that section 2 reveals the issues correlated to existing March algorithms and their test complexities. This also covers the details about required fault model and test procedure for detection of undetectable faults. Section 3 discusses the proposed March 46n, and March_Select_17n a new March algorithm to cover the undetectable faults and also selects the necessary primitives out of given algorithms to result less latency in test time. Finally, Section 4 deals with results and section 5 conclusions.

2. Related Review on March Algorithms, and Test Issues

2.1. Generic March Primitive Notation

March algorithms are very popular in applying as test procedure while detecting faults in SRAM as fault model. March algorithms uses primitives or preserved notations, such as basic read or write operation to observe the inference between outcome of predictable (fault-free) and the experimental (faulty) behavior within the memory. The general representation for preserved notation uses sensitizing operation S, fault observation F and expected outcome R and is represented as follows:

< Sensitize / Faulty output / Read result >

Similarly, in the case of multiple cell or two cell scenario, the primitive will be denoted as

< Sensitize aggressor/victim / Faulty output aggressor/victim / Read result >

Aggressor and victim cells are at the choice of selection in the multi-cell memory environment. Table 1 shows various March algorithms along with their primitive notations, and their fault coverage. Using this primitive notation one can explore huge number of primitive based faults and accordingly fault models. Using SRAM 6T cell as basic model, few fault models ad their primitive notations are shown in Table 1.

Table 1. Various Fault Models along with their primitive notations

S. No	Fault Model [13-23]	Circuit Defect Model	Definition	Primitive Notation		
				S	F	R
1.	State Fault	Short or bridge between any of the access transistors to bit lines.	Within the cell, the stored logic value gets flipped before accessing it prior to the sensitizing operation	1	0	-
				0	1	-
2.	Delay or transition fault	Open at the gate of any of the access transistors	Cell value will not change for any change in the input. Also, read fails due to open at the access transistor. Hence it fails in any transition.	0w1	0	-
				1w0	1	-
3.	Write destructive fault	Short that forms an invert operation between bit lines BL and internal cell node Q	Any attempt for write operation prior to the sensitizing causes cell value to flip (0 to 1 or 1 to 0)	0w0	↑	-
				1w1	↓	-
4.	Read Destructive Fault	Short that forms an invert operation between bit lines BLB and internal cell node QB	Any attempt for read operation causes cell value to flip (0 to 1 or 1 to 0) and hence results in incorrect read value on bit lines	r0	↑	1
				r1	↓	0
5.	Deceptive Read Destructive Fault	Short that forms an invert operation between two internal pmos transistors	Any attempt for read operation changes the cell logic and returns a random value.	r0	↑	0
				r1	↓	1
6.	Random Read Destructive Fault	Short that forms an invert operation between bit lines BLB and internal cell node QB	Any attempt for read operation performed on the cell returns random value while changing the logic value of the cell.	r0	↑	?
				r1	↓	?
7.	Incorrect Read Fault	Short between two bit lines leads to exchange of read data, and hence none of the correct value will be read from the bit lines.	Any attempt for read operation accomplished on the cell yields the inappropriate logic value while keeping the correct value in the cell.	r0	0	1
				r1	1	0
8.	Random Read Fault	Short between bit lines to true node and one more short between write logic line to vdd lead to random return value	Any attempt for read operation make the cell suffers from a random read fault that returns a random	r0	0	?
				r1	1	?

			value while keeping the correct value in the cell.			
9.	Undefined State Fault	Short between both supply lines leads to this undefined state fault.	Any attempt to read or write leads the cell undefined state before the cell accessed.	1	?	-
				0	?	-
10.	Undefined Write Fault	Nodes Q and QB shorted to result the undefined write fault	Any attempt on write operation in the cell leads to undefined state at nodes.	0w0/0 w1	?	-
				1w0/1 w1	?	-
11.	Undefined Read Fault	Nodes Q and QB shorted to result the undefined write fault	Any attempt on read of cell leads to return an undefined state from the cell	rx	?	0 / 1 /?
12.	Stuck-At Fault	Short between cell node to one of the supply lines or one of the bit lines	Any attempt on read or write results in cell stuck-at constant value of logic 1 or 0 leads to stuck at fault	∀	0	-
				∀	1	-
13.	No Access Fault	Short between WL to ground	None of the operation can be performed when the cell suffers from a no access fault	0w1	0	-
				1w0	1	-
				rx	x	?
14.	Data Retention Fault	Open defect that is equivalent to marginal resistance occurrence between load/source transistors to supply rails or between node to access transistors.	DRF persists while cell loses its earlier stored data when not been accessed for long time.	(0/1/x) |t1	↑/↓/ ?	-

2.2. Gaps Identified from the literature- Untestable faults

Simple static and dynamic faults are made detectable by various March algorithms [8, 10], but few from realistic fault models are left treated as undefined. These undefined faults are treated as untestable faults that are not detectable using existing March algorithms as the application is restricted to only functional fault models. The faults that easily escape from the test are falls in this category. When the functional fault model of 6T SRAM expanded with short defect model with all possible shorts between the nodes, few of the fault models resulting with different behavior which have no correlation with existing faults. These faults if not considered in the test algorithms, the test results would not complete in giving maximum fault coverage. Hence March algorithm primitive notations are supposed to expand or remodel in order to support even detect untestable faults too.

2.3. Materials and Methods to detect untestable faults in 6T SRAM Cell

Few of the untestable or undetectable faults are identified with the method of parasitic extraction through layout editor. The untestable faults are named according to their behavior and are Undefined Write Fault (UWF) with QB-VSS short, Undefined Read Fault (URF) with BL-WL or Q –VDD short, Bit Line Delay fault (BDF) with Q-BLB short, Initialization Order Fault (IOF) with QB-VDD short, and Unstabilized Write/Read Fault (USWF, USRF) with both QB and Q shorted, No Access Fault (NAF) with WL shorted to ground VSS, and Write before Access fault (WBA) with WL is shorted to VDD. These fault models are peculiar in behavior and none of the literature represented with any kind of March primitive. The identification of these untested faults is possible with parasitic extraction method in which identification RC at each node varies according to the fault level [18-23]. Fig.1. shows fault free 6T SRAM, Fig.2 shows the correspond layout. Each node reflects with its corresponding parasitic R, C and L values shown besides to the layout.

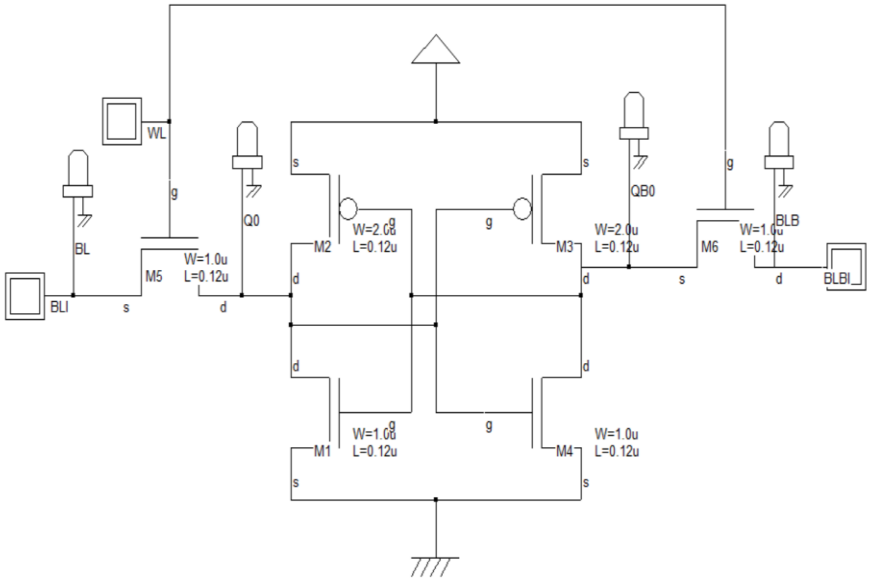


Fig. 1.Fault free SRAM

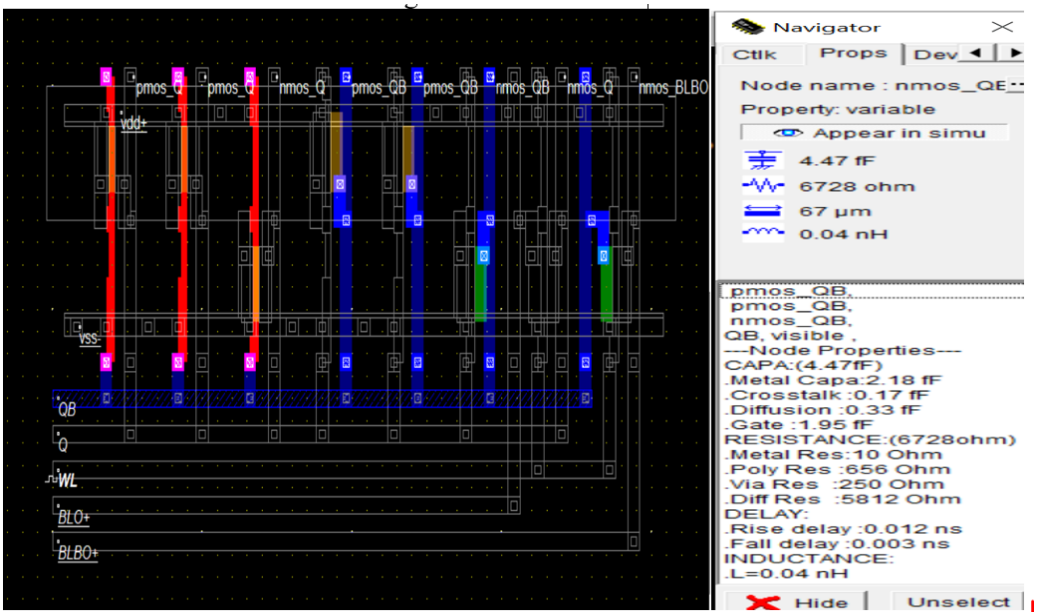


Fig.2. Layout of fault free SRAM and corresponding node parameters

Parasitic extraction method supports either short defect or open defect fault models, in which each node of SRAM is to be either shorted to possible adjacent nodes or to be opened. Each SRAM cell consists of 7 nodes including both bit lines (BL, BLB), word logic line (WL) for accessing the cell in read and write operations, and two internal node points Q and QB for identifying the written value in true and complementary form, and two supply lines VDD and VSS. From these seven nodes the possible shorts after eliminating duplication are (7x6)/2=21 [20]. From these 21 total fault models, each fault model parasitic R&C will be extracted from the corresponding layout, by comparing with fault free model of SRAM one can analyze the defect model of such faulty model.

The total capacitance extracted from any one of the layers in the layout will be the sum of capacitances like gate capacitance, diffusion capacitance, crosstalk capacitance, metal capacitance, and fringe capacitance.

$$C_{layer}=Cg+Cj+Cc+Cm+Cf \quad (1)$$

Similarly, the total resistance extracted from any one of the layers will be the sum of all the resistances like metal resistance, poly resistance, via resistance, and diffusion resistance

$$R_{wire}=Rm+Rpoly+Rvia+Rd \quad (2)$$

In this way the RCs of untestable faults are extracted. Table 2 shows the collective information of all untestable fault model's along with their extracted R and C parasitic values that helped to create fault model dictionary. The highlighted yellow values are the indicators of faults when compared with fault free cell. NA is referred as "Not Accessible" represented to a particular node that disappears while using short fault model. The key point to notice from the extracted RCs is, wherever the fault imposed that particular node only gets effected with raise in RC values, remaining are observed with same as RCs of fault free. Hence keeping the fault model dictionary as testing parameter, one can identify the type of fault occurred immediately by looking into the layout in the physical design phase. This kind of fault analysis may not come from March algorithms, however, the March primitive notations can be improved by incorporating the new primitives that supports in testing of these kind of untestable faults along with existing traditional faults.

Table 2. Fault Models observed from Parasitic Extraction Method

S. No	**Short Defect**	**Fault Model**	**Node Q**		**Node QB**		**Node WL**		**Node BL**		**Node BLB**	
			Fault Free C=4.66fF R=7185Ω		**Fault Free C=4.67 fF R=6728Ω**		**Fault Free C=1.98 fF R=369Ω**		**Fault Free C=1 fF R=1158Ω**		**Fault Free C=0.95 fF R=2094Ω**	
			C,fF	**R,Ω**	**C,fF**	**R,Ω**	**C,fF**	**R,Ω**	**C,fF**	**R,Ω**	**C,fF**	**R,Ω**
1	WL-BL	WBA	6.47	12919	4.67	6728	NA	NA	1.74	1341	0.95	2094
2	QB-VDD	IOF	4.49	7550	6.68	11858	1.98	369	1	1158	0.95	2094
3	Q-BLB	BDF	5.53	9860	4.67	6728	1.98	369	0.98	1158	NA	NA
4	QB-BL	USWF & USRF	4.5	7550	6.27	9954	1.99	369	NA	NA	0.95	2094

3. Proposed March Algorithm and Notation for Untestable Faults

In general March algorithms provide primitives for functional fault models. However to avoid undetectable fault escapism from test algorithms the essential primitives to be included.

3.1. March Primitive notation for Untestable Faults

According to [13-23] March primitive notations are hold good for functional fault models, with a non-empty set, yet the untestable faults that are newly found from parasitic extraction can easily escape from test under this scenario. Hence, using some relaxation to the primitive condition, new primitives are defined for identifying the undefined faults too. For these newly defined undefined fault models, the proposed primitive notation is shown in Table 3.

Table 3. New Fault Models from parasitic extraction method and their primitive notation

S.No	Short Defects	Fault Model	Definition	Operation required	March primitive notation
1	WL-BL or WL-VDD	WBA	Any attempt of write operation causes the cell logic value flips i.e 0 to 1 or vice versa	{(-,r1),(,r0)}	< - /↓↑/- >
2	WL-BLB or QB-VDD	IOF	An attempt of read or write operation order if changed, that causes incorrect write/read result.	{(w0,r0,w1,r1), (w1,r1,w0,r0),(w1,w0,r0),(w0,w1,r1)}	<0//0/1, 1/0/1, 1/0/1, 0/1/0>
3	WL-VSS or QB-BLB	BDF	An attempt to read or write operation takes longer time than usual.	{(w0,r0,w0,r1), (w1,r1,w1,r0), (w0,r1,w1,r1), (w1,r0,w0,r0)}	<0/?/0,1, 1/?/0,1>
4	(QB-BL)/(BL-BLB)	USWF	An attempt of write operation causes a shift in the cell value.	{(w0,r1,w1,r1), (w1,r0,w1,r1), (w0,r0,w1,r1), (w1,r1,w0,r0)}	<0/↑↓/1, 0/↓↑/0>
5	(QB-BL)/(BL-BLB)	USRF	An attempt of write operation causes a continuous shift in the cell or return value	{(w0,r1,r0,?), (w1,r0,r1,?)}	<0/?/0>, <1/?/1>

New March (46n) Algorithm for new fault models for detecting undefined faults (UDFs) along with few existing faults, as shown below:

{(-,r1), (-,r0); (w0,r0,w1,r1),
(w1,r1,w0,r0),(w1,w0,r0),(w0,w1,r1);
(w0,r0,w0,r1), (w1,r1,w1,r0), (w0,r1,w1,r1),
(w1,r0,w0,r0); (w1,r0,w1,r1), (w0,r1,w0,r0);
(w0,r1,r0,?), (w1,r0,r1,?)}

3.2. Working of Proposed March 46n Algorithm

Elements {(-,r1), (-,r0): These are used to test the initial condition of memory by read data. It uses random read '1' or read '0' as the cell stored data is unknown at the initial state. And also, it elevates the defect model of the cell such that the cell flicks its data before taking any initialization step. It leads to write before Access fault (WBA).

Elements (w0, r0, w1, r1) and (w1, r1, w0, r0): This operation allows identifying any SA1/0 fault in the cell

Elements (w1, w0, r0), (w0, w1, r1); This operation allows identifying any SAF and Transition fault TF fault in the cell.
Elements (w0, r0, w0, r1), (w1, r1, w1, r0): To detect SAF, TF, Initialization Order Fault (IOF), and coupling faults
Elements (w0, r1, w1, r1), (w1, r0, w0, r0); To detect SAF, TF and coupling faults.
Elements (w1, r0, w1, r1), (w0, r1, w0, r0): To detect SAF, TF, coupling faults, Bit-Line-Delay Fault (BDF), and dynamic faults
Elements (w0, r1, r0, ?), (w1, r0, r1, ?): To detect SAF, TF, coupling faults and dynamic faults and undefined faults such as Unstabilized Read/write Fault (USRF). The symbol '?' is used to leave the cell for neither read nor write operation, but to observe the change accordingly that occur after write or read operations applied.

3.3. Functionality of each March Element in the proposed Algorithm March_Select_17n
It is observed that the term ↕ (w0) from Mats+ will be considered only once and will be eliminated from all other algorithms, to avoid redundancy. Similarly each primitive term from each algorithm will be compared with all primitive terms of chosen algorithms, to remove the all redundant terms. The eliminated terms are as follows:

{↕ (w0); ↓(r0;w1;w0), ↑ (r0;w1; r1;w0; r0;w1);); ↑ (r1;w0;w1); (r1;w0;w1;w0); ↓(r0;w1;w0), ↑ (r0;w1); ↑ (r1;w0); ↓ (r0;w1); ↓ (r1;w0); ↕ (w0); ↕ (w0); ↓ (r0;w1); ↑(r1;w0; r0;w1); ↑ (r1;w0); ↑ (r0;w1; r1;w0); ↑(r0), {↓ (w0); ↑(r0;w1; r1;w0); ↑ (r0; r0); ↑ (w1); ↓ (r1;w0; r0;w1); ↕ (w0); ↕ (r0); ↕(r1);

Finally, the new array that comprises with irredundant March elements of 17n complexity is as follows:

{↕ (w0); ↑ (r0;w1); ↓(r1;w0), (w0;w1), (w1,w0), (r0), (r0,w0), (w1), (r1), (r1,w1),(w0,r1), (w1,r0),(w1,r1), (-,r1), (-,r0),(w0,r1,r0,?), (w1,r0,r1,?)}

New array of 17n is named as "March_Select_17n (proposed)" and corresponding fault detection capabilities is shown in Table 4, and each March element operation is explained as mentioned.

Table 4. March Elements and operation for detecting faults

Element count	March Elements in March_Select_17n(proposed)	Functionality	Result
1	(w0);	Write '0' by considering address sequence from bottom to top and vice versa	Sensitizing operation for detecting SA1
2	(r0;w1);	Initial w0 from step1followed by r0, then w1	SA1
3	(r1;w0)	Initial w1 from step2 followed by r1 and w0	SA0
4	(w0;w1);	Initial w0 from step3 followed w0, w1	SA1, TF
5	(w1;w0);	W1 from step4 followed by w0	SA0, TF
6,7	R0, (r0,w0)	W0 from step5 followed by r0, then w0/1/0	SA0, TF
8,9,10	(W1), (r1), (r1,w1)	Separate operations of w1, r1 and followed by r1,w1	SAF, TF, Linked Faults
11,12	(w0,r1), (w1,r0)	W0,r1,followed by w1,r0	TF, dynamic faults
13	(w1,r1)	R0 followed by w1,r1 and w0	SAF and TF

14,15	(-,r1), (-,r0);	Without initialization, but read operation	WBA fault
16	(w0,r1,r0,?),	W0, followed by r1,r0 and unexpected cell state	USWF
17	(w1,r0,r1,?)}	W1, followed by r0,r1 and unexpected read result and cell data	USRF

3.4. Algorithm Selection and Proposed less latent March Terms

In this section BIST architecture for the low latency test select is represented. The lists of algorithms under consideration are shown in Table 5, corresponding fault coverage's are shown in Table 6.

Table 5. List of March Algorithms considered to be used in the proposed BIST

S.No	Algorithms [8,9,10,11,12,13,14,40,41]	March Elements notation
1	MATS+	{↕ (w0); ↑ (r0;w1); ↓(r1;w0)}
2	March A	{↕(w0);↑(r0;w1;w0;w1);↑(r1;w0;w1);↓(r1;w0;w1;w0); ↓(r0;w1;w0)}
3	March B	{↕ (w0); ↑ (r0;w1; r1;w0; r0;w1); ↑ (r1;w0;w1); ↓ (r1;w0;w1;w0); ↓(r0;w1;w0)}
4	March C-	{↕ (w0); ↑ (r0;w1); ↑ (r1;w0); ↓ (r0;w1); ↓ (r1;w0); ↕ (r0)}
5	March M	{↕ (w0); ↑(r0;w1; r1;w0); ↕ (r0); ↑(r0;w1); ↕ (r1); ↑(r1;w0; r0;w1); ↕(r1); ↓ (r1;w0)}
6	March SS	{↕ (w0); ↑ (r0; r0;w0; r0;w1); ↑ (r1; r1;w1; r1;w0); ↓ (r0; r0;w0; r0;w1); ↓(r1; r1;w1; r1;w0); ↕ (r0)}
7	March LR	{↕ (w0); ↓ (r0;w1); ↑(r1;w0; r0;w1); ↑ (r1;w0); ↑ (r0;w1; r1;w0); ↑(r0)}
8	March SR	{↓ (w0); ↑(r0;w1; r1;w0); ↑ (r0; r0); ↑ (w1); ↓ (r1;w0; r0;w1); ↓(r1; r1)}
9	March AB	{↕ (w0); ↓ (r0;w1; r1;w1; r1); ↓ (r1;w0; r0;w0; r0); ↑(r0;w1; r1;w1; r1); ↑(r1;w0; r0;w0; r0); ↕ (r0)}
10	March_46n (proposed)	{↕(-,r1), (-,r0); ↑(w0,r0,w1,r1), (w1,r1,w0,r0),(w1,w0,r0),(w0,w1,r1); ↓(w0,r0,w0,r1), (w1,r1,w1,r0), (w0,r1,w1,r1), (w1,r0,w0,r0); ↑(w0,r1,w1,r1), (w1,r0,w1,r1); ↕(w0,r1,r0,?), (w1,r0,r1,?)}

Table 6. Fault coverage by March Algorithms under consideration

S.No	Algorithms [8,9,10,11,12,13,14,40,41]	Fault Coverage (Functional Fault Models)
1	MATS+	SAF, few TFs, IRF, RDF
2	March A	SAF, TF, CF, few Linked Faults
3	March B	SAF, TF, CF, few Linked Faults
4	March C-	SAF, TF, IRF, RDF, few CFs

5	March M	SAF, RDF, IRF, TF, DRDF
6	March SS	SAF, RDF, IRF, TF, WDF, DRDF
7	March LR	SAF, TF, CF, few Linked Faults
8	March SR	SAF, TF, CFs, IRF,RDF, DRDF, SOF
9	March AB	dRDF, dIRF, dDRDF, dCFds, dCFrd, dCFdrd, dCFir, static linked faults
10	March 46n (proposed)	SAF, TF, CFs, IRF,RDF, DRDF, SOF, dynamic DRDFs, Linked Faults, Undetectable Faults (WBAF, IOF, BDF, USWF, USRF)

The proposed work presented in this paper is an outcome of research development on new automated specific hardware which includes software to analyze and select the test sequences of given March algorithms that are used as the input or the base algorithm to generate new test sequences optimized for intended faults' detections while preserving the test complexity.
Pseudo code of algorithm selection of primitive elements is as follows:

```
var alg_a = [{↕ (w0); ↑ (r0;w1); ↓(r1;w0)}]; //Mats+ in array a
var alg_b = [{↕ (w0); ↑ (r0;w1;w0;w1); ↑ (r1;w0;w1); ↓(r1;w0;w1;w0); ↓(r0;w1;w0)}]; //March A in array b
var newArray = [];

            for(var i =0; i < alg_a.length; i++)
                        {
            for(var j =0; j < alg_b.length; j++)
            {
            if(alg_a[i] !=alg_b [j])
                        {
newArray.push(alg_b);
                        }
            If(alg_a[i] | alg_b[j] ==null]
            Return(0)
            }
  }
 //repeat the process by reload alg_a and alg_b with two more new March Algorithms to continue.
```

As shown in Fig.3, initially two arrays are considered with the names "alg_a" and "alg_b", in which each March algorithm is loaded. As the sequences of algorithms and sizes are not same, it should be compared with each March element from each algorithm, and finally un redundant terms will be copied into "newArray" (March_Select_17n (proposed)) that will be used for testing in the given address sequence. The same process will be repeated till all the algorithms under consideration are taken place

.

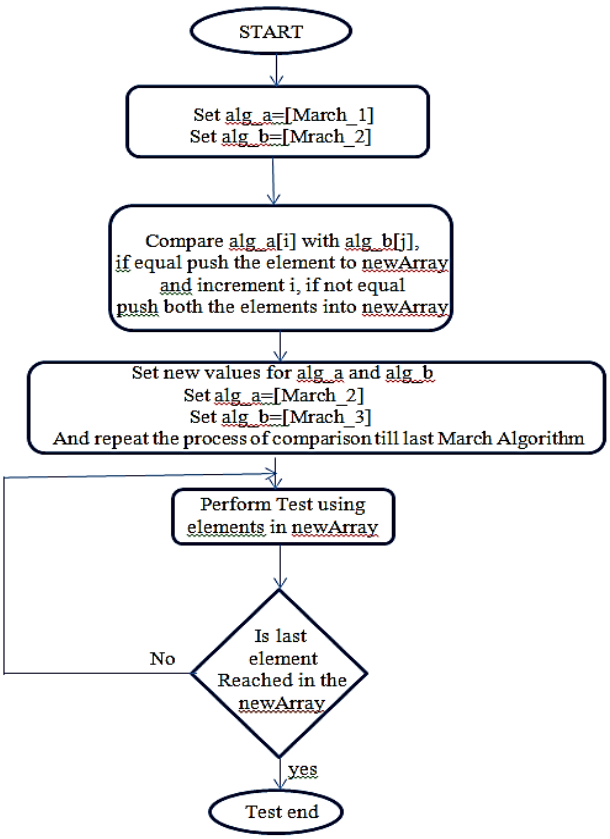


Fig.3. Flowchart for implementing low latent BIST

It consists initialization block in which two arrays will be assigned with two different March algorithms, comparison block in which the elements in each algorithm will be compared with the other array element, to find unique March element in the March_Select_17n (proposed). This process will be repeated till the last March algorithm under consideration. The final elements in the March_Select_17n (proposed) leads to collection of unique March elements that eliminates the problem of redundancy while applying the test as well as time delay in executing each algorithm. After the March_Select_17n (proposed) formed with unique march elements, the test will be performed as per the address order.

3.5. Experimental Setup for the Proposed less latent BIST Architecture:

BIST architecture needs the following major components:

i. A
ddress generator
ii. M
ode Selector for normal/test mode selection
iii. T
est generator
iv. C
ircuit under test
v. C
ontroller
vi. C
omparator

The proposed BIST architecture components are almost same as existing any general BIST model, the only difference occur in the selection of algorithm. The proposed BIST filters out the common terms from the chosen algorithm and uses March_Select_17n(proposed) with irredundant elements, further to reduce test latency. The proposed BIST controller is shown in Fig.4.

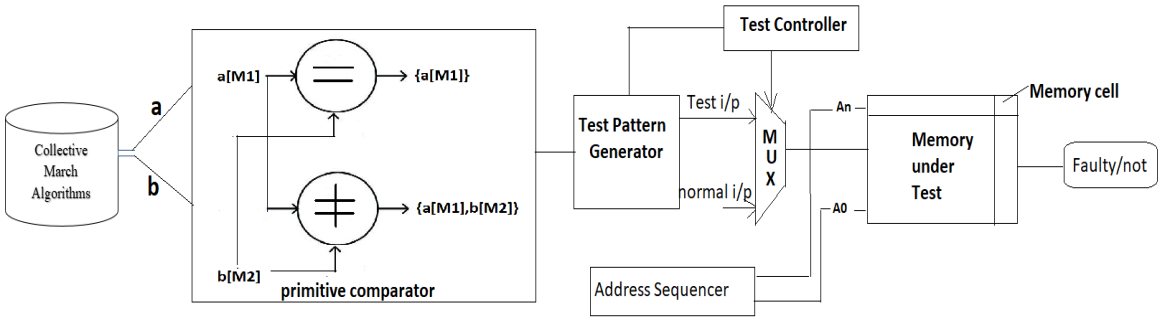


Fig.4. Proposed BIST Controller

Include Figures, Tables, and equations and numbers in the text wherever necessary and provide proper explanations. As shown in Fig.3, initially two arrays are considered with the names “alg_a” and “alg_b”, in which each March algorithm is loaded. As the sequences of algorithms and sizes are not same, it should be compared with each March element from each algorithm, and finally un redundant terms will be copied into “newArray” (March_Select_17n (proposed)) that will be used for testing in the given address sequence. The same process will be repeated till all the algorithms under consideration are taken place.

It consists initialization block in which two arrays will be assigned with two different march algorithms, comparison block in which the elements in each algorithm will be compared with the other array element, to find unique March element in the March_Select_17n(proposed). This process will be repeated till the last March algorithm under consideration. The final elements in the March_Select_17n (proposed) leads to collection of unique March elements that eliminates the problem of redundancy while applying the test as well as time delay in executing each algorithm. After the March_Select_17n (proposed) formed with unique march elements, the test will be performed as per the address order.

3.6. Block description and working of proposed BIST

The proposed consists of blocks like address generator, mode selector, test controller and comparator. Their purpose and functionality is as follows:

Address Generator: This block is simply an adder block, will start with initial address of the memory, increments itself to reach address location. On each visit of particular address location, the controller write or reads as per the algorithmic step specified.

MuX or Mode Selector: This block is to select the circuit in normal mode or in test mode. This can be implemented simply with mux logic itself.

Test Pattern Generator: This block is to select the type of algorithm to test the cell. According to the algorithm primitives order each element will be applied on the cell. Once all the elements are used in the algorithm for test, the next algorithm will start in the given address direction.

Memory under Test: In Memory testing SRAM cell itself will be considered as circuit under test.

Test Controller: It is a logic circuit that helps the algorithm to take a particular operation of write/read either in upward/ downward direction.

Comparator (faulty/not): This circuit is to take actual test output from the cell and expected output, further compares for the cell correctness.

XOR module will be used in this comparator implementation.

BIST controller block is given the input of "March_Select_17n(proposed)" with the collection of final irredundant March elements. Based on the selection of normal or test input, the BIST controller supplies March elements one after the other to the circuit under test in the test mode. The traditional BIST controller will consider each algorithm at a time for testing, but in contrast, the proposed will identify redundancy in the March terms in prior, and the unique March elements only will be allowed for testing. This further reduces the test latency as well as test complexity.

4. Experimentation & Analysis

4.1. Simulation Results of Individual Term used in March Algorithms

This section represents the results observed from each individual term of algorithm chosen, and also the complete algorithm to compare with results obtained from irredundant primitive terms. Each March algorithm comprise tow basic operations i.e write and read. These operations are performed in combinations of multiple write or read as per the primitive used in particular algorithm. Hence, the primitives are selected based on their fault detection capability and observed their simulation results for validity. Faults like SA1, SA0, TF, uses combinations like (w0,r0,w1), (w1,r1,w0), (w0,r0,w1,w0,w1), and (w1,r1,w0,w1,w0) respectively. The other faults followed accordingly with their primitive combinations that are used in various algorithms. Few simulation results for fault detection using primitive combinations are shown in Fig.5 to 8.

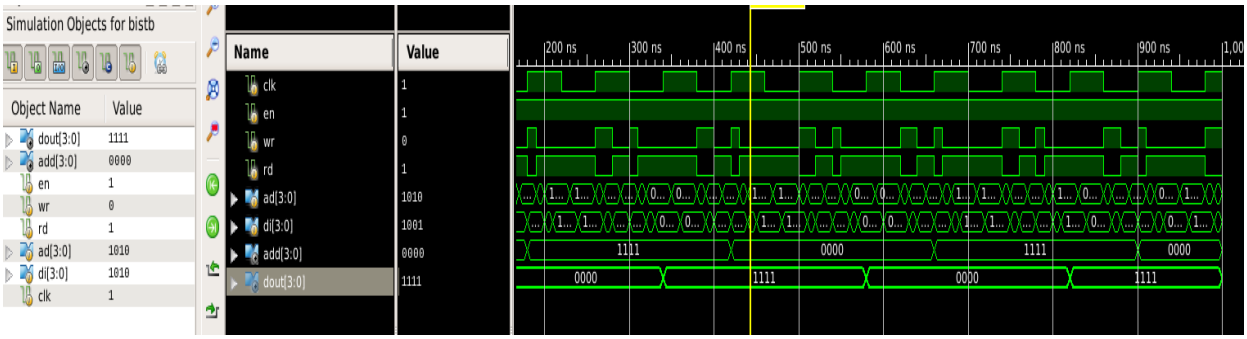


Figure.5. Simulation results for (w0,r0,w1), SA1

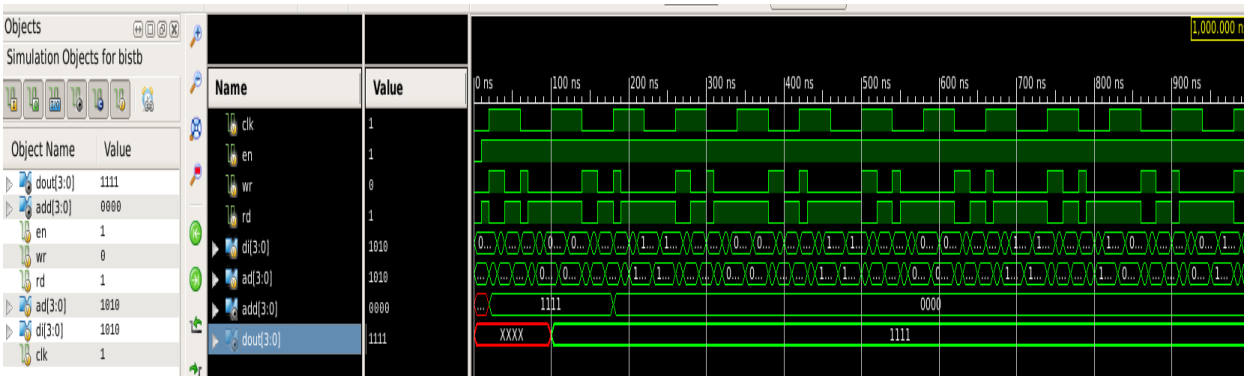


Figure.6. Simulation results for (w1,r1,w0), SA0

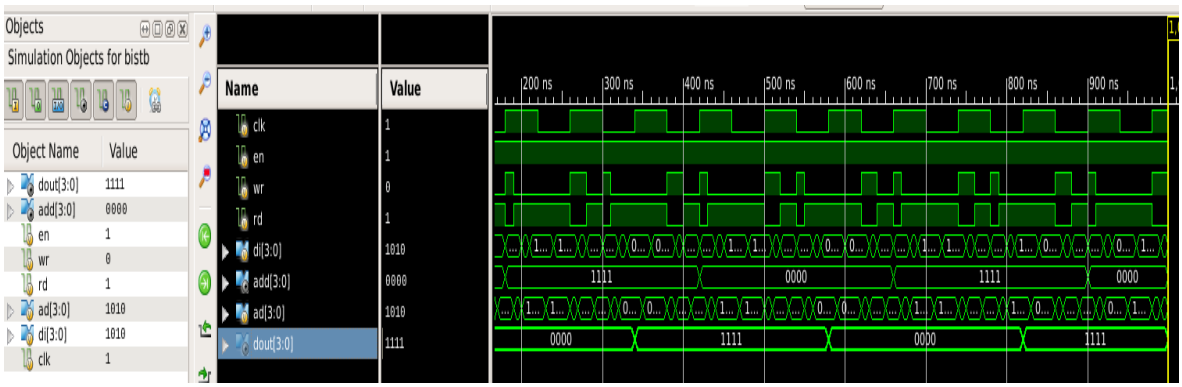


Figure.7. Simulation results for (w0,r0,w1,w0,w1), TF

Figure.8. Simulation results for (w1,r1,w0,w1,w0), TF

4.2. Discussions

Initially, two March algorithms (Mats+ and March A) are considered for comparison, and the selected irredundant terms are collected in the March_Select_17n array as the final list. The rest of the algorithms followed accordingly, and the complete irredundant primitive list is collected in the March_Select_17n array. This array is implemented for testing with various test cases using read and write; corresponding simulation results are observed and are shown in Figs. 9a and 9b. The obtained RTL and technology schematic are shown in Figs. 9c, 9d, and 9e, 9f, respectively.

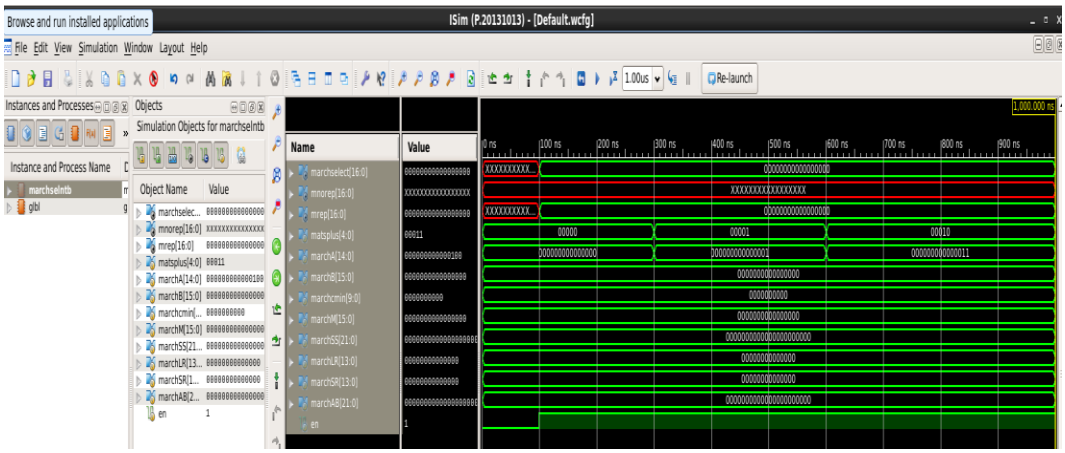


Fig. 9(a). Simulation results for proposed March_Select_17n without fault imposition

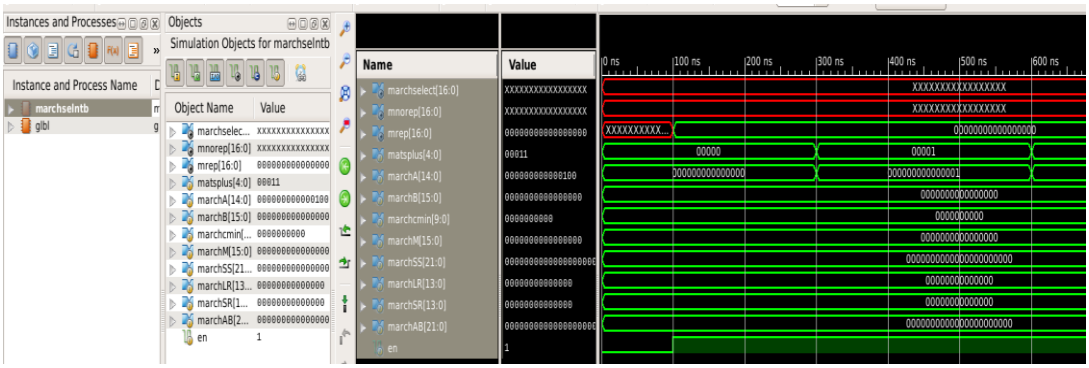
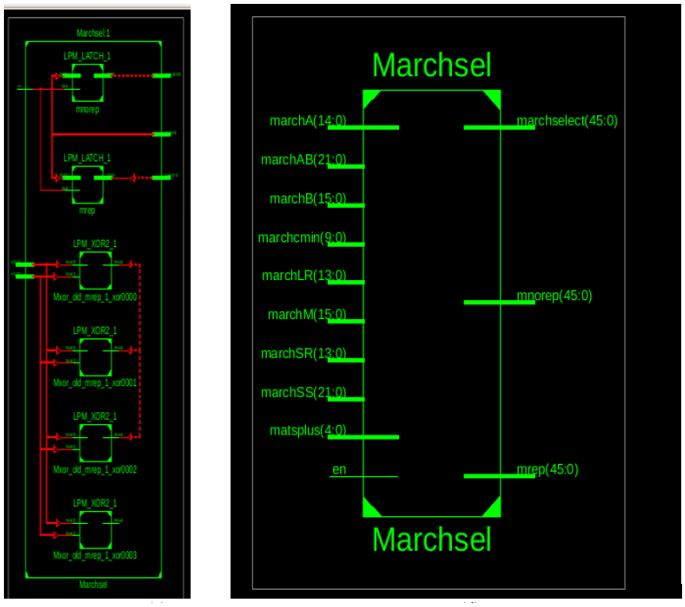


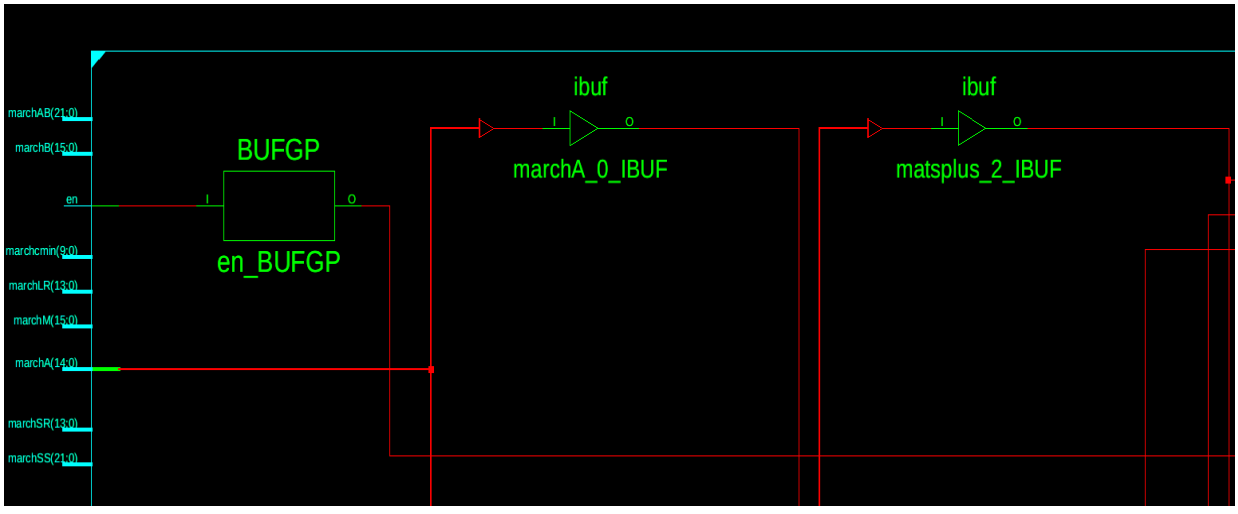
Fig. 9(b). Simulation results for proposed March_Select_17n with fault imposition



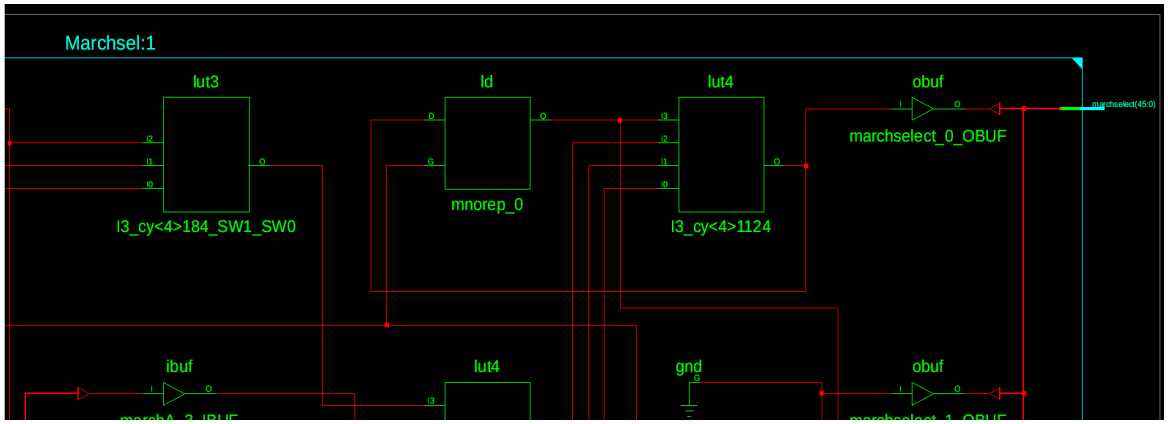
9(c)

9(d)

Figure 9c,d. RTL schematic of first two March algorithm comparison



9(e)



9(f)

Fig. 9e, 9f. Technology schematic

Table 7. Comparative Simulations results for March Algorithms

S.No	Algorithms [8,10,11,13,14, 24]	Complexity	Min. Period, ns	Minimum input arrival time before clock, ns	Maximum output required time after clock, ns	Total XST time. sec
1	MATS+	5n	2.883	4.704	4.285	10
2	March A	15n	2.87	3.884	4.182	11
3	March B	16n	2.87	3.884	4.182	11
4	March C-	10n	2.873	4.688	4.252	10
5	March M	16n	2.85	4.617	4.252	11
6	March SS	22n	2.819	4.665	4.221	11
7	March LR	14n	2.883	4.736	4.285	10
8	March SR	14n	2.842	4.640	4.221	10
9	March AB	22n	2.850	4.665	4.252	11
10	March 46n (proposed)	46n	2.803	4.637	4.182	13
Total Time						108
11	March_Select_17n(proposed)	17n	1.922	5.76	4.20	15

Table 8. Comparative Hardware requirement for selected algorithms

S.No	Type of Algorithm [8,10,11,13,14,24]	Number of Slices:	Number of Slice Flip Flops:	Number of 4 input LUTs:
1	Mats+	3/4656	3/9312	5/9312
2	March A	5/4656	6/9312	9/9312
3	March B	5/4656	6/9312	9/9312
4	March C-	2/4656	3/9312	4/9312
5	March M	2/4656	3/9312	4/9312
6	March SS	2/4656	3/9312	3/9312

7	March LR	3/4656	3/9312	5/9312
8	March SR	2/4656	3/9312	3/9312
9	March AB	2/4656	3/9312	4/9312
10	March 46n (proposed)	4/4656	6/9312	8/9312
11	March_Select_17n(proposed)	10/4656	2/9312	18/9312

Table.9 Comparison of March Test Algorithms in single cell fault coverage

Type of Test Algorithm [8,10,11,13,14,24]	SAF	TF	IRF	RDF	CF	DRDF	WDF	SOF	Dynamic faults	Untestable faults
MATS+	Y	50%	Y	Y	-	-	-	-	-	-
March A	Y	Y	-	-	50%	-	-	-	-	-
March B	Y	Y	-	-	Y	-	-	-	-	-
March C-	Y	Y	Y	Y	Y	-	-	-	-	-
March M	Y	Y	Y	Y	-	Y	Y	-	-	-
March SS	Y	Y	Y	Y	-	Y	Y	-	-	-
March LR	Y	Y	-	-	Y	-	-	-	-	-
March SR	Y	Y	Y	Y	Y	Y	-	Y	-	-
March AB	Y	Y	Y	Y	Y	Y	-	-	Y	-
March_Selct_17n (proposed)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

The selection of unique March primitive and elimination of redundant term during comparison phase is observed by additional LUTS as shown in corresponding simulation results of Fig. 9c to 9f. The area overhead is to be compromised for achieving required speed of testing. The comparisons in simulation parameters terms of primitive complexity, minimum period requirement, minimum input arrival time, maximum output required time, and total XST time are done for chosen algorithms as shown in Table 7. It is observed for individual algorithm level XST times, the overall XST time as 108ns, and for the proposed March_Select_17n as 15ns that results in 86% of speed achievement. In the similar line the hardware complexity is observed in Table 8, at the individual algorithmic level the highest hardware complexity observed in terms of number of 4 input LUTs are 9 (March A, March B), and for the proposed March_Select_17n it costs with 18. The hardware complexity raised by 50%. The overall performance of proposed March_Select_17n is compared with other algorithms under consideration, shown in Table 9.

5. Conclusions & Future Scope

In this paper, aiming for a less latent March Algorithm for high fault coverage, a novel BIST architecture design is discussed. Due to the high test length in the existing March algorithms, the test time increases, and the corresponding BIST implementation also needs high power while in the test phase. Further, undetected hidden fault identification using the parasitic extraction method is also discussed. The limitation of existing March algorithms, which are not adequate to detect hidden and undetected faults, is overcome by the proposed March 46n algorithm. The existing March algorithms are compared in terms of complexity, test time, and hardware requirements with the proposed March 46n algorithm. Further, the proposed less latent March algorithm is used in the design of a novel BIST architecture that compares each term from each March algorithm under

consideration to avoid redundancy and to bring out irredundant March primitives as a test sequence, resulting in March_Select_17n. Using the chosen existing March algorithms, the total time required for testing 16x4 SRAM memory (64 bits) is observed to be 108 seconds (Table 7). But with the reduced irredundant March_Select_17n algorithm, the test time is reduced to 15 seconds with a minimum period of 1.922 ns, a minimum setup time of 5.76 ns, and a maximum hold time of 4.2 ns. For achieving this speed improvement of 86%, the compensation happened in hardware in terms of the number of four input LUTs and the slice requirement have been raised by 50%. In future, it is essential to develop test algorithms with less complexity and less latency for higher end technologies such as at the level of 7nm, by considering low power sub threshold parameters into consideration.

References

1. Dutta T., Pahwa G., Trivedi A.R., Sinha S., Agarwal A., Chauhan Y.S. Performance evaluation of 7nm node negative capacitance FinFET based SRAM. IEEE Electron. Devices Letters. 38 (2017). 1161–1164. https://doi:10.1109/LED.2017.2712365.

2. J. Singh, K. Ramakrishnan, S. Mookerjea, S. Datta, N. Vijaykrishnan, and D. Pradhan. A novel Si-tunnel FET based SRAM design for ultra-low power 0.3 V VDD applications. 15th Asia and South Pacific Design Automation Conference. (2010). 181–186. https://doi.org/10.1109/ASPDAC.2010.5419897.

3. Calimera, Andrea & MacIi, A & MacIi, E & Poncino, Massimo. Design techniques and architectures for low-leakage SRAMs. IEEE Transactions on Circuits and Systems I: Regular Papers. (2012). 59. 1992-2007, https://doi.org/10.1109/TCSI.2012.2185303.

4. Mohammed, Mahmood Uddin & Nizam, Athiya & Ali, Liaquat & Chowdhury, Masud. A Low Leakage SRAM Bitcell Design Based on MOS-Type Graphene Nano-Ribbon FET. (2019). 1-4. https://doi.org/10.1109/ISCAS.2019.8702461.

5. Srinivasa S., Aziz A., Shukla N., Li X., Sampson J., Datta S., Kulkarni J.P., Narayanan V., Gupta S.K. Correlated material enhanced SRAM with robust low power operation. IEEE Trans. Electron Devices. (2016). 63, 4744–4752. https://doi:10.1109/TED.2016.2621125.

6. Rahma M.A., Anis M. Nanometer Variation-Tolerant SRAM: Circuit and Statistical Design for Yield. Springer; New York, NY, USA. (2013). https://doi:10.1007/978-1-4614-1749-1.

7. Lin, Chen-Wei & Chen, Hung-Hsin & Yang, Hao-Yu & Chao, Mango & Huang, Rei-Fu. Fault Models and Test Methods for Sub threshold SRAMs. Computers, IEEE Transactions on. (2010). 62. 1-10. https://doi.10.1109/TEST.2010.5699245

8. Balwinder Singh, Sukhleen Bindra Narang, and Arun Khosla. Modelling and Simulation of Efficient March Algorithm for Memory Testing. S. Ranka et al. (Eds.): IC3 Part II, CCIS 95, (2010). 96-107. https://doi.10.1007/978-3-642-14825-5_9.

9. S. Irobi, Z. Al-Ars and S. Hamdioui. Detecting Memory Faults in the Presence of Bit Line Coupling in SRAM Devices. IEEE International Test Conference, (2010). https://doi.10.1109/TEST.2010.5699246.

10. Aiman Zakwan Jidin, Razaidi Hussin, Mohd Syafiq Mispan, Lee Weng Fook, Loh Wan Ying. Reduced March SR Algorithm for Deep-Submicron SRAM Testing. In proceedings of IEEE International Conference on Semiconductor Electronics (ICSE). 978-1-6654-8246-2/22 IEEE. (2022). https://doi.10.1109/ICSE56004.2022.9863178.

11. Aiman Zakwan Jidin, Razaidi Hussin, Lee Weng Fook, Mohd Syafiq Mispan. A review paper on memory fault models and test algorithms. Bulletin of Electrical Engineering and Informatics, 10(6), (2021), 3083-3093. https://doi.10.11591/eei.v10i6.3048.

12. Tin Quang Bui, Lam Dang Pham, Hieu Minh Nguyen, Viet Thai Nguyen, Thong Chi Le, Trang Hoang. An Effective Architecture of Memory Built-In Self-Test for Wide Range of SRAM. In proceedings of International Conference on Advanced Computing and Applications. (2016). 121-124. https://doi.10.1109/ACOMP.2016.026.

13. Aiman Zakwan Jidin, Razaidi Hussin, Lee Weng Fook, Mohd Syafiq Mispan, Nor Azura Zakaria, Loh Wan Ying, and Norshuhani Zamin. Generation of New Low-Complexity March Algorithms for Optimum Faults Detection in SRAM. (2022). https://doi:10.1109/TCAD.2022.3229281.

14. Muddapu Parvathi, N. Vasantha, K. Satya Parasad. Modified March C - Algorithm for Embedded Memory Testing. International Journal of Electrical and Computer Engineering. 2(5). (2012). 571-576. ISSN: 2088-8708. https://doi.10.5121/vlsic.2012.3504.
15. Nor Azura Zakaria, W.Z.W. Hassan, I.A. Halin, R.M. Sidek, , Xiaoqing Wen. Fault Detection with Optimum March Test Algorithm. Journal of Theoretical and Applied Information Technology. (2013). 47(1). https://doi.10.1109/ISMS.2012.88.
16. Zhikuang Cai, Ying Wang, ShihuanLiu, Kai Lv, Zixuan Wang. A Novel BIST Algorithm for Low-voltage SRAM. IEEE International Test Conference in Asia (ITC-Asia). (2019). 133-138. https://doi.10.1109/ITC-Asia.2019.00036.
17. Balwinder Singha Arun Khoslab Sukhleen Bindra Narangc. Area Overhead and Power Analysis of March Algorithms for Memory BIST. International Conference on Communication Technology and System Design. Procedia Engineering 30 (2012). 930 – 936, Published by Elsevier Ltd. https://doi:10.1016/j.proeng.2012.01.947.
18. M. Venkatesham, S. K. Sinha and M. Parvathi. Study on Paradigm of Variable Length SRAM Embedded Memory Testing. In Proceedings of the Fifth International Conference on Electronics, Communication and Aerospace Technology (ICECAT). (2021). https://doi.10.1109/ICECA52323.2021.9675983.
19. M.Parvathi, K.Satya Prasad, N.Vasantha. Testing of Embedded SRAMs using Parasitic Extraction Method. In Proceedings of Robotic, Vision, Signal Processing and Power Applications (ROVISP), Empowering Research and Innovation, Editors: Ibrahim, H., Iqbal, S., Teoh, S.S., Mustaffa, M.T. (Eds.), ISBN 978-981-10-1721-6, Springer LNEE. (2017). 47-61. https://doi.10.1007/978-981-10-1721-6.
20. V. Maddela, S. K. Sinha, M. Parvathi and V. Sharma. Fault Detection and Analysis in embedded SRAM for sub nanometer technology. In proceedings of International Conference on Applied Artificial Intelligence and Computing (ICAAIC), Salem, India, (2022). 1784-1788. https://doi.10.1109/ICAAIC53929.2022.9793265.
21. M.Parvathi, N. Vasantha, and K. Satya Prasad. New Fault Model Analysis for Embedded SRAM Cell for Deep Submicron Technologies using Parasitic Extraction Method. In Proceedings of IEEE conference on VLSI Systems, Architecture, Technology and Applications (VLSI-SATA). 978-1-4799-7925-7. (2015). 1-6. https://doi.10.1109/VLSI-SATA.2015.7050471.
22. V. Maddela, S. K. Sinha and M. Parvathi. Extraction of Undetectable Faults in 6T- SRAM Cell. 2021 In proceedings of International Conference on Communication, Control and Information Sciences. (2021). 1-5, https://doi:10.1109/ICCISc52257.2021.9484987.
23. Parvathi, M. (2024). SRAM Memory Testing Methods and Analysis: An Approach for Traditional Test Algorithms to ML Models, in Machine Learning Algorithms Using Scikit and Tensor Flow Environments, edited by Puvvadi Baby Maruthi, et al., IGI Global. 295-317. https://doi.org/10.4018/978-1-6684-8531-6.ch015.
24. Jidin, Aiman Zakwan & Hussin, Razaidi & Fook, Lee & Mispan, Mohd Syafiq & Ying, Loh. Automatic generation of user-defined test algorithm description file for memory BIST implementation. International Journal of Reconfigurable and Embedded Systems (IJRES). (2022). 11(103). https://doi.10.11591/ijres.v11.i2.pp103-114.
25. A. Ghukasyan, G. Tshagharyan, G. Harutyunyan and Y. Zorian. Overcoming Embedded Memory Test & Repair Challenges in the Gate-All-Around Era. IEEE 41st VLSI Test Symposium, USA. (2023), 1-4. https://doi:10.1109/VTS56346.2023.10140110.
26. Bernardi, P.; Guerriero, A.M.; Insinga, G.; Paganini, G.; Carnevale, G.; Coppetta, M.; Mischo, W.; Ullmann, R. Built-In Self-Test Architecture Enabling Diagnosis for Massive Embedded Memory Banks in Large SoCs. Electronics (2024). 13(303). https://doi.org/10.3390/electronics13020303.

Authors Contribution Statement:
MV: Fault Models design, circuit simulations, extraction of RC values for chosen technologies.

MP: Fault model primitive conversion, simulation testing, and verification, performance comparison with the existing algorithms based on literature, manuscript preparation, and submission.

Funding: The project is solely developed in the institute lab environment with the existing test equipment and tools under no additional aid or cost, without any funds or grants support.

Competing Interests: The Authors in this paper declares that there are no conflicts of interest regarding the publication of this manuscript.

Data Availability: Upon reasonable request, the authors in this paper will provide the experimental values from this work as well as the computational work from the current investigation.