International Journal of Electrical and Computer Engineering (IJECE)
Vol. 15, No. 1, February 2025, pp. 163~173
ISSN: 2088-8708, DOI: 10.11591/ijece.v15i1.pp163-173

A new 13N-complexity memory built-in self-test algorithm to balance static random access memory static fault coverage and test time

Aiman Zakwan Jidin[1,2], Razaidi Hussin[3], Mohd Syafiq Mispan[1,2], Lee Weng Fook[4]
[1]Fakulti Teknologi dan Kejuruteraan Elektronik dan Komputer, Universiti Teknikal Malaysia Melaka, Durian Tunggal, Malaysia
[2]Center for Telecommunication Research and Innovation, Universiti Teknikal Malaysia Melaka, Durian Tunggal, Malaysia
[3]Faculty of Electronics Engineering and Technology, Universiti Malaysia Perlis, Arau, Malaysia
[4]Emerald System Design Center, Bayan Lepas, Malaysia

Article Info

Article history:

Received Apr 23, 2024
Revised Aug 27, 2024
Accepted Sep 3, 2024

Keywords:

March test algorithm
Memory built-in self-test
Memory fault coverage
Randon access memory
Unlinked static fault

ABSTRACT

As memories dominate the system-on-chip (SoC), their quality significantly impacts the chip manufacturing yield. There is a growing need to reduce the chip production time and cost, which mainly depends on the testing phase. Hence, a memory built-in self-test (MBIST) utilizing a low-complexity, high-fault-coverage test algorithm is essential for efficient and thorough memory testing. The March AZ1 algorithm, with 13N complexity, was created earlier to balance the test length and fault coverage. However, poor positioning of a write operation in its test sequence caused the reduction of the transition coupling fault (CFtr) detection. This paper presents the creation of the March AZ algorithm, modified from the March AZ1 algorithm, to increase CFtr coverage while preserving the same complexity. It was accomplished by analyzing the fault coverage offered by the March AZ1 algorithm and then reorganizing its test sequence to address the limitation in detecting CFtr. The newly produced March AZ1 algorithm was successfully implemented in an MBIST controller. The simulation tests validated its functionality and demonstrated that the CFtr coverage was enhanced from 62.5% to 75%, achieving an overall fault coverage of 83.3%. Therefore, with 13N complexity, it offers the best fault coverage among all the existing test algorithms with a complexity below 18N.

Corresponding Author:

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

1. INTRODUCTION

Memory testing is becoming essential in designing system-on-chips (SoCs) since they are nowadays memory dominant, where the memories use up to 94% of their areas [1]–[3]. As a result, a good chip manufacturing yield is significantly influenced by memory quality [2], [4]. Additionally, memories are more prone to failure than sequential logic due to their high-density nature [5]. Many static memory fault models are established to represent the actual manufacturing defect at the logical abstraction level, as described in Table 1. Stuck-at fault (SAF), transition fault (TF), read destructive fault (RDF), incorrect read fault (IRF), deceptive read destructive fault (DRDF), and write disturb fault (WDF) are classified as single-cell faults (SCF), whose occurrences are sensitized and detected in the same memory cell. Meanwhile, transition coupling fault (CFtr), deceptive read destructive coupling fault (CFdrd), and write disturb coupling fault

(CFwd) are the double-cell faults (DCF), where the fault detected in a victim cell (v) is caused by the state of the aggressor cell (a) [6]–[8].

Each single-cell faults (SCF) is described by its faults primitives (FP) and conventionally notated as <$S/F/O$>, where S indicates the fault sensitizing operation(s), F is the v state if faulty, and O is the read output (if applicable) [9], [10]. Each SCF has 2 FPs since x equals either 0 or 1. Meanwhile, a DCF's FP is notated as <S_v; $S_a/F/O$>, where S_a and S_v are the sensitizing operators or states at the a and v cells, respectively. Each DCF consists of 8 since two possible scenarios are anticipated: the a-cell's address is inferior ($a < v$) or superior ($a > v$) to the v-cell address. Since numerous memories on a chip need to be tested automatically, memory built-in self-test (MBIST) is a widely used method for memory testing [11]. It can automate test executions and output checking, and thus, the dependency on costly testing equipment is reduced [10], [12]–[14]. It performs a series of test operations defined by the applied test algorithm, consisting of reading (rx) or writing (wx) the x logic to every cell inside the tested memory [15], [16]. These test operations are conducted in the ascending (⇑) or descending (⇓) address order.

Table 1. The descriptions of unlinked static fault models

Fault	FP	Faulty Behavior	Detection Requirement
SAF	$< x/x'/- >$	v-cell is stuck at the x-state regardless of the input value.	Write x' to cells followed by a read operation.
TF	$< xwx'/x/- >$	v-cell fails to transit from x to x'.	Write x' to x-state cells followed by a read operation.
RDF	$< rx/x'/x' >$	A read from the v-cell unexpectedly changes its state and returns an incorrect value.	Read from x-state cells.
IRF	$< rx/x/x' >$	A read from the v-cell unexpectedly returns an incorrect value without changing its state.	Read from x-state cells.
DRDF	$< rx/x'/x >$	A read from the v-cell unexpectedly changes its state but returns the correct value.	Read twice from x-state cells.
WDF	$< xwx/x'/- >$	A write-to-x to the v-cell that contains an x unexpectedly changes its state to x'.	Write x to x-state cells followed by a read operation.
CFtr	$< x;\ xwx'/x/- >_{a>v,}$ $< x;\ xwx'/x/- >_{a<v,}$ $< x';\ xwx'/x/- >_{a>v,}$ $< x';\ xwx'/x/- >_{a>v}$	v-cell fails to transit from x to x' when its a-cell is in a given state (x or x').	Write x' to x-state cells followed by a read operation when a-cell is in the x or x' state.
CFdrd	$< x;\ rx/x'/x >_{a>v,}$ $< x;\ rx/x'/x >_{a<v,}$ $< x';\ rx/x'/x >_{a>v,}$ $< x';\ rx/x'/x >_{a<v}$	A read from the v-cell unexpectedly changes its state but returns the correct value when its a-cell is in a given state (x or x').	Read twice from x-state cells when a-cell is in the x or x' state.
CFwd	$< x;\ xwx/x'/- >_{a>v,}$ $< x;\ xwx/x'/- >_{a<v,}$ $< x';\ xwx/x'/- >_{a>v,}$ $< x';\ xwx/x'/- >_{a>v}$	A write-to-x to the v-cell that contains an x unexpectedly changes its state to x' when its a-cell is in a given state (x or x').	Write x to x-state cells followed by a read operation when a-cell is in the x or x' state.

The semiconductor industry prefers March test algorithms since they have design simplicity and linear complexity, defined in the order of N (the size of the tested memory) [2], [17]–[19]. Several March test algorithms are listed in Table 2. They are distinguished by their test sequences, complexities, and fault coverages. The stuck-at fault (SAF) represents incorrect read fault (IRF) and read destructive fault (RDF) coverages since their detection requirements are alike [20]. The shown fault coverage is computed by dividing the number of detectable FPs by 2 for each SCF and by 8 for each DCF. A March test algorithm with a complexity higher than or equal to 18N, like the March MSS algorithm [20], offers complete coverage of all targeted static faults in static random access memory (SRAM). Meanwhile, a lower-complexity test algorithm is necessary to produce a shorter test time and lower cost. However, based on Table 2, it has poor coverage of DRDF, WDF, CFdrd, and CFwd, which are relevant to memories fabricated using the nanometer process technologies [21].

Therefore, the March AZ1 (13N) and March AZ2 (14N) algorithms were created to balance the complexity and coverage of the targeted faults [22]. The former offers 80.6% of overall fault coverage, providing complete SCF coverage, 62.5% coverage of CFtr, and 75% coverage of CFdrd and CFwd. Meanwhile, the latter offers a slight enhancement in CFtr coverage (75%), thus offering 83.3% of overall fault coverage, the best among all existing below 18N-complexity test algorithms [22]. The latter can detect a specific FP of CFtr (CFtr $< 1;\ 1w0/1/- >_{a>v}$) that is undetectable by the former. However, its complexity is 1N more than the former, requiring a slightly longer test time.

This paper presents the March AZ algorithm, a new test algorithm that improves the March AZ1 algorithm's coverage of CFtr while maintaining its complexity at 13N. It was accomplished by analyzing the detectability of all FPs using an automated fault detection analyzer, which identifies each FP's sensitizing and detecting test operations within the March AZ1 algorithm's test sequence. Subsequently, the weakness in CFtr detection was recognized from the analysis output and addressed through test operations and test elements reorganization. The functionality of the new March AZ algorithm was verified via a simulation conducted using the implemented MBIST controller. Finally, its fault coverage was evaluated by performing a test on a fault-injected SRAM as the memory model in the simulation. The results demonstrate that the new March AZ algorithm provides similar unlinked static fault coverage to the March AZ2 algorithm, which offers the best coverage to date among all existing test algorithms with a complexity lower than 18N [22]. However, with 1N complexity lesser than the latter, the former produces a faster test completion time and, thus, can reduce the test cost.

Table 2. Several March algorithms test sequences, complexities, and fault coverages

Test algorithm	Complexity	Test sequence	SCF					DCF	
			SAF	TF	DRDF	WDF	CFtr	CFdrd	CFwd
March C- [6]	10N	⇕(w0); ⇑(r0, w1); ⇑(r1, w0); ⇓(r0, w1); ⇓(r1, w0); ⇕(r0)	100%	100%	0%	0%	100%	0%	0%
March CL [23]	12N	⇕(w0); ⇑(r0, w1); ⇑(r1, r1, w0); ⇓(r0, w1, r1); ⇓(r1, w0); ⇕(r0)	100%	100%	50%	0%	100%	50%	0%
March LR [24]	14N	⇕(w0); ⇓(r0, w1); ⇑(r1, w0, r0, w1); ⇑(r1, w0); ⇑(r0, w1, r1, w0); ⇑(r0)	100%	100%	0%	0%	100%	0%	0%
March SR [6]	14N	⇕(w0); ⇑(r0, w1, r1, w0); ⇑(r0, r0); ⇑(w1); ⇓(r1, w0, r0, w1); ⇓(r1, r1)	100%	100%	100%	0%	100%	50%	0%
March C+ [25]	14N	⇕(w0); ⇑(r0, w1, r1); ⇑(r1, w0, r0); ⇓(r0, w1, r1); ⇓(r1, w0, r0); ⇕(r0)	100%	100%	100%	0%	100%	100%	0%
March AZ1 [22]	13N	⇕(w0); ⇓(r0, w1); ⇑(w1, r1, r1, w0); ⇑(w0, r0); ⇑(r0, w1, w1, r1); ⇑(r1)	100%	100%	100%	100%	62.5%	75%	75%
March AZ2 [22]	14N	⇕(w0); ⇓(w0, r0); ⇑(r0, w1, w1, r1); ⇑(r1, w0); ⇓(r0, w1, w1, r1); ⇑(r1);	100%	100%	100%	100%	75%	75%	75%
March MSS [20]	18N	⇕(w0); ⇑(r0, r0, w1, w1); ⇑(r1, r1, w0, w0); ⇓(r0, r0, w1, w1); ⇓(r1, r1, w0, w0); ⇕(r0)	100%	100%	100%	100%	100%	100%	100%
March SS [2]	22N	⇕(w0); ⇑(r0, r0, w1, w1); ⇑(r1, r1, w0, w0); ⇓(r0, r0, w1, w1); ⇓(r1, r1, w0, w0); ⇕(r0)	100%	100%	100%	100%	100%	100%	100%

2. THE MARCH AZ1 ALGORITHM REVIEW

Table 3 shows the six test elements in the March AZ1 algorithm's test sequence, labelled TE_0 through TE_5, separated by semicolons [22]. The test elements will be executed sequentially during the test: All test operations defined in TE_i must be performed on all memory cells before moving on to the next TE_{i+1}. Plus, 13 read or write operations must be performed on all N memory cells, explaining its 13N complexity.

Table 3. The March AZ1 algorithm descriptions

Test element	Test sequence	Test description
TE_0	⇕(w0)	All cells are set to 0.
TE_1	⇓(w1)	All cells are set to 1 in descending address order.
TE_2	⇑(w1, r1, r1, w0)	All cells are sequentially set to 1, read twice (expecting a 1 at the output), and set to 0 in ascending address order.
TE_3	⇑(w0, r0)	All cells are sequentially set to 0 before being read (expecting a 0 at the output) in ascending address order.
TE_4	⇑(r0, w1, w1, r1)	All cells are sequentially read (expecting 0), set to 1 twice, and reread (expecting 1) in ascending address order.
TE_5	⇑(r1)	All cells are read (expecting 1) in ascending address order.

A fault detection analysis was conducted on the March AZ1 algorithm using a developed fault detection analyzer that identifies the sensitizer and detector pairs for all targeted FPs within the test sequence [26]. The flowchart in Figure 1 depicts the analysis process that was conducted. Once the March AZ1 algorithm's test sequence was read and extracted, the analyzer determined the cell trend of each test element, which indicates how the cells' states are changed when a test element is executed during the test. Next, it

identified all possible sensitizer and detector pairs of each detectable FP found within the analyzed test sequence, starting from the first test operation defined in TE_0 until the last test operation in TE_5, based on their detection requirements described in Table 1. The process was repeated for all 36 targeted FPs. Specifically, for DCF detection analysis, the predetermined cell trends, which indicate the way all memory cells' contents change during the execution of a test element, are needed to decide the corresponding FP (either $a < v$ or $a > v$) [26].

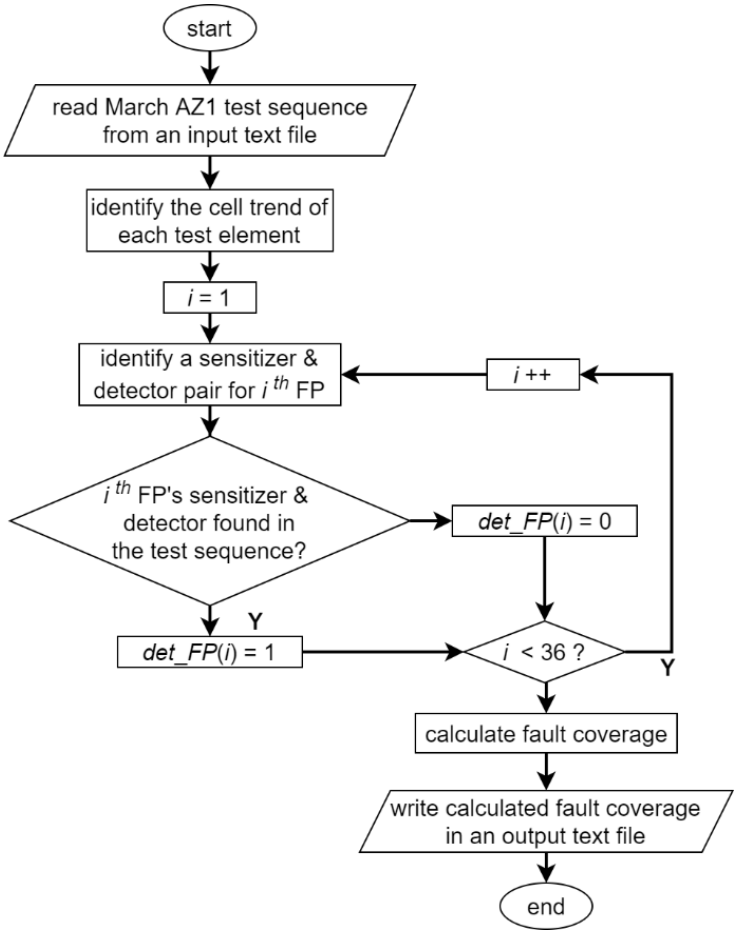

Figure 1. The fault detection analysis process flow

Each FP was associated with a bit in the *det_FP* bus for fault coverage computation purposes, which was set to high when its sensitizer-detector pair was identified within the analyzed test sequence. Therefore, the fault coverage was computed by calculating the high *det_FP* bits divided by the total FPs 36. Table 4 shows the sensitizer and detector pairs for each FP identified within the March AZ1 algorithm's test sequence during the analysis. The $TE_{i\text{-}j}$ notations signify that the jth test operation in TE_i is recognized as a sensitizing or detecting operation for a particular FP.

Table 4 demonstrates that all targeted SCFs are detectable since their FPs have at least one identified sensitizer-detector pair. So, the March AZ1 algorithm offers 100% of all SCFs. Additionally, the fault analyzer identified the sensitizer-detector pairs for 5 FPs of CFtr. Hence, CFtr coverage equals 62.5% (5 detectable FPs out of 8). Meanwhile, the fault analyzer identified the sensitizer-detector pairs for 6 FPs of each CFdrd and CFwd. Hence, the CFdrd and CFwd coverages equal 75% (6 detectable FPs out of 8). Consequently, the fault detection analysis derived the expected fault coverage by the March AZ1 algorithm, as presented in Table 2. By comparing its fault coverage to the March AZ2 algorithm with 14N test complexity [22], which is available in Table 2, the analyzed March AZ1 algorithm has a slightly lower coverage of CFtr since it cannot detect the CFtr $< 1; 1w0/1/->_{a>v}$, as proven by the analysis output presented in Table 4.

Table 4. The analysis of the March AZ1 algorithm's fault coverage

Fault	FP	Identified (Sensitizer, Detector)	Detection status	Fault coverage
SAF	<1/0/->	(TE_{2-1}, TE_{2-2}), (TE_{4-3}, TE_{4-4})	Yes	2/2 (100%)
	<0/1/->	(TE_{3-1}, TE_{3-2})	Yes	
TF	<0w1/0/->	(TE_{1-1}, TE_{2-2}), (TE_{4-2}, TE_{4-4})	Yes	2/2 (100%)
	<1w0/1/->	(TE_{2-4}, TE_{3-2})	Yes	
RDF	<r0/1/1>	(TE_{2-1}, TE_{2-2}), (TE_{4-3}, TE_{4-4})	Yes	2/2 (100%)
	<r1/0/0>	(TE_{3-1}, TE_{3-2})	Yes	
IRF	<r0/0/1>	(TE_{2-1}, TE_{2-2}), (TE_{4-3}, TE_{4-4})	Yes	2/2 (100%)
	<r1/1/0>	(TE_{3-1}, TE_{3-2})	Yes	
DRDF	<r0/1/0>	(TE_{3-2}, TE_{4-1})	Yes	2/2 (100%)
	<r1/0/1>	(TE_{2-2}, TE_{2-3}), (TE_{4-4}, TE_{5-1})	Yes	
WDF	<0w0/1/->	(TE_{3-1}, TE_{3-2})	Yes	2/2 (100%)
	<1w1/0/->	(TE_{2-1}, TE_{2-2}), (TE_{4-3}, TE_{4-4})	Yes	
CFtr	<0; 0w1/0/->$_{a>v}$	(TE_{4-2}, TE_{4-4})	Yes	5/8 (62.5%)
	<0; 0w1/0/->$_{a<v}$	(TE_{1-1}, TE_{2-2})	Yes	
	<1; 0w1/0/->$_{a>v}$	(TE_{1-1}, TE_{2-2})	Yes	
	<1; 0w1/0/->$_{a<v}$	(TE_{4-2}, TE_{4-4})	Yes	
	<0; 1w0/1/->$_{a>v}$	Not found	No	
	<0; 1w0/1/->$_{a<v}$	(TE_{2-4}, TE_{3-2})	Yes	
	<1; 1w0/1/->$_{a>v}$	Not found	No	
	<1; 1w0/1/->$_{a<v}$	Not found	No	
CFdrd	<0; r0/1/0>$_{a>v}$	(TE_{3-2}, TE_{4-1})	Yes	6/8 (75%)
	<0; r0/1/0>$_{a<v}$	(TE_{3-2}, TE_{4-1})	Yes	
	<1; r0/1/0>$_{a>v}$	Not found	No	
	<1; r0/1/0>$_{a<v}$	Not found	No	
	<0; r1/0/1>$_{a>v}$	(TE_{4-4}, TE_{5-1})	Yes	
	<0; r1/0/1>$_{a<v}$	(TE_{2-2}, TE_{2-3})	Yes	
	<1; r1/0/1>$_{a>v}$	(TE_{2-2}, TE_{2-3})	Yes	
	<1; r1/0/1>$_{a<v}$	(TE_{4-4}, TE_{5-1})	Yes	
CFwd	<0; 0w0/1/->$_{a>v}$	(TE_{3-1}, TE_{3-2})	Yes	6/8 (75%)
	<0; 0w0/1/->$_{a<v}$	(TE_{3-1}, TE_{3-2})	Yes	
	<1; 0w0 /1/->$_{a>v}$	Not found	No	
	<1; 0w0/1/->$_{a<v}$	Not found	No	
	<0; 1w1/0/->$_{a>v}$	(TE_{4-3}, TE_{4-4})	Yes	
	<0; 1w1/0/->$_{a<v}$	(TE_{2-1}, TE_{2-2})	Yes	
	<1; 1w1/0/->$_{a>v}$	(TE_{2-1}, TE_{2-2})	Yes	
	<1; 1w1/0/->$_{a<v}$	(TE_{4-3}, TE_{4-4})	Yes	

3. THE NEW MARCH AZ ALGORITHM CREATION

As stated in Table 1, a CFtr occurrence can be sensitized in a *v*-cell by writing an *x'* logic to the cell that contains an *x* logic when the *a*-cell is in a given state. Then, the write operation is succeeded by a read operation to detect any faulty behavior from the *v*-cell. According to [20], [22], the CFtr <1; 1w0/1/->$_{a>v}$ can be sensitized and detected by using one of the following test sequences, where *F*(*x*) represents any operation that produces an *x*-state in the memory cells and * indicates that the associated operations are optional:

- Condition 3.1: $\Updownarrow(\ldots, F(1))$; $\Uparrow(F(1)*, w0, w0*\ r0, F(0)*)$;
- Condition 3.2: $\Updownarrow(\ldots, F(1))$; $\Uparrow(F(1)*, w0, w0*)$; $\Updownarrow(r0, \ldots)$;
- Condition 3.3: $\Updownarrow(\ldots, F(1))$; $\Updownarrow(w0, w0*, r0, F(0)*, F(1))$;

In the March AZ1 algorithm's test sequence, the cells' transition from 1 to 0 can only occur at TE_2: $\Uparrow(w1, r1, r1, w0)$, where the $w0$ operation should set the cells' states to 0. A subsequent read operation can then detect the faulty behaviour caused by the CFtr <1; 1w0/1/->$_{a>v}$. Yet, this $w0$ operation in TE_2 is followed by another w0 operation in TE_3: $\Uparrow(w0, r0)$ before the required read operation. Therefore, this test sequence does not meet Condition 3.1 to Condition 3.3 requirements. In fact, the $w0$ operation in TE_3 acts as the CFtr <1; 1w0/1/->$_{a>v}$ fault recovered, masking its occurrence from being detected by the $r0$ operation in TE_3, as illustrated in Figure 2 using a 4-cell memory as the example where the *v*-cell and *a*-cell are set to address 0 and 2, respectively. In TE_2 operation, the *v*-cell, affected by the CFtr <1; 1w0/1/->$_{a>v}$ fault, fails to change its state to low when the *w*0 operation is performed since its *a*-cell (cell 2) is in a high state. Somehow, the *w*0 operation in TE_3 successfully changes its state to low since its *a*-cell is no longer in a high state.

So, the March AZ1 algorithm's TE_2 and TE_3 were reorganized to solve this issue: the $w0$ operation in TE_3 was moved to the end of TE_2. Subsequently, the newly modified TE_2 consists of $\Uparrow(w1, r1, r1, w0, w0)$ test sequence, whereas the new test sequence for TE_3 is $\Uparrow(r0)$. Consequently, the newly reorganized TE_2 and TE_3 fulfil the required test sequence defined by Condition 3.2 and should be able to detect the CFtr <1; 1w0/1/->$_{a>v}$. The newly modified March AZ1 algorithm is called the March AZ algorithm, with the same 13N complexity and the new test sequence: $\Updownarrow(w0)$; $\Downarrow(r0, w1)$; $\Uparrow(w1, r1, r1, w0, w0)$; $\Uparrow(r0)$; $\Uparrow(r0, w1, w1, r1)$; $\Uparrow(r1)$. The fault detection analysis was redone using the new March AZ algorithm. The analysis results in Table 5 prove that the CFtr coverage was improved from 62.5% by the March AZ1, as stated in Table 4, to

75% by enabling the detection of the CFtr <1; 1w0/1/->$_{a>v}$. Furthermore, it also proves that the proposed test element reorganization did not affect the detections of other FPs as their coverages remain unchanged.

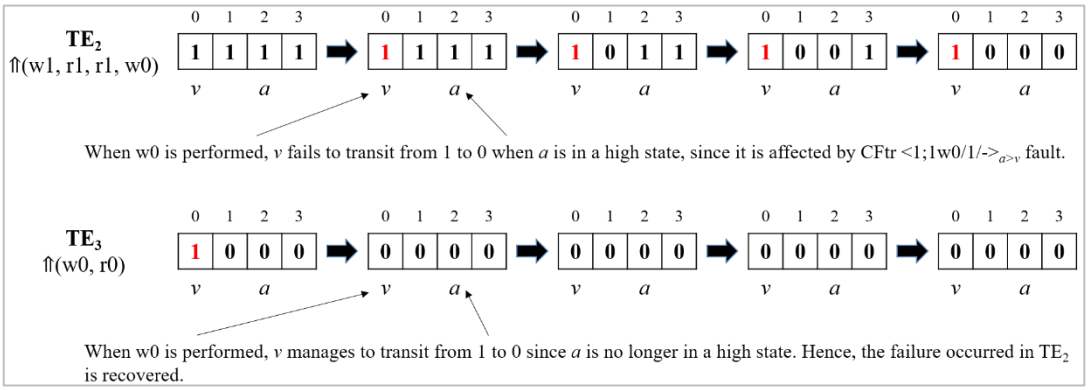

Figure 2. Illustration of the CFtr <1; 1w0/1/->$_{a>v}$ fault recovering at TE3 of the March AZ1 algorithm

Table 5. The new March AZ algorithm's fault detection analysis

Fault	FP	Identified (Sensitizer, Detector)	Detection status	Fault coverage
SAF	<1/0/->	(TE_{2-1}, TE_{2-2}), (TE_{4-3}, TE_{4-4})	Yes	2/2 (100%)
	<0/1/->	(TE_{2-5}, TE_{3-1})	Yes	
TF	<0w1/0/->	(TE_{1-1}, TE_{2-2}), (TE_{4-2}, TE_{4-4})	Yes	2/2 (100%)
	<1w0/1/->	(TE_{2-4}, TE_{3-1})	Yes	
RDF	<r0/1/1>	(TE_{2-1}, TE_{2-2}), (TE_{4-3}, TE_{4-4})	Yes	2/2 (100%)
	<r1/0/0>	(TE_{2-5}, TE_{3-1})	Yes	
IRF	<r0/0/1>	(TE_{2-1}, TE_{2-2}), (TE_{4-3}, TE_{4-4})	Yes	2/2 (100%)
	<r1/1/0>	(TE_{2-5}, TE_{3-1})	Yes	
DRDF	<r0/1/0>	(TE_{3-1}, TE_{4-1})	Yes	2/2 (100%)
	<r1/0/1>	(TE_{2-2}, TE_{2-3}), (TE_{4-4}, TE_{5-1})	Yes	
WDF	<0w0/1/->	(TE_{2-5}, TE_{3-1})	Yes	2/2 (100%)
	<1w1/0/->	(TE_{2-1}, TE_{2-2}), (TE_{4-3}, TE_{4-4})	Yes	
CFtr	<0; 0w1/0/->$_{a>v}$	(TE_{4-2}, TE_{4-4})	Yes	6/8 (75%)
	<0; 0w1/0/->$_{a<v}$	(TE_{1-1}, TE_{2-2})	Yes	
	<1; 0w1/0/->$_{a>v}$	(TE_{1-1}, TE_{2-2})	Yes	
	<1; 0w1/0/->$_{a<v}$	(TE_{4-2}, TE_{4-4})	Yes	
	<0; 1w0/1/->$_{a>v}$	Not found	No	
	<0; 1w0/1/->$_{a<v}$	(TE_{2-4}, TE_{3-1})	Yes	
	<1; 1w0/1/->$_{a>v}$	(TE_{2-4}, TE_{3-1})	Yes	
	<1; 1w0/1/->$_{a<v}$	Not found	No	
CFdrd	<0; r0/1/0>$_{a>v}$	(TE_{3-1}, TE_{4-1})	Yes	6/8 (75%)
	<0; r0/1/0>$_{a<v}$	(TE_{3-1}, TE_{4-1})	Yes	
	<1; r0/1/0>$_{a>v}$	Not found	No	
	<1; r0/1/0>$_{a<v}$	Not found	No	
	<0; r1/0/1>$_{a>v}$	(TE_{4-4}, TE_{5-1})	Yes	
	<0; r1/0/1>$_{a<v}$	(TE_{2-2}, TE_{2-3})	Yes	
	<1; r1/0/1>$_{a>v}$	(TE_{2-2}, TE_{2-3})	Yes	
	<1; r1/0/1>$_{a<v}$	(TE_{4-4}, TE_{5-1})	Yes	
CFwd	<0; 0w0/1/->$_{a>v}$	(TE_{2-5}, TE_{3-1})	Yes	6/8 (75%)
	<0; 0w0/1/->$_{a<v}$	(TE_{2-5}, TE_{3-1})	Yes	
	<1; 0w0/1/->$_{a>v}$	Not found	No	
	<1; 0w0/1/->$_{a<v}$	Not found	No	
	<0; 1w1/0/->$_{a>v}$	(TE_{4-3}, TE_{4-4})	Yes	
	<0; 1w1/0/->$_{a<v}$	(TE_{2-1}, TE_{2-2})	Yes	
	<1; 1w1/0/->$_{a>v}$	(TE_{2-1}, TE_{2-2})	Yes	
	<1; 1w1/0/->$_{a<v}$	(TE_{4-3}, TE_{4-4})	Yes	

4. RESULTS AND DISCUSSION

The new March AZ algorithm's test sequence was hard-coded as the user-defined algorithm (UDA) inside an MBIST controller, generated using Siemens Tessent memory BIST software as the electronic design automation (EDA) tool. After that, it was simulated in the siemens QuestaSim simulator using the created test benches and test patterns. Two different tests were conducted in simulations on the implemented MBIST controller: a test on a fault-free memory and a test on a fault-injected memory.

4.1. Test on the fault-free memory model

This test assessed the functionality of the created MBIST with the new March AZ algorithm as the UDA. It was evaluated by observing the *MBISTPG_GO* flag, which should stay high until the test was completed or when the *MBISTPG_DONE* flag was asserted. The test completion time was also measured, which should equal the UDA's complexity multiplied by N and the clock period (20 ns). A 1-kB memory was used as the test memory; thus, N equals 1024. Figure 3 presents the simulation waveform in QuestaSim for the fault-free memory test. The output data read from the memory cell (*dout*) was compared to the expected data generated by the MBIST controller (*BIST_EXPECT_DATA*) whenever *CMP_EN* is high. The *MBISTPG_GO* flag was asserted to indicate the start of the test and remained high until the test completion, as signified by a high *MBISTPG_DONE* flag. This observation signifies no discrepancy between *dout* and *BIST_EXPECT_DATA* during the comparison. Additionally, the test completion time, measured from the start until the end of the test, equals 266,240 ns. It is similar to the expected test completion time since 13*1024*20 ns equals 266,240 ns. Therefore, this test's observation validated the implemented MBIST's correct functionality, which used the March AZ as the UDA. Furthermore, it also demonstrates that the new March AZ algorithm produces a test 20,480 ns shorter than the March AZ2 algorithm, which requires 286,720 ns [22], on the same memory model under test.

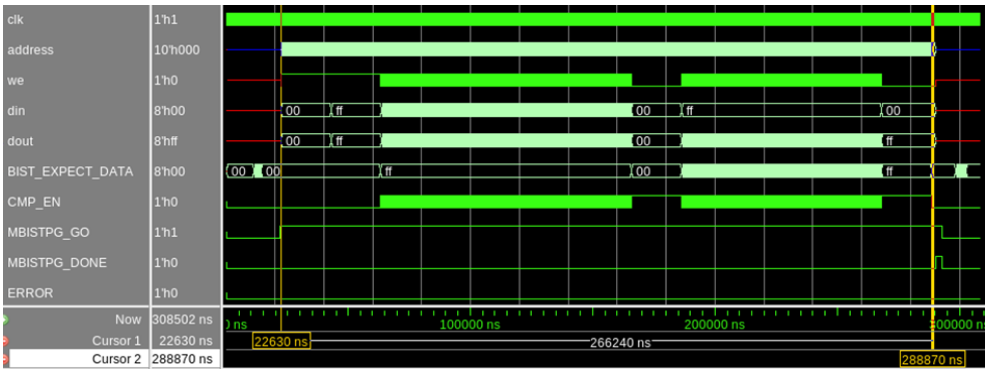

Figure 3. The simulation waveform observed in QuestaSim from the test on the fault-free memory model

4.2. Test on the fault-injected memory model

This test assessed the fault coverage of the applied March AZ algorithm. The behavioral model of the memory used in the previous test was modified to introduce all FPs to be detected and, hence, imitate their faulty behaviors during the test in the simulation. Figure 4 shows the distribution of the affected victim cells for all introduced FPs (notated as V_i) and the corresponding aggressor cells (notated as A_i) for each DCF; the addresses of these cells were randomly generated within the given specifications, e.g., the address of A_i must be greater than V_i for all FPs with $a>v$. The SAF $<x/x'/->$ was introduced by fixing the input data *din* value to *x* when the write-enable signal *we* was high and the *address* was equal to the affected cell chosen. The RDF $<rx/x'/x'>$ occurrences were introduced by altering the low state of the affected *v*-cells to high when they were about to be read (*we* is low). In contrast, the IRF $<rx/x/x'>$ occurrences were replicated by overwriting the output *dout* value to *x'* when the affected *v*-cells contained logic *x* and were about to be read.

The CFtr $<y;\ xwx'/x/->_{a>v}$ and CFtr $<x;\ xwx'/x/->_{a>v}$ occurrences were produced by cancelling the w*x'* operation onto the affected *v*-cells that contained *x* when the corresponding *a*-cells are in *y*-state, where $y=\{0, 1\}$. At the same time, the TF$<xwx'/x/->$ is considered detectable when at least one CFtr $<y;\ xwx'/x/->$ was detected. Meanwhile, the occurrences of CFdrd $<y;\ rx/x'/x>_{a>v}$ and CFdrd $<y;\ rx/x'/x>_{a>v}$ were imitated by altering the contents of the affected *v*-cells containing logic *x* to *x'* when they were read and the corresponding *a*-cells contained logic *y*. Similarly, DRDF $<rx/x'/x>$ is considered detectable when at least one CFdrd $<y;\ rx/x'/x>$ was detected. Lastly, the occurrences of CFwd $<y;\ xwx/x'/->_{a>v}$ and CFwd $<y;\ xwx/x'/->_{a<v}$ were created by changing the input *din* value to *x'* when the affected *v*-cells contained logic *x* and were about to be rewritten to *x*, and when the corresponding *a*-cells stored logic *y*. Hence, WDF $<xwx/x'/->$ is deemed detectable when at least one CFwd $<y;\ xwx/x'/->$ was detected.

Figure 5 displays the simulation waveform of the MBIST operation on the fault-injected memory using the new March AZ as the UDA. In this test, the values of all fault detection flags were observed when the test was completed (indicated by a high *MBISTPG_DONE* flag) and recorded in Table 6. Therefore, the March AZ algorithm's fault coverage was determined by counting the high bits in each fault's detection flag. It shows that the March AZ algorithm detected the injected CFtr $<1;\ 1w0/1/->_{a>v}$, which was undetected by

the March AZ1 algorithm. Therefore, the former provides a better CFtr (75%) and overall fault coverage (83.3%) than the latter (62.5% and 80.6%, respectively). It provides similar fault coverage compared to the 14N-complexity March AZ2 algorithm, whose fault coverage is presented in Table 2. However, since its complexity is 1N lower than the March AZ2 algorithm, its MBIST operation may require a shorter completion time. Consequently, as proven by the simulation results obtained from the tests on both fault-free and fault-injected memories, the new March AZ algorithm, with 13N complexity, offers the best balance between memory testing time and fault coverage since it provides the best coverage of the targeted faults among all existing test algorithms with a complexity below 18N and produces a shorter test time than the March AZ2 algorithm.

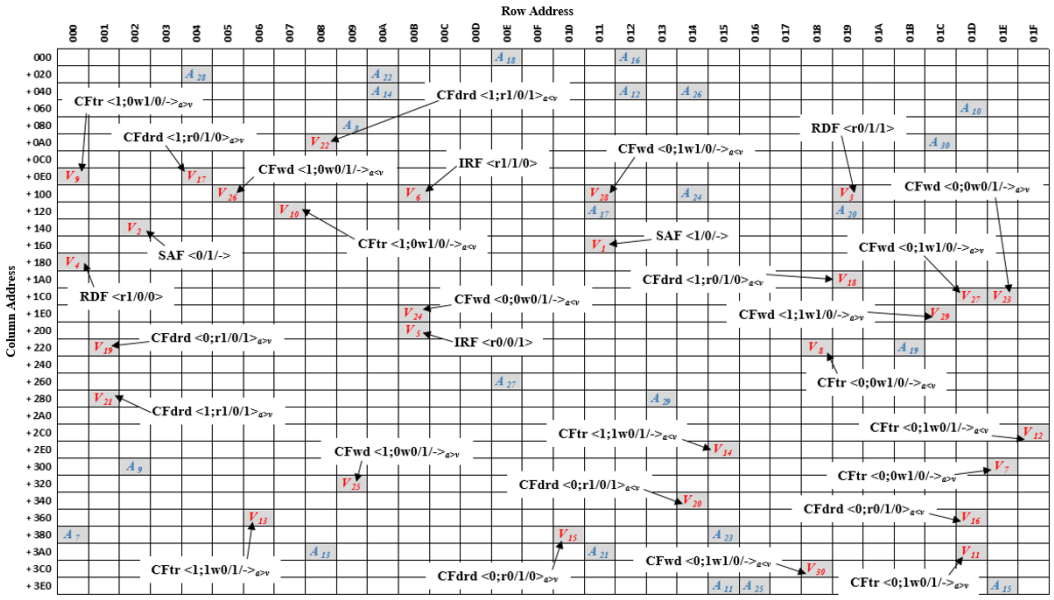

Figure 4. The distribution of the affected *v*-cells and the corresponding *a*-cells (for DCF) in the fault-injected memory model used for the simulation

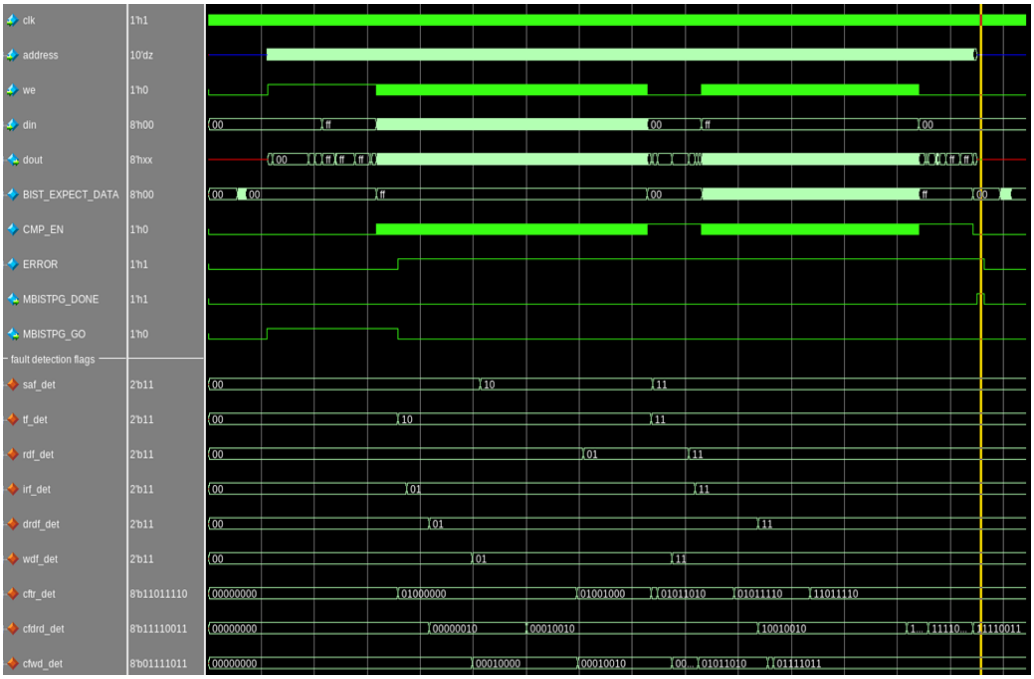

Figure 5. The waveform observed from the test on the fault-injected memory, using the March AZ as the UDA

Table 6. The March AZ algorithm's fault coverage derived from the simulation

Fault	Detection Flag	Observed Detection Flag Value	Derived Fault Coverage
SAF	saf_det	11_b	2 detected FPs out of 2 (100%)
TF	tf_det	11_b	2 detected FPs out of 2 (100%)
RDF	rdf_det	11_b	2 detected FPs out of 2 (100%)
IRF	irf_det	11_b	2 detected FPs out of 2 (100%)
DRDF	drdf_det	11_b	2 detected FPs out of 2 (100%)
WDF	wdf_det	11_b	2 detected FPs out of 2 (100%)
CFtr	cftr_det	11110110_b	6 detected FPs out of 8 (75%)
CFdrd	cfdrd_det	11001111_b	6 detected FPs out of 8 (75%)
CFwd	cfwd_det	11001111_b	6 detected FPs out of 8 (75%)
Overall fault coverage			30 detected FPs out of 36 (83.3%)

5. CONCLUSION

This article introduces the new march AZ algorithm, which enhances the CFtr and overall fault coverages offered by the existing March AZ1 algorithm while keeping the complexity at 13N. The March AZ1 algorithm was first analyzed to identify its weakness in detecting CFtr due to the poor positioning of a write operation. Hence, two test elements within its test sequence were reorganized by moving the identified w0 operation from one test element to another to solve the unnecessary CFtr recovery and improve CFtr detection without involving additional test operations. The newly created test sequence, the March AZ algorithm, was reanalyzed to ensure that CFtr coverage was improved without affecting other faults' detections. It then served as the test algorithm in the implemented MBIST controller, which was later used in simulations to conduct tests on two different memory models. The first test on a fault-free memory demonstrated its correct functionality, as no mismatch between the read and expected data was found during the simulation. Then, the second test conducted on a fault-injected memory validated that, with 13N complexity, it offers 83.3% of overall fault coverage, similar to the fault coverage provided by the March AZ2 algorithm with 14N complexity. Consequently, it offers the best balance between the test completion time and fault coverage among all test algorithms with a complexity lower than 18N since it provides the highest coverage of the intended faults with only 13N test complexity.

ACKNOWLEDGEMENTS

The authors would like to acknowledge Universiti Teknikal Malaysia Melaka (UTeM), Universiti Malaysia Perlis (UniMAP), and the Ministry of Higher Education Malaysia for their contribution, assistance, and support to this research under the research grant FRGS/1/2024/TK07/UTEM/02/17.

REFERENCES

[1] Nisha O. S. and Sivasankar K., "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.

[2] G. Prasad Acharya, M. Asha Rani, G. Ganesh 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, Apr. 2022, doi: 10.11591/ijeecs.v26.i1.pp96-104.

[3] D. Jamal and R. Veetil, "Efficient MBIST area and test time estimator using machine learning technique," in *2023 36th International Conference on VLSI Design and 2023 22nd International Conference on Embedded Systems (VLSID)*, Jan. 2023, pp. 223–228, doi: 10.1109/VLSID57277.2023.00054.

[4] P. Ramakrishna, T. Vamshika, and M. Swathi, "FPGA implementation of memory BISTs using single interface," *International Journal of Recent Technology and Engineering (IJRTE)*, vol. 9, no. 3, pp. 55–58, Sep. 2020, doi: 10.35940/ijrte.B3975.099320.

[5] R. Silveira, Q. Qureshi, and R. Zeli, "Flexible architecture of memory BISTs," in *2018 IEEE 19th Latin-American Test Symposium (LATS)*, Mar. 2018, pp. 1–6, doi: 10.1109/LATW.2018.8349666.

[6] 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*, Feb. 2012, pp. 700–704, doi: 10.1109/ISMS.2012.88.

[7] A. Z. Jidin, R. Hussin, W. F. Lee, and M. S. Mispan, "MBIST implementation and evaluation in FPGA based on low-complexity march algorithms," *Journal of Circuits, Systems and Computers*, vol. 33, no. 08, 2023, doi: 10.1142/S0218126624501524.

[8] I. G. Matri, Aishwaraya, N. Shreya, S. V. Siddamal, and S. V. Budihal, "A novel march XR algorithm, design, and test architecture for memories," in *Advances in Electrical and Computer Technologies*, Springer Nature Singapore, 2022, pp. 321–329, doi: 10.1007/978-981-19-1111-8_26.

[9] Y. Xiao, S. Lu, E. Wang, R. Zhu, and Z. Dai, "Test primitive: a straightforward method to decouple march," *arXiv Prepr.*, pp. 1–9, 2023, doi: 10.48550/arXiv.2309.03214.

[10] R. Wang, Z. Huang, G. Cai, and Z. Yu, "A built-in self-test circuit based on march FRDD algorithm for FinFET memory," in *Industrial Engineering and Applications: Proceedings of the 10th International Conference on Industrial Engineering and Applications (ICIEA 2023)*, Jul. 2023, doi: 10.3233/ATDE230069.

[11] V. S. Chakravarthi, *A practical approach to VLSI system on chip (SoC) design*. Cham: Springer International Publishing, 2022, doi: 10.1007/978-3-031-18363-8.

[12] 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)*, Aug. 2021, pp. 76–79, doi: 10.1109/RSM52397.2021.9511602.

[13] 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)*, May 2019, pp. 61–65, doi: 10.1109/RTEICT46194.2019.9016882.

[14] 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)*, 2022, pp. 78–84, doi: 10.1109/AIC55036.2022.9848869.

[15] S. B. Ghale and P. N., "Design and implementation of memory BIST for hybrid cache architecture," in *2021 6th International Conference on Communication and Electronics Systems (ICCES)*, 2021, pp. 26–31, doi: 10.1109/ICCES51350.2021.9489225.

[16] J. Kruthika, G. R. Nisha, R. Gayathri, and V. Jeyalakshmi, "SRAM memory built in self-test using march algorithm," in *2022 International Conference on Augmented Intelligence and Sustainable Systems (ICAISS)*, Nov. 2022, pp. 1288–1292, doi: 10.1109/ICAISS55157.2022.10010813.

[17] 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, Dec. 2022, doi: 10.3233/FI-222141.

[18] S. Martirosyan and G. Harutyunyan, "An efficient fault detection and diagnosis methodology for volatile and non-volatile memories," in *2019 Computer Science and Information Technologies (CSIT)*, Sep. 2019, pp. 47–51, doi: 10.1109/CSITechnol.2019.8895189.

[19] S. F. Abd Gani, M. F. Miskon, R. A. Hamzah, M. S. Hamid, A. F. Kadmin, and A. I. Herman, "Stereo matching algorithm using deep learning and edge-preserving filter for machine vision," *Bulletin of Electrical Engineering and Informatics*, vol. 13, no. 3, pp. 1685–1693, Jun. 2024, doi: 10.11591/eei.v13i3.5708.

[20] 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)*, 2005, pp. 53–59, doi: 10.1109/VTS.2005.56.

[21] S. Hamdioui, "Testing embedded memories: a survey," in *Mathematical and Engineering Methods in Computer Science*, A. Kučera, T. A. Henzinger, J. Nešetřil, T. Vojnar, and D. Antoš, Eds. Berlin: Springer Berlin Heidelberg, 2013, pp. 32–42, doi: 10.1007/978-3-642-36046-6_4.

[22] 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, 2023, doi: 10.1109/TCAD.2022.3229281.

[23] 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)*, 2002, pp. 256–261, doi: 10.1109/OLT.2002.1030228.

[24] 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)*, Nov. 2022, pp. 578–582, doi: 10.1109/APCCAS55924.2022.10090328.

[25] Z. Zhi-chao, H. Li-gang, and W. Wu-Chen, "SRAM BIST design based on march C+ algorithm," *Mod. Electron. Tech*, vol. 34, no. 10, pp. 149–151, 2011.

[26] A. Z. Jidin, R. Hussin, L. W. Fook, and M. S. Mispan, "An automation program for march algorithm fault detection analysis," in *2021 IEEE Asia Pacific Conference on Circuit and Systems (APCCAS)*, 2021, pp. 149–152, doi: 10.1109/APCCAS51387.2021.9687806.

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 a field-programmable gate array (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 design for testability (DFT), very large-scale integration (VLSI), and field-programmable gate array (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 associate professor 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.