A review paper on memory fault models and test algorithms

Aiman Zakwan Jidin[1], Razaidi Hussin[2], Lee Weng Fook[3], Mohd Syafiq Mispan[4]
[1,2]Faculty of Electronics Engineering Technology, Universiti Malaysia Perlis, Malaysia
[1,4]Faculty of Electrical and Electronics Engineering Technology, Universiti Teknikal Malaysia Melaka, Malaysia
[3]Emerald Systems Design Center, Malaysia

Article Info

Article history:

Received Apr 20, 2021
Revised Aug 19, 2021
Accepted Oct 9, 2021

Keywords:

Built-in self-test
Design for testability
March test algorithm
Memory fault model
Random access memory

ABSTRACT

Testing embedded memories in a chip can be very challenging due to their high-density nature and manufactured using very deep submicron (VDSM) technologies. In this review paper, functional fault models which may exist in the memory are described, in terms of their definition and detection requirement. Several memory testing algorithms that are used in memory built-in self-test (BIST) are discussed, in terms of test operation sequences, fault detection ability, and also test complexity. From the studies, it shows that tests with 22 N of complexity such as March SS and March AB are needed to detect all static unlinked or simple faults within the memory cells. The N in the algorithm complexity refers to Nx*Ny*Nz whereby Nx represents the number of rows, Ny represents the number of columns and Nz represents the number of banks. This paper also looks into optimization and further improvement that can be achieved on existing March test algorithms to increase the fault coverage or to reduce the test complexity.

Corresponding Author:

Razaidi Hussin
Faculty of Electronic Engineering Technology
Universiti Malaysia Perlis
02600 Arau, Perlis, Malaysia
Email: shidee@unimap.edu.my

1. INTRODUCTION

Design for testability (DFT) is a design technique used by IC designers and manufacturers to enhance the controllability and observability of the device under test (DUT), and subsequently improving the fault coverage to >90% for large design [1], [2]. In the IC design flow for ASIC or SOC, during the DFT stage, auxiliary circuitries are added to the design to allow testability of the device after fabrication. DFT concerns both logic circuits as well as memories inside a chip. In recent years, the trend shows SOCs are memory-dominant chips, where embedded memories occupied up to 90% of the total chip area. Therefore, the quality of the memories is the main factor in having a good manufacturing yield [3]-[9]. Testing an embedded memory like a DRAM and SRAM can be very challenging, due to its extreme density [10]. Furthermore, most IC chips are manufactured using very deep submicron (VDSM), more defects are occurring during the chip fabrication process resulting in complex chip testing [9], [11]-[13]. Memories in a chip can be tested using the memory BIST technique, which goal is to ensure the memories are free from any defect resulting in higher yields. Memory BIST technique also allows the reduction in overall testing cost since no external tester is needed and testing can be performed in parallel thus shortening test time [14]-[21].

Memory BIST works on testing the memory under test (MUT) by applying a sequence of test operations or a test algorithm. In order to efficiently test a memory, there are two main criteria: fault coverage and test complexity. The former determines the fault detection ability, whereby the higher the fault coverage, the better the fault detection. It can be calculated in:

$$Fault\ coverage = \frac{No\ of\ detectable\ faults}{No\ of\ total\ possible\ faults}$$

While the latter determines the length or duration it will take to complete the memory testing. The test complexity is usually written as $O(xN^m)$, where x is the number of test operations required and N is the size of the memory. The test complexity is said to be linear if m is equal to 1, otherwise, it is non-linear. Table 1 shows the time it takes to complete the memory testing, based on the memory size and the test complexity [22], [23]. In the case of tests with linear test complexity, there are x read or write operations that need to be performed on each of N memory cells. Hence, to determine the test length, it will take a minimum of xN clock cycles to complete the test.

Table 1. Memory testing as a function of test complexity and memory size [23]

Size (N)	Complexity			
	N	$N \log N$	$N^{3/2}$	N^2
1K	0.0001s	0.001s	0.0033s	0.105s
16K	0.00016s	0.0224s	0.21s	27s
64K	0.0064s	0.1s	1.678s	7.17m
256K	0.0256s	0.46s	13.4s	1.9h
1M	0.102s	2.04s	1.83m	1.27d
16M	1.64s	39.36s	1.9h	326d
64M	6.56s	2.843m	15.25h	14.3y
256M	26.24s	12.25m	5.1d	229y
1G	1.75m	52.48m	40.8d	3659y

In this paper, the functional fault models are discussed. As the number of possible faults can be unlimited [24], this paper focuses on the commonly used fault models in the literature. This paper also discusses different memory test algorithms with a focus on March series test algorithms as they are commonly used in the industry due to their simplicity, linear-time test complexity, and low area overhead [25]-[27]. The notations used for describing the fault models and the test algorithms are also presented.

2. MEMORY FUNCTIONAL FAULT MODELS

Three commonly used terms in DFT: defect, fault, and error. According to [28], a defect is an unintended difference between the circuit design and the implemented hardware, which occurs during manufacturing or the use of the devices. A fault is the representation of a defect in the abstracted function level. For example, when there is a short-circuit between a net and power supply V_{DD} in the device, this defect is represented as a stuck-at 1 fault in the abstracted function level. As a result, a defective device will produce incorrect outputs called the error. In a digital system, a comparison of the logical behavior of the tested system with the behavior of the good system is often used when conducting the test. Therefore, there is a need for all physical failures during manufacturing to be modeled as logical faults [5].

Figure 1 shows the general architecture of a random access memory (RAM) [29], [30]. It normally consists of memory cells, an address decoder, a write driver, and a sense amplifier. A fault may occur in any one of these blocks. As multiple faults may exist in memory simultaneously, they can influence or interact with each other. This scenario is referred to as the linked fault model. Fault models that are linked can be the same or of different types. They can also work in such a manner that one fault masks the behavior of another fault [31], [32]. This paper focuses on reviewing unlinked faults whereby multiple faults do not interact with each other.

The functional fault models of memory can be a *static* fault or a *dynamic* fault. A *static* fault model can be sensitized by only one operation (read or write), while a *dynamic* fault model requires more than one operation to be sensitized. Therefore, the number of dynamic fault sets can be unlimited theoretically [33]. In this review paper, only the static faults are discussed. The examples of a static fault:

a. Stuck-at fault (SAF)-the value of a cell is forced to a logic 0 (SAF0) or logic 1 (SAF1), regardless of the value of the logic.
b. Transition fault (TF)-the memory cell fails to transition from a logic 0 to logic 1, or from a logic 1 to logic 0.
c. Read destructive fault (RDF)-the content of a memory cell can be changed during a reading process.
d. Deceptive read destructive fault (DRDF)-the content of a memory cell can be changed during a reading process, but the read output has the correct value.
e. Write disturb fault (WDF)-the content of a memory cell is changed when performing a non-transition write operation.
f. Incorrect read fault (IRF)-a read on a memory cell returns an incorrect value, while its state remains unchanged.

g. Coupling faults (CF)-a write operation in one cell (also referred to as the aggressor cell) influences the value in another cell (the victim cell). There are six types of CFs:
– State coupling fault (CFst)-a victim cell is forced to a logic 0 or logic 1 when the aggressor cell is in a given state
– Idempotent coupling fault (CFid)-a change of value in the aggressor cell will unexpectedly change the value in the victim cell
– Inverse coupling fault (CFin)-the value of the victim cell will complement the value of the aggressor cell
– Transition coupling fault (CFtr)-a victim cell fails to transition from low to high (or high to low) when the aggressor cell is in a given state.
– Read destructive coupling fault (CFrd)-the content of a victim cell can be changed during a reading process when the aggressor is in a given state.
– Deceptive read destructive coupling fault (CFdrd)-the content of a victim cell can be changed during a reading process when the aggressor is in a given state, but the read output has the correct value.
– Write destructive coupling fault (CFwd)-the content of a victim cell is changed when performing a non-transition write operation when the aggressor cell is in a given state.

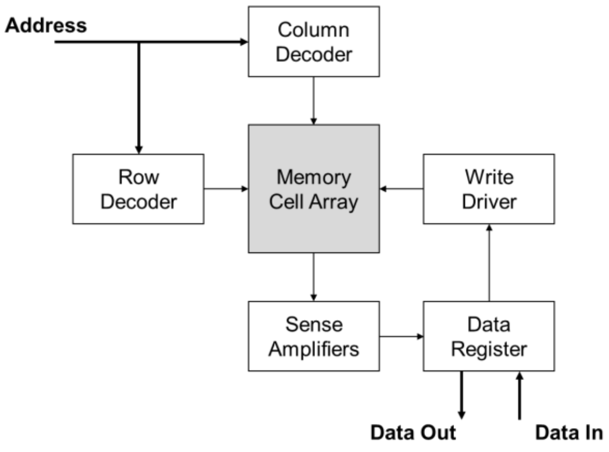

Figure 1. RAM general architecture

As device technologies move towards very deep submicron, faults like DRDF, WDF, CFdrd, and CFwd are becoming more relevant in today's embedded memories compare to conventional faults like CFid, CFin, and CFst [24]. Table 2 summarizes all the mentioned fault models, in terms of their fault primitives (FPs) and the detection requirement. A fault primitive (FP) is a combination of sensitizing operation sequence (*S*), faulty behavior (*F*), and the output of the read operation (*R*), for each of these fault models. It is denoted as <*S* / *F* / *R*> for single-cell faults (SCFs) or <S_a;S_v / *F* / *R*> for double-cell faults (DCFs), where *a* and *v* stand for the aggressor and the victim cells, respectively. The set of *S* is defined as $S \in \{0, 1, X, w0, w1, r0, r1\}$, where X indicates that the value of S doesn't have any importance. The faulty behavior *F* can be either: $F \in \{0, 1, ?\}$, where '?' denotes an undefined logic. While the read operation return value *R* can take one of the following values: $R \in \{0, 1, ?, -\}$, where '?' denotes an undefined logic and '–' is used when the output data is not applicable [31], [34]-[36].

It can also be seen from Table 2 that, the detection of double-cell faults can be very complex since it involves more than one operation for the sensitization and detection, and it also requires the test to be performed in both address order, so that it would cover the cases where the address of the aggressor cell is lower than the victim cell (denotes as $a<v$) and vice-versa (denotes as $a>v$) [23], [37]. Moreover, in many cases, the sensitization of the victim cell must consider all possible values of the aggressor cell (logic 0 and 1). Therefore, it can be said that there are 8 possible cases for each of the CFtr, CFrd, CFdrd, and CFwd [25], [38], [39].

Apart from the fault models previously mentioned, which may occur within the memory cells, other types of faults can appear in other parts of the memory block. The faults that could happen in the address decoder are referred to as the address decoder faults (ADF). There are four possible scenarios for ADF:
– No memory cell can be accessed by certain addresses
– Multiple cells can be simultaneously accessed by a certain address
– A certain memory cell can be accessed by multiple addresses
– A certain memory cell is not accessible by any address

Table 2. Common memory fault models with their fault primitives and detection requirement [32], [40]

Fault Type	SCF/DCF	FP	Detection Requirement
SAF	SCF	<0 / *1* / -> *<1 / 0 / ->*	To detect SAF*x*, an opposite value *x'* is written to the cells. The read will return the *x* value in the case of faulty. The test must be performed for both *x*=0 and *x*=1.
TF	SCF	<0w1 / 0 / -> <1w0 / 1 / ->	The contents of the cells must be set to *x* first. Then, *x'* value is written to the cells, followed by a read operation. The read will return the *x* value in the case of faulty. The test must be performed for both *x*=0 and *x*=1.
CFst	DCF	<0;0 / 1 / -> <0;1 / 0 / -> <1;0 / 1 / -> <1;1 / 0 / ->	For a complete CFst/CFid/CFin detection, the test algorithm must contain the following sequence [32]: ⇑(rx, …, wx'); ⇑(rx', …, wx); ⇓(rx, …, wx'); ⇓(rx', …, wx)
CFid	DCF	<0w1;0 / 1 / -> <0w1;1 / 0 / -> <1w0;0 / 1 / -> <1w0;1 / 0 / ->	For a complete CFst/CFid/CFin detection, the test algorithm must contain the following sequence [32]: ⇑(rx, …, wx'); ⇑(rx', …, wx); ⇓(rx, …, wx'); ⇓(rx', …, wx)
CFin	DCF	<0;*X* / 1 / -> <1;*X* / 0 / ->	For a complete CFst/CFid/CFin detection, the test algorithm must contain the following sequence [32]: ⇑(rx, …, wx'); ⇑(rx', …, wx); ⇓(rx, …, wx'); ⇓(rx', …, wx)
RDF	SCF	<0r0 / 1 / 1> <1r1 / 0 / 0>	The contents of the cells must be set to *x* first, followed immediately by a read operation. The read will return the *x'* value in the case of faulty. The test must be performed for both *x*=0 and *x*=1.
DRDF	SCF	<0r0 / 1 / 0> <1r1 / 0 / 1>	The contents of the cells must be set to *x* first, followed immediately by consecutive double read operations. The second read will return the *x'* value in the case of faulty. The test must be performed for both *x*=0 and *x*=1.
WDF	SCF	<0w0 / 1 / -> <1w1 / 0 / ->	The contents of the cells must be set to *x* first. Then, another write *x* value operation is performed (non-transition), followed by a read operation. The read will return the *x'* value in the case of faulty. The test must be performed for both *x*=0 and *x*=1.
IDF	SCF	<0r0 / 0 / 1> <1r1 / 1 / 0>	The contents of the cells must be set to *x* first followed by a read operation. The read will return the *x'* value in the case of faulty. The test must be performed for both *x*=0 and *x*=1.
CFtr	DCF	<0;0w1 / 0 / -> <0;1w0 / 1 / -> <1;0w1 / 0 / -> <1;1w0 / 1 / ->	The contents of the victim cell and the aggressor cell must be set to *x* and *y* first, where *y*=0 or 1. Then, *x'* value is written to the victim cell, followed by a read operation. The read will return the *x* value in the case of faulty. The test must be performed for both *x*=0 and *x*=1, and for both *a*<*v* and *a*>*v* cases.
CFrd	DCF	<0;0r0 / 1 / 1> <0;1r1 / 0 / 0> <1;0r0 / 1 / 1> <1;1r1 / 0 / 0>	The contents of the victim cell and the aggressor cell must be set to *x* and *y* first, where *y*=0 or 1, followed immediately by a read operation on the victim cell. The read will return the *x'* value in the case of faulty. The test must be performed for both *x*=0 and *x*=1, and for both *a*<*v* and *a*>*v* cases.
CFdrd	DCF	<0;0r0 / 1 / 0> <0;1r1 / 0 / 1> <1;0r0 / 1 / 0> <1;1r1 / 0 / 1>	The contents of the victim cell and the aggressor cell must be set to *x* and *y* first, where *y*=0 or 1, followed immediately by a double read operation on the victim cell. The second read will return the *x'* value in the case of faulty. The test must be performed for both *x*=0 and *x*=1, and for both *a*<*v* and *a*>*v* cases.
CFwd	DCF	<0;0w0 / 1 / -> <0;1w1 / 0 / -> <1;0w0 / 1 / -> <1;1w1 / 0 / ->	The contents of the victim cell and the aggressor cell must be set to *x* and *y* first, where *y*=0 or 1. Then, another write *x* value operation is performed to the victim cell, followed by a read operation. The read will return the *x'* value in the case of faulty. The test must be performed for both *x*=0 and *x*=1, and for both *a*<*v* and *a*>*v* cases.

According to Siemens [40], ADFs can be detected by using any March style test, and thus, no additional detection requirement is needed. Another type of fault model is the stuck-open faults (SOF), which can occur at the sense amplifier block. If the sense amplifier does not contain a data latch, SOF can be treated as a SAF. Otherwise, the following sequence must be applied for the detection [40]:

– The cell under test must be storing a logic *x* (where *x* can be either 0 or 1)
– An inverse logic *x'* is written to the cell
– The value stored in the cell is read and compared with the expected value

The mention detection requirement should be covered by the test for detecting the transition faults (TF). There are other types of fault models that can occur in the memory but are rarely discussed in the previous research works. For example, the neighborhood pattern-sensitive faults (NPSF) that are different from other coupling faults since they involve several aggressor cells and one victim cell [41]. There are several types of faults under the NPSF model, such as the single-port bitline coupling faults, the access transistor leakage current fault, the Static NPSF (SNPSF), the passive NPSF (PNPSF), and the active NPSF (ANPSF) [40], [42]. Plus, other types of fault models are the write recovery fault (WRF), read enable fault (REF), and memory select fault (MSF) [32], [40], [41].

3. MEMORY TESTING ALGORITHMS

Table 3 describes the notation used to define the test operation sequences of a test algorithm [43], [44]. The notation indicates the number of elements in a test, the test operations, and the address order. An element in the test algorithm describes a series of operations to be performed on each memory cell, according to the set address order (ascending or descending), before proceeding to the next test element. For example, in the case of MATS++ algorithm [42], [45]:

⇕(w0); ⇑(r0, w1); ⇓(r1, w0, r0);

There are 3 elements, each of them is separated by a semicolon. From this notation, it can be derived that:

- In the first element, all memory cells will be set to 0s, regardless of the order of the address
- In the second element, a read operation (expecting logic 0) and then a write 1 operation are performed on cell 0. The same procedure is repeated for cells 1, 2, …, N-1.
- In the third element, a read operation (expecting logic 1), a write 0 operation, and another read operation (expecting logic 0) are performed on cell N-1. The same procedure is repeated for cells N-2, N-3, …, 1, 0.
- The number of test operations required on each cell is 6. Hence, its test complexity is 6*N*.

Table 3. The notations in the memory testing algorithms [18], [46]-[48]

Symbol	Description
↑ or ⇑	address sequence changes in ascending order
↓ or ⇓	address sequence changes in descending order
↕ or ⇕	address sequence can change either way
R0	read operation (reading a 0 from a cell)
R1	read operation (reading a 1 from a cell)
W0	write operation (writing a 0 to a cell)
W1	write operation (writing a 1 to a cell)

There are various algorithms that can be used in testing the embedded memories. Galloping pattern and walking pattern tests such as GALPAT and WALPAT have quadratic test complexities ($O(4N^2)$ and $O(4N^{1.5})$, respectively), and thus, it will take a very long time to complete the test [23]. Based on Table 1, it would require at least 229 years to complete a GALPAT test on a 256 Mbit memory, and 5.1 days for the case of a WALPAT test. As such, a test with linear complexity is preferred. Classical test algorithms like the Zero-One algorithm and Checkerboard algorithm are very simple in terms of test complexity (4*N*). For Zero-One test algorithm:

⇓ (w0); ⇑ (r0); ⇑ (w1); ⇓ (r1);

It only requires 4 test operations on each memory cells. This results in a very low fault coverage since it can only be used to detect the SAF and half of the TF [13]. The Checkerboard test presented minor improvement in the detection of TFs and some CFs, instead of writing 0s or 1s to all cells, the test pattern used is alternated between 0 and 1 for consecutive cells.

March-series test algorithms are widely used in the industry, due to their simplicity yet having a good fault coverage. MATS++ algorithm, which was previously mentioned above, is an example of a March test algorithm. It overcomes the weakness in Zero-One algorithm by detecting all the TFs. The detection of coupling faults is very poor. There are also test algorithms like March X (6 *N*) [49] and its extension, March Y (8 *N*), which were developed to enhance its detection of CFin and ADF [32]. Research in [7] proposed optimization in March Y test algorithm, with the second and third elements of the test algorithm executed in parallel, and thus, the test complexity is reduced by 3*N*. However, no improvement on the fault coverage was made and some modifications need to be done on the SRAM architecture to be testable by using the proposed test algorithm.

This weakness of having poor coverage on coupling faults was overcome by the development of March C algorithm, with 11*N* test complexity [17], [50]. By following the detection requirement mentioned in Table 2 in Section 2, it can be used to detect all the conventional CFs like CFid, CFin, and CFst [32], [51], [52]. However, research in [44] proved that a read operation seems to be unnecessary and thus, can be removed. Hence, March C-algorithm was proposed, with the reduction of 1*N* test complexity while maintaining the same fault coverage. March LR algorithm with 14 *N* test complexity was also proposed to detect several linked faults [53]. The authors in [14] claim the algorithms can detect all simple faults and coupling faults. However, by observing the test operation sequences in the algorithm and comparing them with the requirement described in Table 2, this algorithm is incapable of detecting several faults like DRDF, WDF, CFdrd, and CFwd (e.g. neither non-transition write operation nor consecutive double read operation found in the algorithm).

Many researchers worked on improving the March test algorithms to detect new faults introduced by VDSM devices, such as DRDF, WDF, IRF, CFtr, CFdrd, CFwd, and CFir. March CL (12 *N*) was developed [54], [55] and can detect half of the DRDF (only when the value of both coupling cells are 1s). This is achieved by having a consecutive double read operation of value 1 in the algorithm. Meanwhile, March SR (14 *N*) was proposed [56], [57] to have a complete detection of DRDF and some CFdrds, by adding consecutive double read operations for both logic 0 and logic 1 in the algorithm. Both March CL and March SR algorithms are unable to detect CFwd.

March RAW algorithm was proposed [39] and able to detect all static and several dynamic faults, with 26 *N* of test complexity. As can be seen from its test sequence given in Table 4, the detection requirement for all targeted faults listed in Table 2 is achieved. In addition, the author also proposed March RAW1, with 13 *N* test complexity and targeted only for single-cell faults. However, the optimization was made by March AB [24] and March SS [38]. By considering all faults coverage conditions, both works managed to identify and remove the redundancy of test operations presented in March RAW, and thus, reducing the test complexity from 26*N* to 22*N*.

Table 4 summarizes the test complexity and the test operation sequences of the existing and previously proposed March test algorithms. While Table 5 and Table 6 summarize their detection of single-cell faults and double-cell faults, respectively.

Table 4. Summary of March test algorithms complexity and test operation sequence

Test Algorithm	Test Complexity	Test Operation Sequence
Zero-One	4 *N*	⇕(w0); ⇕(r0); ⇕(w1); ⇕(r1)
MATS++	6 *N*	⇕(w0); ⇑(r0, w1); ⇓(r1, w0, r0)
March X	6 *N*	⇕(w0); ⇑(r0, w1); ⇓(r1, w0); ⇕(r0)
March Y	8 *N*	⇕(w0); ⇑(r0, w1, r1); ⇓(r1, w0, r0); ⇕(r0)
March C	11 *N*	⇕(w0); ⇑(r0, w1); ⇑(r1, w0); ⇕(r0); ⇓(r0, w1); ⇓(r1, w0); ⇕(r0)
March C-	10 *N*	⇕(w0); ⇑(r0, w1); ⇑(r1, w0); ⇓(r0, w1); ⇓(r1, w0); ⇕(r0)
March CL	12 *N*	⇕(w0); ⇑(r0, w1); ⇑(r1, r1, w0); ⇓(r, w1, r1); ⇓(r1, w0); ⇕(r0)
March SR	14 *N*	⇕(w0); ⇑(r0, w1, r1, w0); ⇑(r0, r0); ⇑(w1); ⇓(r1, w0, r0, w1); ⇓(r1, w1)
March LR	14 *N*	⇕(w0); ⇓(r0, w1); ⇑(r1, w0, r0, w1); ⇑(r1, w0); ⇑(r0, w1, r1, w0); ⇑(r0)
March RAW	26 *N*	⇕(w0); ⇑(r0, w0, r0, r0, w1, r1); ⇑(r1, w1, r1, r1, w0, r0); ⇓(r0, w0, r0, r0, w1, r1); ⇓(r1, w1, r1, r1, w0, r0); ⇕(r0)
March RAW 1	13 *N*	⇕(w0); ⇕(w0, r0); ⇕(r0); ⇕(w1, r1); ⇕(r1); ⇕(w1, r1); ⇕(r1); ⇕(w0, r0); ⇕(r0)
March SS	22 *N*	⇕(w0); ⇑(r0, r0, w0, r0, w1); ⇑(r1, r1, w1, r1, w0); ⇓(r0, r0, w0, r0, w1); ⇓(r1, r1, w1, r1, w0); ⇕(r0);
March AB	22 *N*	⇕(w0); ⇓ (r0, w1, r1, w1, r1); ⇓ (r1, w0, r0, w0, r0); ⇑ (r0, w1, r1, w1, r1); ⇑ (r1, w0, r0, w0, r0); ⇕(r0);

Table 5. Summary of March test algorithms single-cell fault coverage

Test Algorithm	SAF	TF	RDF	DRDF	WDF	IF
Zero-One	F	50%				
MATS++	F	F	F	-	-	-
March X	F	F	F	-	-	-
March Y	F	F	F	-	-	-
March C	F	F	F	-	-	-
March C-	F	F	F	-	-	-
March CL	F	F	F	50%	-	F
March SR	F	F	F	F	-	F
March RAW	F	F	F	F	F	F
March LR	F	F	F	-	-	F
March RAW1	F	F	F	F	F	F
March SS	F	F	F	F	F	F
March AB	F	F	F	F	F	F

Several test algorithms were developed to detect NPSFs but they came with very high test complexities. As mentioned in [41], a 68 *N*-March test algorithm can be used for ANPSF and PNPSF, while it requires the use of a 96 *N*-March test to detect all ANPSF, PNPSF, and SNPSF. Meanwhile, several commercial test algorithms are also made available by the memory BIST tools provider such as Mentor graphics. MarchCHKBcil algorithm (44 *N*) can be applied to detect the write recovery fault, the single-port bitline coupling faults, and the access transistor leakage current fault, whereas MarchCHKBvcd algorithm (68 *N*) offers extra coverage on the detection of read enable fault and memory select fault [40].

Table 6. Summary of March test algorithms double-cell fault coverage

Test Algorithm	CFst	CFid	CFin	CFtr	CFdrd	CFwd
March X	-	-	F	50%	-	-
March Y	-	-	F	50%	50%	-
March C	F	F	F	F	-	-
March C-	F	F	F	F	-	-
March CL	F	F	F	F	50%	-
March SR	F	F	F	F	50%	-
March LR	F	F	F	F	-	-
March RAW	F	F	F	F	F	F
March SS	F	F	F	F	F	F
March AB	F	F	F	F	F	F

4. PREVIOUS WORKS ON IMPROVING THE MEMORY TESTING ALGORITHMS

By observing these test algorithms fault coverage provided in Table 5 and Table 6, it shows that those with high test complexities (22 *N* and above) like March SS, March AB, and March RAW have full coverage on the unlinked faults in a RAM, whereas those with low test complexities (below 10 *N*) like MATS++, March X, and March Y algorithms have very poor fault coverage. For those with mid-range test complexities (between 10 *N* and 14 *N*) have poor coverage on faults like DRDF, WDF, CFdrd, and CFwd. Based on the review, several works were done to improve the existing March algorithms.

For example, the reduction of the March C- test complexity was proposed by [11], which managed to attain an 8 *N* test complexity. This was achieved by dividing the test operations into two subgroups which are executed in parallel:

M1: ⇑(w0); ⇑(r0, w1); ⇑(r1); ⇓(w0); ⇓(r0, w1); ⇓(r1)

M2: ⇑(w1); ⇑(r1, w0); ⇑(r0); ⇓(w1); ⇓(r1, w0); ⇓(r0)

By observing the proposed algorithm, it can be seen that M2 is exactly the complement of M1, and thus, only one test bit generator is adequate for both subgroups, where an inverter is added to invert the test bit for M2. Meanwhile, research in [10] proposes the modification in the memory BIST design to fusion three different algorithms (MATS, March X, and March C) in one design. The proposed technique was proven inefficient in improving the fault detection of the memory BIST, as it only allows the system to select one test algorithm to be used (among three options available) when the circuit is operating, by utilizing a multiplexer.

An improvement of March C- algorithm was proposed in [58], by introducing a new March C+ algorithm, with the following test sequences: ⇕(w0); ⇑(r0, w1, r1); ⇑(r1, w0, r0); ⇓(r0, w1, r1); ⇓(r1, w0, r0); ⇕(r0). In this proposed new algorithm, a read operation is added at the end of each test element 2 to test element 5, and, as the consequence, it has full coverage on DRDF and CFdrd, by having double read operations. Since 4 read operations are added to the test sequences, the test complexity increased from 10 *N* to 14 *N*. Furthermore, the March C+ algorithm was extended by adding more test operations, to become a new algorithm with 22 *N* test complexity as shown in: ⇕(w0); ⇑(w0, r0, w1, w1, r1); ⇑(w1, r1, w0, w0, r0); ⇓(r0, w0, w1, w1, r1); ⇓(r1, w1, w0, w0, r0); ⇕(r0) [59]. Hence, just like March SS and March AB, this algorithm has full coverage on all unlinked faults in a RAM. Since it is named March Y by the author, it will be referred to as March Y2 in this review paper, to avoid confusion with March Y algorithm previously mentioned in section 3 [49].

An improvement was proposed in [25] by developing an automation program that can optimize the test operation of the existing March test algorithms, by having a set of test sequence generation rule schemes. This research managed proposes March CL-1 algorithm (⇕(w0); ⇑(r0, w0); ⇑(r0); ⇑(r0, w1); ⇓(r1, w1); ⇕(r1); ⇓(r1, w0); ⇕(r0)) and March-SR1 algorithm (⇕(w0); ⇑(r0, w0, r0, w1); ⇑(r1, r1); ⇑(w1); ⇓(r1, w0, r0, w0); ⇓(r0, w0)), as the results of March CL and March SR algorithm optimization. In addition, it also improves the fault detection of both algorithms in CFtr, CFdrd, and also CFwd, and increased their fault coverages, while maintaining the same test complexity. Moreover, the author also claims that a 100% fault coverage can be attained by adding another 4N test complexity, but this claim is still unproven by any work.

Table 7 summarizes the fault coverage achieved by the improved March algorithm. In most cases, additional test operations need to be added to the algorithms test sequences to improve their fault coverages, like in the case of March C+ and March Y2 algorithms. This is the trade-off to be considered since the enhancement of the fault coverage will certainly increase the test duration and thus, the overall test cost. In contrast, some research focus only on reducing the test complexity of the existing algorithm and does not offer any improvement on the fault coverage [7], [11]. While some research work like the one proposed by

[10] offers neither the improvement of the fault coverage nor the reduction of the test complexity, but it only proposed a customizable memory BIST implementation.

The research proposed in [25] managed to optimized the March SR algorithm to improve its fault coverage while maintaining the same test complexity. However, some weaknesses were identified in this research. Firstly, it focused only on improving the detection of SCFs. Secondly, the proposed optimization technique only works on several March algorithms e.g. March SR and March CL. Thirdly, the proposed technique does not work on removing any redundancies in the test operation sequences. Therefore, a research gap is identified, where the existing March algorithms can be further optimized to increased their fault coverage, by overcoming these three weaknesses. This will involve the change of the address order for certain test elements, rearrangement of several test operations in the sequences, and also restructuring several test elements, while still maintaining the same test complexity.

Besides, this review also identified the lack of research work to optimize the complexity of test algorithms on detecting faults like NPSFs, write recovery fault, and read enable fault, as the detection of these types of faults normally requires the testing algorithms with high test complexities. This will allow the reduction of the test time and thus, the test cost while maintaining the test quality. In addition, more research works could be done in the future to cover the test of different types of memories such as DRAM and Flash, as well as the emerging memory technologies of in-memory computing, which are very popular for Internet-of-Thing applications [60], [61].

Table 7. Fault coverage summary of the previously improved March algorithm

Test Algorithm	Test Complexity	SAF	TF	RDF	DRDF	WDF	IF	CFtr	CFdrd	CFwd
Fusion of MATS, March X, and March C [7]	Depend on the selected test algorithm during execution									
Modified March C- [11]	8 *N*	F	F	F	-	-	-	F	-	-
Improved March CL [25]	12 *N*	F	F	F	F	F	F	50%	50%	50%
Improved March SR [25]	14 *N*	F	F	F	F	F	F	50%	62.5%	50%
March C+ [58]	14 *N*	F	F	F	F	-	F	F	F	-
March Y2 [59]	22 *N*	F	F	F	F	F	F	F	F	F

5. CONCLUSION

This paper presented the review on functional fault models that commonly occur in memories, especially for RAMs. Each can be denoted by their respective FPs which indicate their sensitizers and the value of the faulty output, from which the detection requirement for each of these faults was derived, as shown in Table 2. The Memory BIST technique is widely used to test embedded memories on a chip, where an efficient test algorithm is necessary to ensure that the test can be performed with high quality and within a reasonable test length, to ensure a very good chip manufacturing yield and also maintain a reasonable overall testing costs. This paper has discussed several March test algorithms and some of their modifications proposed by previous researches, each of which offers different fault coverage and test complexities. The analysis shows that in most cases, high fault coverage test algorithms require more test operations, and thus, increasing the test complexity and the test cost. Based on the review, a minimum of 22 *N* test complexity is needed to detect all unlinked faults in RAM. However, an optimization can be done on the existing algorithms with lower test complexities to improve their fault coverages, especially on the DCFs, while maintaining the same test complexity. This can be achieved by removing test operations redundancies and rearranging test operations in the sequences.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the Faculty of Electronic Engineering Technology, Universiti Malaysia Perlis (UniMAP), Universiti Teknikal Malaysia Melaka (UTeM), and Ministry of Higher Education Malaysia, for their contribution, support to this research, and financial assistance under the SLAB scheme. The authors also acknowledge Nor Azura Zakaria for her assistance and knowledge sharing which indirectly contribute to this research.

REFERENCES

[1] V. S. Chakravarthi, "A practical approach to VLSI System on Chip (SoC) design : a comprehensive guide," Bangalore: Springer, p. 312, 2019, doi: 10.1007/978-3-030-23049-4.
[2] L. Y. Ungar, “Design for Testability (DFT) to Overcome Functional Board Test Complexities in Manufacturing

Test," *Proceedings IPC APEX*, 2017.
[3] U. Schlichtmann, "Tomorrows high-quality SoCs require high-quality embedded memories today," *Proceedings International Symposium on Quality Electronic Design*, 2002, pp. 225-, doi: 10.1109/ISQED.2002.996735.
[4] E. J. Marinissen, B. Prince, D. Keltel-Schulz and Y. Zorian, "Challenges in embedded memory design and test," *Design, Automation and Test in Europe*, vol. 2, pp. 722-727, 2005, doi: 10.1109/DATE.2005.92.
[5] I. Mrozek, "Multi-run memory tests for pattern sensitive faults," *2019th ed. Bialystok*, Pl: Springer, 2018.
[6] G. Harutyunyan, S. Shoukourian, V. Vardanian and Y. Zorian, "A New Method for March Test Algorithm Generation and Its Application for Fault Detection in RAMs," in I*EEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 6, pp. 941-949, June 2012, doi: 10.1109/TCAD.2012.2184107.
[7] O. S. Nisha and K. Sivasankar, "Architecture for an efficient MBIST using modified march-y algorithms to achieve optimized communication delay and computational speed," *International Journal of Pervasive Computing and Communications*, vol. 17, no. 1, pp. 135-147, 2021, doi: 10.1108/IJPCC-05-2020-0032.
[8] R. K. Sharma and A. Sood, "Modeling and Simulation of Multi-operation Microcode-Based Built-In Self Test for Memory Faults," *2010 International Conference on Signal Acquisition and Processing*, pp. 8-12, 2010, doi: 10.1109/ICSAP.2010.61.
[9] S. Hamdioui, G. Gaydadjiev and A. J. van de Goor, "The state-of-art and future trends in testing embedded memories," *Records of the 2004 International Workshop on Memory Technology, Design and Testing, 2004*., 2004, pp. 54-59, doi: 10.1109/MTDT.2004.1327984.
[10] T. Koshy and C. S. Arun, "Diagnostic data detection of faults in RAM using different march algorithms with BIST scheme," 2*016 International Conference on Emerging Technological Trends (ICETT)*, 2016, pp. 1-6, doi: 10.1109/ICETT.2016.7873754.
[11] M. Parvathi, N. Vasantha and K. S. Parasad, "Modified March C-With Concurrency in Testing For Embedded Memory Applications," *International Journal of VLSI Design & Communication Systems*, vol. 3, no. 5, p. 43, 2012, doi: 10.5121/vlsic.2012.3504.
[12] J. Kinseher, M. Richter and I. Polian, "On the Automated Verification of User-defined MBIST Algorithms," *ZuE 2015; 8. GMM/ITG/GI-Symposium Reliability by Design*, pp. 1-6, 2015.
[13] P. E. Joseph and P. R. Antony, "VLSI design and comparative analysis of memory BIST controllers," *2014 First International Conference on Computational Systems and Communications (ICCSC)*, 2014, pp. 372-376, doi: 10.1109/COMPSC.2014.7032661.
[14] R. Manasa, R. Verma and D. Koppad, "Implementation of BIST Technology using March-LR Algorithm," *2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, 2019, pp. 1208-1212, doi: 10.1109/RTEICT46194.2019.9016784.
[15] M. Jahnavi, P. S. Indrani and M. J. C. Prasad, "Implementation of Concurrent Online MBIST for RFID Memories using March SS Algorithm," *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, vol. 3, no. 6, pp. 2278-3075, 2013.
[16] R. S. Thakur and A. Awasthi, "A Review Paper on Memory Testing using BIST," *Global Research and Development Journal for Engineering*, vol. 1, no. 4, pp. 94-98, 2016.
[17] L. Zhang, Z. Wang, Y. Li and L. Mao, "A Precise Design for Testing High-Speed Embedded Memory using a BIST Circuit," *IETE Journal of Research*, vol. 63, no. 4, pp. 473-481, 2017, doi: 10.1080/03772063.2017.1285259.
[18] A. A. Wojciechowski, K. Marcinek and W. A. Pleskacz, "Configurable MBIST Processor for Embedded Memories Testing," *2019 MIXDES - 26th International Conference "Mixed Design of Integrated Circuits and Systems,"* 2019, pp. 341-344, doi: 10.23919/MIXDES.2019.8787161.
[19] P P. K. Joshi and A. Kapse, "A BIST with diagnostic data compression for embedded RAMs," *2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, 2016, pp. 337-340, doi: 10.1109/ICACCCT.2016.7831658.
[20] P. K. John and R. Antony P, "BIST Architecture for Multiple RAMs in SoC," *Procedia Computer Science*, vol. 115, pp. 159-165, 2017, doi: 10.1016/j.procs.2017.09.121.
[21] C. Hou, J. Li and T. Fu, "A BIST Scheme With the Ability of Diagnostic Data Compression for RAMs," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 12, pp. 2020-2024, Dec. 2014, doi: 10.1109/TCAD.2014.2363393.
[22] P. K. Prabagaran, "Optimizing RAM Testing Method for Test Time Saving Using Automatic Test Equipment," Universiti Sains Malaysia, 2017.
[23] L.-T. Wang, C.-W. Wu and W. Xiaoqing, "VLSI Test Principles and Architectures: Design for Testability," *Elsevier*, 2006.
[24] A. Bosio, S. Di Carlo, G. Di Natale and P. Prinetto, "March AB, a state-of-the-art march test for realistic static linked faults and dynamic faults in SRAMs," *IET Computers & Digital Techniques*, vol. 1, no. 3, pp. 237-245, 2007, doi: 10.1049/iet-cdt:20060137.
[25] N. A. Zakaria, W. Z. W. Hasan, I. A. Halin, R. M. Sidek and X. Wen, "Fault Detection with Optimum March Test Algorithm," *2012 Third International Conference on Intelligent Systems Modelling and Simulation*, 2012, pp. 700-704, doi: 10.1109/ISMS.2012.88.
[26] G. Harutunyan, V. A. Vardanian and Y. Zorian, "Minimal March Tests for Dynamic Faults in Random Access Memories," *Eleventh IEEE European Test Symposium (ETS'06)*, 2006, pp. 43-48, doi: 10.1109/ETS.2006.32.
[27] 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)*, 2019, pp. 47-51, doi: 10.1109/CSITechnol.2019.8895189.
[28] M. L. Bushnell and V. D. Agrawal, "Essentials Of Electronic Testing For Digital, Memory And Mixed-Signal Vlsi

Circuits," New York, Boston, Dordrecht, London, Moscow: *Kluwer Academic Publisher*, 2002, doi: 10.1007/b117406.

[29] B. Mohammad, "Embedded Memory Design for Multi-Core and Systems on Chip," Springer, vol. 116, 2014.

[30] Kevin Zhang, "Embedded Memories for Nano-Scale VLSIs," *Springer*, 2009.

[31] A. J. Van De Goor and G. Gaydadjiev, “March U: a test for unlinked memory faults,” vol. 144, no.3, pp. 155-160, 1997, doi: 10.1049/ip-cds:19971147.

[32] R. D. Adams and P. Drive, "High Performance Memory Testing: Design Principles, Fault Modeling and Self-Test," *Kluwer Academic Publisher*, vol. 22A, 2003, doi: 10.1007/b101876.

[33] Z. Al-Ars and A. J. van de Goor, "Approximating infinite dynamic behavior for DRAM cell defects," *Proceedings 20th IEEE VLSI Test Symposium (VTS 2002)*, 2002, pp. 401-406, doi: 10.1109/VTS.2002.1011171.

[34] D. Hayrapetyan and A. Manukyan, "Modeling dynamic single-cell and coupling faults via automata models," *2017 Computer Science and Information Technologies (CSIT)*, 2017, pp. 65-68, doi: 10.1109/CSITechnol.2017.8312142.

[35] V. G. Mikitjuk, V. N. Yarmolik and A. J. van de Goor, "RAM testing algorithms for detection multiple linked faults," *Proceedings ED&TC European Design and Test Conference*, 1996, pp. 435-439, doi: 10.1109/EDTC.1996.494337.

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

[37] S. Hamdioui, R. Wadsworth, J. D. Reyes and A. J. van de Goor, “Memory Fault Modeling Trends: A Case Study,” *Journal of Electronic Testing*, vol. 20, no. 3, pp. 245–255, 2004, doi: 10.1023/B:JETT.0000029458.57095.bb.

[38] S. Hamdioui, A. J. van de Goor and M. Rodgers, "March SS: a test for all static simple RAM faults," *Proceedings of the 2002 IEEE International Workshop on Memory Technology, Design and Testing (MTDT2002)*, 2002, pp. 95-100, doi: 10.1109/MTDT.2002.1029769.

[39] S. Hamdioui, Z. Al-Ars and A. J. van de Goor, "Testing static and dynamic faults in random access memories," *Proceedings 20th IEEE VLSI Test Symposium (VTS 2002)*, 2002, pp. 395-400, doi: 10.1109/VTS.2002.1011170.

[40] Siemens, “Tessent Memory BIST User’s Manual For Use with Tessent Shell,” 2020.

[41] Kuo-Liang Cheng, Ming-Fu Tsai and Cheng-Wen Wu, "Neighborhood pattern-sensitive fault testing and diagnostics for random-access memories," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 11, pp. 1328-1336, Nov. 2002, doi: 10.1109/TCAD.2002.804101.

[42] A. J. van de Goor, "Testing semiconductor memories : theory and practice," Chichester; New York: *J. Wiley & Sons*, 1991.

[43] A. J. van de Goor, S. Hamdioui and H. Kukner, "Generic, orthogonal and low-cost March Element based memory BIST," *2011 IEEE International Test Conference*, 2011, pp. 1-10, doi: 10.1109/TEST.2011.6139148.

[44] A. J. Van De Goor, "Using march tests to test SRAMs," in *IEEE Design & Test of Computers*, vol. 10, no. 1, pp. 8-14, March 1993, doi: 10.1109/54.199799.

[45] Jin-Fu Li and Cheng-Wen Wu, "Memory fault diagnosis by syndrome compression," *Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001*, 2001, pp. 97-101, doi: 10.1109/DATE.2001.915007.

[46] S. Sharma and V. Moyal, “FSM BASED BIST ARCHITECTURE,” *International Journal of Engineering Sciences and Management*, no. 2, pp. 185-188, 2012.

[47] S. Hamdioui, Z. Al-Ars, A. J. Van De Goor and M. Rodgers, “Dynamic Faults in Random-Access-Memories: Concept, Fault Models and Tests,” *Journal of Electronic Testing*, vol. 19, no. 2, pp. 195-205, 2003, doi: 10.1023/A:1022802010738.

[48] A. S. Syed, D. E. Rani and M. A. Ahmed, “Embedded Memory Test Strategies and Repair,” *International Journal of Engineering*, vol. 30, no. 6, pp. 839-845, 2017, doi: 10.5829/idosi.ije.2017.30.06c.03.

[49] S. M. Al-Harbi and S. K. Gupta, "A methodology for transforming memory tests for in-system testing of direct mapped cache tags," *Proceedings. 16th IEEE VLSI Test Symposium (Cat. No.98TB100231)*, 1998, pp. 394-400, doi: 10.1109/VTEST.1998.670897.

[50] M. Marinescu, “Simple and Efficient Algorithms for Functional RAM Testing,” in *ITC*, 1982, pp. 236-239.

[51] M. -. Chang, W. K. Fuchs and J. H. Patel, "Diagnosis and repair of memory with coupling faults," in *IEEE Transactions on Computers*, vol. 38, no. 4, pp. 493-500, April 1989, doi: 10.1109/12.21142.

[52] Jen-Chieh Yeh, Chi-Feng Wu, Kuo-Liang Cheng, Yung-Fa Chou, Chih-Tsun Huang and Cheng-Wen Wu, "Flash memory built-in self-test using March-like algorithms," *Proceedings First IEEE International Workshop on Electronic Design, Test and Applications '2002*, 2002, pp. 137-141, doi: 10.1109/DELTA.2002.994602.

[53] A. J. van de Goor, G. N. Gaydadjiev, V. G. Mikitjuk and V. N. Yarmolik, "March LR: a test for realistic linked faults," *Proceedings of 14th VLSI Test Symposium*, 1996, pp. 272-280, doi: 10.1109/VTEST.1996.510868.

[54] V. A. Vardanian and Y. Zorian, "A March-based fault location algorithm for static random access memories," *Proceedings of the 2002 IEEE International Workshop on Memory Technology, Design and Testing (MTDT2002)*, 2002, pp. 62-67, doi: 10.1109/MTDT.2002.1029765.

[55] W. Z. W. Hasan, M. Othman and B. S. Suparjo, "A Realistic March-12N Test And Diagnosis Algorithm For SRAM Memories," *2006 IEEE International Conference on Semiconductor Electronics*, 2006, pp. 919-923, doi: 10.1109/SMELEC.2006.380773.

[56] N. A. Zakaria, W. Z. W. Hasan, I. Abdul Halin, R. M. Sidek and X. Wen, "Testing Static Single Cell Faults using static and dynamic data background," *2011 IEEE Student Conference on Research and Development,* 2011, pp. 1-6, doi: 10.1109/SCOReD.2011.6148694.

[57] S. Hamdioui and A. J. Van De Goor, "An experimental analysis of spot defects in SRAMs: realistic fault models and tests," *Proceedings of the Ninth Asian Test Symposium*, 2000, pp. 131-138, doi: 10.1109/ATS.2000.893615.
[58] W. Wu-chen, "SRAM BIST Design Based on March C+ Algorithm," *Modern Electronics Technique*, vol. 34, no. 10, pp. 149-151, 2011.
[59] Y. Wang, Q. Zheng and Y. Yuan, "The Improvement of March C+ Algorithm for Embedded Memory Test," in *Computer Engineering and Technology*, 2016, pp. 31–37, doi: 10.1007/978-3-662-49283-3_4.
[60] K. Roy, I. Chakraborty, M. Ali, A. Ankit and A. Agrawal, "In-Memory Computing in Emerging Memory Technologies for Machine Learning: An Overview," *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1-6, doi: 10.1109/DAC18072.2020.9218505.
[61] T. Tsai, J. Li, C. Hsu and C. Sun, "Testing of In-Memory-Computing 8T SRAMs," *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2019, pp. 1-4, doi: 10.1109/DFT.2019.8875487.

BIOGRAPHIES OF AUTHORS

Aiman Zakwan Jidin is currently a Ph.D. candidate at Universiti Malaysia Perlis, Malaysia. His research topic is focusing on optimizing memory testing algorithm efficiency for improving fault coverage. Previously, he obtained his MEng in Electronic and Microelectronic System from ESIEE Engineering Paris, France in 2011, before working as FPGA IP Core Design Engineer at Altera Corporation Malaysia (now part of Intel). He is a full-time lecturer and researcher at Universiti Teknikal Malaysia Melaka (UTeM), in Electronic and Computer Engineering. His research interests include DFT, VLSI, and FPGA system design.

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 (or previously known as KUKUM) in 2002. He is currently a full-time Senior Lecturer at the Faculty of Electronic Engineering Technology, Universiti Malaysia Perlis.

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's specialization is in synthesizing and tweaking synthesis for performance and low power, leading 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, "Learning from VLSI Design Experience" ISBN: 978-3030032371 with Springer Press, "VHDL Coding and Logic Synthesis with Synopsys" ISBN: 0-12-440651-3 with Academic Press Publication, "Verilog Coding for Logic Synthesis" ISBN: 0-471-42976-7 with John Wiley Publication and "VLIW Microprocessor Hardware Design for ASICs and FPGA" ISBN: 978-0071497022 with McGraw Hill Publication. Lee is also the inventor and co-inventor of 14 design patents granted by the US Patent and Trademark Office (US Patent # 7,057,949 7,010,736 6,891,752 6,771,0936,665,214 6,654,349 6,622,274 6,587,982 6,549,477 6,546,410 6,532,175

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. degree. He obtained his Ph.D. degree 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 Electrical and Electronics Engineering Technology, Universiti Teknikal Malaysia Melaka. His current research interests include hardware security, CMOS reliability, VLSI design, and Electronic Systems Design.