Mathematical Problems of Computer Science 30, 5–17, 2008.

An Efficient Method for Generation of March Tests Based on Formulas

Gurgen Harutunyan†, Davit Melkumyan†, Hasmik Elchyan‡, Valery Vardanian†

† Virage Logic
e-mail: {gurgen, davit.melkumyan, valery.vardanian}@viragelogic.com
‡Yerevan State University
e-mail: hasmik.elchyan@yahoo.com

Abstract

A general method for generation of minimal March tests to detect or diagnose any subclass of simple static or dynamic faults in Static RAMs is described. The proposed method is shown to generate all possible March tests satisfying certain necessary conditions for detection of faults. A correspondence between March tests and natural numbers is established that allows construct a formula that enables generation of all March tests detecting certain faults. As an example, the method is applied for construction of new minimal March tests for detection of several subclasses of three-operation dynamic faults. The method can be generalized for detection/diagnosis of any subset of static or dynamic faults.

1 Introduction

The problem of development of minimal March test algorithms for fault detection or diagnosis in Random Access Memories (RAM) (see [1]) is of prime importance in connection with the increasing density of embedded memories and their dominating portion in system-on-chips (SOC). The memory test algorithms should apply efficient procedures for effective detection, diagnosis and location of defective cells. Recently, several methodologies have been developed to automatically generate March tests for detection and diagnosis of functional fault models (FFMs).

The March test generation method presented in [2] is based on the notion of a transition tree where each path from the root to a leaf corresponds to a certain March test. A March test that covers a selected subset of FFMs is searched in the generated tree. If there is a solution it can be found, but, however, this approach has some disadvantages. The transition tree is unbounded and the search process is of exponential complexity, in general. Besides searching for the shortest path in the transition tree leads to an exhaustive search. In [3], an improvement of [2] is proposed. It restricts the search process to the parts of the tree where a solution exists. Thus, this method allows finding the solution more efficiently. However, fault modeling by means of transition matrices is complex and creates difficulties when extending this approach to more complex problems like fault diagnosis. In [4], another method of March test generation using the notion of Finite State Machines (FSM) is proposed

to construct a memory model and reduce the problem of March test generation for fault detection to an equivalent problem on FSM. Another approach for generation of uniform and minimal March tests, based on development of necessary and sufficient conditions for fault detection, was proposed in [5]. However, this approach was very sensitive to generation of necessary and sufficient conditions. Due to fast development of the technology, new FFMs and defects appear in the production and having a simple concept of fault detection is a principal challenge. The generation of necessary and sufficient conditions for the new FFMs (see [6]) is complicated since the sensitization of conditions for the new faults is rather more complex. Thus, the proposed approach has some principal disadvantages. In [7], the authors developed a simulation-based March test generation method. Based on this approach, several known March tests of different lengths are generated and their fault coverage is calculated by means of RAMSES [6]. Thus, March tests covering different selected sets of FFMs can be generated. However, since exhaustive search is used to find the March tests for detection or diagnosis of a certain set of FFMs, it becomes very essential to reduce drastically the amount of algorithms being considered because the number of March tests grows exponentially depending on the length of March tests. The authors developed some techniques that help reduce the number of tests being considered, but most of them are not so much efficient than they could be and, applying proposed heuristic reductions leads to the loss of guaranties for generation of tests of minimum length. However the proposed approach is flexible enough to adapt easily to new FFMs.

In [8], a method was proposed for generation of efficient March tests for detection or diagnosis of larger classes of unlinked static memory faults. The fault simulation technique was exploited to calculate fault coverage of the generated March tests and a number of restrictions were developed that are used during March test generation. This approach brought to significant reduction of the number of the considered tests during the search process. These restrictions bring to prevention of generating redundant March tests and are proven to not affect on the capability of generation of efficient March tests. Note that generation of March tests for diagnosis is considered as well. Anyhow, the proposed approach is still of exponential complexity and did not allow obtaining minimal tests since some branches in the search tree were not considered at all.

In [9], the authors propose a new approach to automatically generate March tests targeting both static and dynamic memory faults. The main drawback is the complexity (NP-completeness) that reduces the number of total faults that can be included in the fault list. However, it should be noted that, although the authors claim they obtained minimal March tests by this method, it is not true (see, e.g., [10]).

In [11], the authors have shown that the action of single or multiple read immediately after a write operation may cause the inversion of the value stored in the cell. Consequently in order to sensitize this fault it is necessary that the test algorithm has sequences with multiple read operations like w0r0n and w1r1n. In [12] the authors have demonstrated that a cell can undergo a stress equivalent to a read operation (Read Equivalent Stress, RES) when a read/write operation is performed on other cells of the same word line.

In this paper, we propose another, principally new, approach for generation of March tests for detection or diagnosis of certain classes of FFMs. The essence of the approach is in that we do not generate all possible March tests of certain lengths during the search process, but our search is an oriented search. We generate only those March tests of certain lengths that satisfy several necessary conditions for fault detection or diagnosis. Only for those tests we do fault simulation to find tests of minimum length that detect or diagnose

certain classes of FFMs. More precisely, we put into correspondence to each March test a certain integer. Generation of March tests is reduced to generation of a sequence of numbers in a certain range. Actually, the sequence of generated March tests can be described by a formula. According to this formula, we can generate a sequence of integers. Each integer can be interpreted (decoded) as a March(-like) test satisfying some necessary conditions for detection or diagnosis of a certain class of FFMs. Instead of generating all possible March-like sequences(MLS) we generate those ones, which satisfy the necessary conditions and can be described by an integer according to a certain formula. According to the formula we generate all March-like sequences satisfying the necessary (in general, necessary or/and sufficient) conditions. Then from each March-like sequence we generate all March tests that satisfy those necessary (necessary or/and sufficient conditions) by adding the address directions. Then we simulate the currently obtained March test to check if it detects (or diagnoses) all FFMs under consideration. If the test detects all FFMs then we have found the minimal test, otherwise we pass to generation of the next March test and repeat the procedure. In Figure 1 you can see March test generation flow.

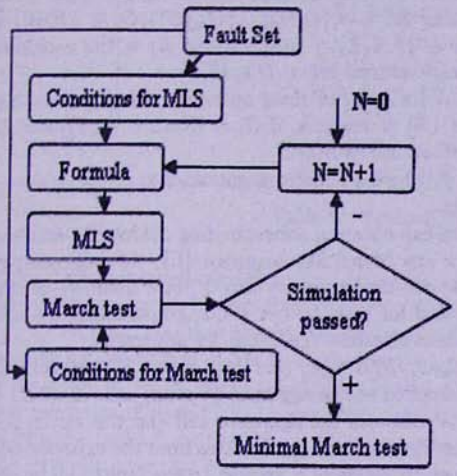

Figure 1: March Tests Generation Flow

It is very important to note that the step of fault simulation can be skipped if the conditions under consideration were sufficient too. Thus for sufficient conditions, the time reduction for generation of minimal tests is relatively higher. This approach reduces drastically the complexity of the process for generation of March tests of minimum length. We have to do fault simulation only for those tests obtained by decoding of certain integers that satisfy the necessary conditions for fault detection or diagnosis. This allows us to reduce drastically the complexity of the generation process. And the experimental results done for several subclasses of three-operation dynamic faults show that we can generate the minimal March tests for certain FFMs in several seconds.

FFM	Fault primitives
$dCFds_{RRR}$	< RxRxRx; x / $\bar{x}$ / - >
$dCFds_{RWR}$	< RxWxRx; x / $\bar{x}$ / - >
$dCFds_{WRR}$	< xWxRxRx; x / $\bar{x}$ / - >
$dCFds_{WWR}$	< xWyWxRx; x / $\bar{x}$ / - >

Table 1: The subset of three-operation dynamic disturb coupling faults

2 Main notations

All necessary definitions on March test algorithms, FPs and FFMs, can be found in [1],[13],[14].

A March test M is a test algorithm with a finite number of March elements $M = \{M_1, M_2, ..., M_k\}$ where each March element M_i consists of an addressing direction $\forall_i$ and a finite number of Read / Write operations $M_i = \forall_i\{O_1(D), ..., O_m(D)\}, O_j \in \{R, W\}$ is a R (Read) or W (Write) operation, $\forall_i \in \{\Uparrow, \Downarrow, \Updownarrow\}$, $\Uparrow$ (respectively, $\Downarrow$) is the ascending (descending) address order, $\Updownarrow$ is an arbitrary address order, $D \in \{0, 1\}$.

A sequence $\{O\}_k = O_1, O_2, ..., O_k$, of Write and / or Read operations of given length k, will be called a March-like sequence (see [15]) of length k, if $O_1 = Wx, x \in \{0, 1\}$, and for each j, $1 < j < k - 1$, the following conditions are satisfied:

(i) If O_j = Rx or Wx then $O_{j+1} \neq$ R $\bar{x}$, where $\bar{x}$ denotes negation of x;

(ii) If O_j = R$\bar{x}$ or W$\bar{x}$ then $O_{j+1} \neq$ Rx.

Obviously, from each March test M we can obtain a corresponding March-like sequence {M} uniquely. Note that, in general, for any March-like sequence {L}, we can construct many March tests by partitioning the March-like sequence into a finite numbers of subsequences $L_1, L_2, ..., L_s$ in different ways and for each $L_i, 1 < i < s$, constructing a March element $M_i = \{\forall_i L_i\}$ by inserting an address direction $\forall_i \in \{\Uparrow, \Downarrow, \Updownarrow\}$ arbitrarily.

For CFs the FPs are described as $< S_a; S_v/F/R >_{[a\$v]}$ (see [16]), where S_a (respectively, S_v) describes the sequence of operations applied to the aggressor (victim) cell "a" ("v") or its state, "$" denotes relation "<" or ">" between the aggressor cell and the victim cell. The faulty behavior F is the observed memory behavior that deviates from the expected one, and $R \in \{0, 1, -\}$ is the result of the Read operation of S applied to the faulty cell in case S ends with operation Read. A "-" in R means the output data is not applicable.

Table 1 describes the subset of "three-operation dynamic disturb" coupling faults (CF). All these faults have the common feature that the third operation of the sensitizing sequence is a Read operation. In Table 1, we denote $x, y \in \{0, 1\}$. Note that, in particular, the so-called Hammer test can be considered as a three-operation dynamic Read disturb CF with three Read operations applied consecutively. Certain defects (see for example, [11],[12]) manifest themselves as dynamic faults and can be detected by consecutive Read operations applied after a Write operation.

3 Main algorithm

Our general method has been applied to construct new minimal March tests for three-operation dynamic disturb CFs. In [15], we have described a method for March test gen-

FFM	March test	March test description	Length
$dCFds_{RRR}$	March RRR	⇕(W0); ⇑(R0,R0,R0,W1,R1,R1,R1); ⇑(R1,R1,R1,W0,R0,R0,R0); ⇕(R0)	16N
$dCFds_{RWR}$	March RWR	⇕(W0); ⇑(R0,W0,R0,W1,R1); ⇑(R1,W1,R1,W0,R0); ⇓(R0,W0,R0,W1,R1); ⇓(R1,W1,R1,W0,R0); ⇕(R0)	22N
$dCFds_{WRR}$	March WRR	⇕(W0); ⇑(R0,W0,R0,R0,W1,R1,R1); ⇑(R1,W1,R1,R1,W0,R0,R0); ⇓(R0,W0,R0,R0,W1,R1,R1); ⇓(R1,W1,R1,R1,W0,R0,R0); ⇕(R0)	30N
$dCFds_{WWR}$	March WWR	⇕(W0); ⇑(R0,W0,W0,R0,W1,W1,R1,W0,W1,R1, W1,W0,R0,W1,W0,R0,W0,W1,R1,W1,W1,R1, W0,W0,R0); ⇕(R0); ⇕(W1); ⇑(R1,W1,W1,R1, W0,W0,R0,W1,W0,R0,W0,W1,R1,W0,W1, R1, W1,W0,R0,W0,W0,R0,W1,W1,R1); ⇕(R1)	54N
$dCFds_{OOR}$	March OOR	⇕(W0); ⇑(R0,W0,R0,W0,W0,R0,W1,W1, R1,R1,W0,W1,R1,W1,W0,R0,R0,R0, W1,R1); ⇑(R1,W1,R1,W1,W1,R1,W0,W0, R0,R0,W1,W0,R0,W0,W1,R1,R1,R1,W0, R0); ⇓(R0,W0,R0,W0,W0,R0,W1,W1,R1, R1,W0,W1,R1,W1,W0,R0,R0,R0,W1,R1); ⇓(R1,W1,R1,W1,W1,R1,W0,W0,R0,R0, W1,W0,R0,W0,W1,R1,R1,R1,W0,R0); ⇕(R0)	82N

Table 2: Minimal March tests

eration. We developed conditions for March-like sequences and for March tests. And using those conditions we do limited steps of search. But for larger length of March tests the search time is too long. So, that is why, we developed a more efficient method which constructs formulas for generation of March tests using those conditions. As all minimal March tests for detection of static faults are already known we applied our method for the subset of three-operation dynamic CFs to show that our method is not only for static faults but also for dynamic faults Note that the method proposed in this paper can be used efficiently not only for fault detection but also for diagnosis too. We have brought experimental results showing how much is the reduction of generated tests for an example of a subclass of three-operation dynamic faults. In Table 2, new minimal March tests for detection of $dCFds_{RRR}$, $dCFds_{RWR}$, $dCFds_{WRR}$, $dCFds_{WWR}$ are described. Also a minimal test March OOR is obtained for detection of all FFMs from $dCFds_{RRR}$, $dCFds_{RWR}$, $dCFds_{WRR}$, $dCFds_{WWR}$.

Our method takes certain conditions as inputs and constructs a formula describing all March-like sequences satisfying the given conditions. We describe the March-like sequences satisfying the conditions by formulas and not another way (for example by "regular expressions") since during March test generation process we need to compare two March-like sequences and it is easier to do that for two integers than the same for two sequences or regular expressions. The other reason is that we define an operation "*" for those formulas and via that operation we can obtain new formulas from the two formulas. For example, if we have faults F1 and F2 and, their formulas, respectively, Formula 1 and Formula 2, are already known for them, then for the set of faults F1 ⋃ F2 we do not need to construct a

new formula by our method. We can use operation "*", and obtain a new formula by this operation. More briefly, Formula 3 = (Formula 1) * (Formula 2).

Let us encode W0 by 0, W1 by 1 and the Read operation by 2, respectively. If Li = 2 then we should decode the i-th operation by R0, if the last number, with index lower than i, non equal to 2, was 0, and otherwise by R1 if that number was 1. Thus, we can put a unique sequence of numbers 0, 1, 2 into correspondence to each March-like sequence. Let us denote this function as Code, which takes a March-like sequence as an input and returns an integer.

Denote by $(x_1, x_2, ..., x_k)_n$ the representation of a sequence of numbers in the n-ary system of numbers. Then each sequence of numbers from the set {0, 1, 2} can be represented as an integer in the ternary system of numbers {0, 1, 2}. For example, March-like sequence {W0, W1, R1, W0, W0, R0} corresponds to $(012002)_3$ in the ternary system, which is integer 137, i.e. Code{W0W1R1W0W0R0) = 137. The function inverse to Code, function $Code^{-1}$ takes an integer and the length of March-like sequence as inputs and returns the March-like sequence. For example, $Code^{-1}(1388, 8)$ = {W0, W1, R1, R1, W0, W1, W0, R0}.

Let Σ = { W0, R0, W1, R1, '...' }, and we shall denote by Σ^* the set of all words constructed in alphabet Σ.

Definition 1 *Let us define condition for March-like sequences recursively.*

1. *Σ^* is a condition.*
2. *If C is a condition then (C) is a condition too.*
3. *If C_1 and C_2 are conditions then $C_1 \vee C_2$ and $C_1 \& C_2$ are conditions.*
4. *There are no other conditions.*

Definition 2 *A condition is basic if it does not contain 'V' and '&' symbols. For example "...W0 W0 R0..." is a basic condition.*

Definition 3 *A condition is simple if it does not contain & symbols.*

For example (...W0 R0...)V(...W1 R1...) is a simple condition.

Definition 4 *A condition is in disjunctive normal form (DNF) or simply a condition is normalized if it is a finite disjunction of finite conjunctions of a basic conditions.*

Now we shall give the description of the algorithm which generates formula having necessary and sufficient or only necessary conditions for detection/diagnosis of faults. The algorithm is divided into 4 parts:

1. Normalization of a condition
2. Simplification of a condition
3. Parameterization of a condition
4. Formula construction

Now we shall describe each part separately.

1. Normalization of condition. In this step we perform all possible following transformations of condition

- R1&(R2VR3) $\Longrightarrow$ (R1&R2) V (R1&R3)
- (R1VR2)&R3 $\Longrightarrow$ (R1&R3) V (R2&R3)

Where R1, R2, R3 are any parts of conditions.

Proposition 1 *Any condition can be transformed to disjunctive normal form applying finite number of several transformations:*

- *R1&(R2VR3) = (R1&R2) V (R1&R3)*
- *(R1VR2)&R3 = (R1&R3) V (R2&R3)*

2. Simplification of normalized condition. Let R be a condition, L(R) be all march like sequences satisfying condition R, R = R1 & R2, where R1, R2 are basic conditions. We propose a method for constructing basic condition R3, L(R3) = L(R1)$\bigcap$L(R2). This method is divided into 3 parts.

1. Construction of deterministic finite automata (DFA) corresponding to conditions
2. Performing operation on DFA-s
3. Get the condition corresponding to the resulted automaton

Example: (... W0 R0 ...) & (... W1 R1 ...)

Let M1 = (Q1, Σ, delta1, q1, F1), M2 = (Q2, Σ, delta2, q2, F2), where M1 and M2 are finite automata(see [17]).

Q1 = {q0, q1, q2 }, Σ = {W0, W1, R }, delta1, q1=q0, F1 = {q2}

Q2 = {p0, p1, p2 }, Σ = {W0, W1, R }, delta2, q2=p0, F2 = {p2}

In Table 3 transition functions delta1, delta2 are described.

The construction of a DFA, M3, such that L(M3) = L(M1)$\bigcap$ L(M2) is given as follows.

Let S1 x S2 be the cross product of sets S1 and S2, all ordered pairs. Then M3 = (Q1 x Q2, Σ, delta3, [q1,q2], F1 x F2) where [q1,q2] is an ordered pair from Q1 x Q2, delta3 is constructed from delta3([x1,x2],a) = [delta1(x1,a), delta2(x2,a)] for all a in Σ and all [x1,x2] in Q1 x Q2. You can find the table description of delta3 in Table 4.

Proposition 2 *L(M3) = L(M1)$\bigcap$ L(M2)*

In the third step we construct a regular expression corresponding to the result automaton. The construction algorithm is presented in [17]. And we take as a condition this regular expression.

As a result, we get (...W0 R0...W1 R1...) V (...W1 R1...W0 R0...)

Here is the example of simplification.

(...W0R0R0R0...W0R0R0R0...V ...W0R0R0R0R0R0R0...) &

(...W1R1R1R1...W1R1R1R1 ... V ...W1R1R1R1R1R1R1...) =

(...W0R0R0R0...W0R0R0R0...W1R1R1R1...W1R1R1R1...) V

(...W0R0R0R0...W1R1R1R1...W0R0R0R0...W1R1R1R1...) V

(...W0R0R0R0...W1R1R1R1...W1R1R1R1...W0R0R0R0...) V

delta1	W0	W1	R	delta2	W0	W1	R
q0	q1	q0	q0	p0	p0	p1	p0
q1	q1	q0	q2	p1	p0	p1	p2
q2	q2	q2	q2	p2	p2	p2	p2

Table 3: Transition functions delta1 and delta2

delta3	W0	W1	R
q00	q10	q01	q00
q10	q10	q01	q20
q01	q10	q01	q02
q20	q20	q21	q20
q02	q21	q02	q20
q21	q20	q21	q22
q12	q12	q02	q22
q22	q22	q22	q22

Table 4: Transition function delta3

(...W1R1R1R1...W1R1R1R1...W0R0R0R0...W0R0R0R0...) V
(...W1R1R1R1...W0R0R0R0...W1R1R1R1...W0R0R0R0...) V
(...W1R1R1R1...W0R0R0R0...W0R0R0R0...W1R1R1R1...) V
(...W0R0R0R0R0R0R0...W1R1R1R1...W1R1R1R1...) V
(... W1R1R1R1...W0R0R0R0R0R0R0...W1R1R1R1...) V
(... W1R1R1R1... W1R1R1R1 ... W0R0R0R0R0R0R0...) V
(... W1R1R1R1R1R1R1... W0 R0 R0 R0 ... W0 R0 R0 R0 ...) V
(... W0 R0 R0 R0 ... W1R1R1R1R1R1... W0 R0 R0 R0 ...) V
(... W0 R0 R0 R0 ... W0 R0 R0 R0 ... W1R1R1R1R1R1R1...) V
(... W0 R0 R0 R0 R0 R0 R0 ... W1R1R1R1R1R1R1...) V
(... W1R1R1R1R1R1R1 ... W0R0R0R0R0R0R0 ...)

3. Parameterization.

Let R be a normalized condition. $R = A_1 \vee A_2 ... \vee A_N$, where $A_i, 1 \leq i \leq N$ are basic conditions and $A_i = (C_{i_1}, ..., C_{i_{k_i}})$, where C_{ij} is a '...' symbol or is a $Op_{ij} \in \{W(d_{ij}), R(d_{ij})\}, d_{ij} \in \{0, 1\}, 1 \leq i \leq N, 1 \leq j \leq k_i$

Now we shall define the parameterization step which gets the normalized conditions and returns a fully parameterized normalized condition.

Suppose we have A_i and $A_j, i \neq j, k_i = k_j$. We say that we can parameterize these conditions if $\forall p, 1 \leq p \leq k_i$ for both conditions p-th symbols are '...' or are the same operations with the same arguments or with different arguments. If they are with the same arguments we get a variable instead of the argument, otherwise the negation of the variable.

For example, if the conditions are (... W1 W0 R0 ...) and (... W0 W1 R1 ...) the parameterized condition would be $(...WxW\bar{x}R\bar{x}...), x \in \{0, 1\}$.

Definition 5 *Condition R is fully parameterized if for any pair (A_i, A_j) of basic conditions, $1 \leq i, j \leq N, i \neq j$, we can not perform the parameterization step.*

Note that after finite number of parameterization steps we would get a fully parameterized condition.

After parameterization of our example we would get:

$(...WyRyRyRy...WyRyRyRy...W\bar{y}R\bar{y}R\bar{y}R\bar{y}...W\bar{y}R\bar{y}R\bar{y}R\bar{y}...)V$
$(...WyRyRyRy...W\bar{y}R\bar{y}R\bar{y}R\bar{y}...WyRyRyRy...W\bar{y}R\bar{y}R\bar{y}R\bar{y}...)V$
$(...WyRyRyRy...W\bar{y}R\bar{y}R\bar{y}R\bar{y}...W\bar{y}R\bar{y}R\bar{y}R\bar{y}...WyRyRyRy...)V$
$(...WyRyRyRyRyRyRy...W\bar{y}R\bar{y}R\bar{y}R\bar{y}...W\bar{y}R\bar{y}R\bar{y}R\bar{y}...)V$
$(...W\bar{y}R\bar{y}R\bar{y}R\bar{y}...WyRyRyRyRyRyRy...W\bar{y}R\bar{y}R\bar{y}R\bar{y}...)V$
$(...W\bar{y}R\bar{y}R\bar{y}R\bar{y}...W\bar{y}R\bar{y}R\bar{y}R\bar{y}...WyRyRyRyRyRyRy...)V$
$(...WyRyRyRyRyRyRy...W\bar{y}R\bar{y}R\bar{y}R\bar{y}R\bar{y}R\bar{y}R\bar{y}...)$

4. Formula Construction. In this stage we construct a formula from the parameterized simple condition.

First, let us define the following functions, which are the basic blocs for construction of the formula.

$$R_n(i) = \left\{ \begin{array}{ll} 1, & \text{if } i = n \\ 0, & \text{if } i \neq n \end{array} \right\}, \quad l(x) = \left\{ \begin{array}{ll} 1, & \text{if } x > 0 \\ 0, & \text{if } x = 0 \end{array} \right\},$$

$$F_k(n, \tilde{A}, \tilde{S}, \tilde{Y}, \tilde{P}) = \sum_{j=1}^{k+1} y_j 3^{\sum_{r=1}^{j-1} p_r + s_r} + \sum_{j=1}^{k} a_j 3^{p_j + \sum_{r=1}^{j-1} p_r + s_r},$$

$$y_r \in \{0, ..., 3^{p_r - 1}\}, 1 \leq r \leq k+1, \frac{(3^{S_k} y_{k+1} + a_k)}{3^{s_k + p_{k+1} - 1}} > 1$$

$$a_e \in \{0, ..., 3^{S_e - 1}\}, 1 \leq e \leq k,$$

$$\tilde{A} = (a_1, a_2, ..., a_k), \tilde{S} = (s_1, s_2, ..., s_k), \tilde{Y} = (y_1, y_2, ..., y_{k+1}), \tilde{P} = (p_1, p_2, ..., p_{k+1})$$

Let $R = A_1 \vee A_2 \vee ... \vee A_n$ be a condition, where $A_i = ...C_{i1}...C_{i2}, ..., C_{ik_i}..., 1 \leq i \leq n$ and C_{ij} is a sequence of operations. For example, R can be $(..., W0, R0, ..., W1, R1, ...) \vee (..., W1, R1, ..., W0, R0, ...)$.

Here is the formula which gives the codes of all March-like sequences satisfying the condition R.

$$F(n, x, Y, P) = \sum_{i=1}^{N} ([R_i(j) l(n - \sum_{t=1}^{k_i} |C_{it}| + 1) \times$$

$$\times F_{k_i}(n, Code(C_{i1}), ..., Code(C_{ik_i}), |C_{i1}|, ..., |C_{ik_i}|, y_{i1}, ..., y_{ik_i+1}, p_{i1}, ..., p_{ik_i+1})]$$

In the formula, n is the length of March-like sequences, $x \in \{0,1\}$. y_{ij} and p_{ij} are variables with ranges that are specified in the definition of functions F_k. Putting in the formula all the allowed values from those ranges, we get all the March-like sequences of length n. Y and P are the sets of variables y_{ij} and p_{ij} respectively.

We will explain the details by means of an example of constructing it for a certain subclass of three-operation dynamic faults. Also, we described in Table 2 five new minimal March tests obtained by us by using the new method of March test generation.

Let us consider the class of three-operation dynamic faults denoted by us as $dCFds_{RRR}$.

Here are the propositions for detection of $dCFds_{RRR}$.

Proposition 3 *In order that a March test detect $dCFds_{RRR}$ it is necessary that the corresponding March-like sequence to contain at least two disjoint entries of each of the sequences R0, R0, R0 and R1, R1, R1. [x]* denotes expression "x" is present or missing.*

Proposition 4 *For detection of* $< RxRxRx; x / \bar{x} / - >_{a<v}$, *it is necessary and sufficient that the March test contain the entries from one of the following cases:*

1. ... $\Uparrow$ ([Rx, ...,Opx,]* Rx, Rx, Rx, ...); ...

2. ... $\Updownarrow$ (..., Rx, Rx, Rx [,..., Opx]*), $\Updownarrow$(Rx, ...), ...

Proposition 5 *For detection of* $<RxRxRx; \bar{x}/x/->_{a<v}$, *it is necessary and sufficient for the March test to contain the entries from one of the following cases:*

1. ... $\Uparrow(R(\bar{x}), ..., Rx, Rx, Rx, ...)$; ...

2. ... $\Downarrow$(..., Rx, Rx, Rx,..., $Op(\bar{x})$); $\Updownarrow(R(\bar{x}),...)$;...

Proposition 6 *For detection of* $<RxRxRx; x/\bar{x}/->_{a>v}$, *it is necessary and sufficient for the March test to contain the entries from one of the following cases:*

1. ... $\Downarrow$([Rx, ...,Opx,]* Rx, Rx, Rx,);

2. ... $\Uparrow$ (..., Rx, Rx, Rx [,..., Opx]*); $\Updownarrow$(Rx,);

Proposition 7 *For detection of* $<RxRxRx; \bar{x}/x/->_{a>v}$, *it is necessary and sufficient for the March test to contain the entries from one of the following cases:*

1. ... $\Downarrow(R(\bar{x}), ..., Rx, Rx, Rx, ...)$; ...

2. ... $\Uparrow$(..., Rx, Rx, Rx,..., $Op(\bar{x})$); $\Updownarrow(R(\bar{x}),...)$;...

As an illustration of our method, we shall show how we obtained the minimal March test March RRR for $dCFds_{RRR}$.

We have constructed a formula describing all March-like sequences satisfying Proposition 3. After it, from those March-like sequences, we constructed all March tests satisfying the formula taking into account conditions in Propositions 4-7. For n = 14, the corresponding formula is: $F(x)=729x+728+2187(729(\bar{x})+728)=1594323(\bar{x})+729x+1592864$, where $x \in \{0, 1\}$. $F(0)=3187187=(12222220222222)_3$, $Code^{-1}(3187187, 14)$=W1, R1, R1, R1, R1, R1, R1, W0, R0, R0, R0, R0, R0, R0. $F(1)=1593593=(02222221222222)_3$, $Code^{-1}(1593593, 14)$=W0, R0, R0, R0, R0, R0, R0, W1, R1, R1, R1, R1, R1, R1.

Actually, we have obtained two March-like sequences satisfying the formula. Constructing all March tests from those two March-like sequences, we obtain that none of them satisfies all conditions in Propositions 4-7. Thus, we can assure that the minimal March test for $dCFds_{RRR}$ is longer than 14. We have got the same result for n=15. So, it means that any March test detecting $dCFds_{RRR}$ has at least 16 operations. We have constructed all March-like sequences of length 16 and have generated from them many March tests satisfying Propositions 4-7. Note that they are minimal as we mentioned above that there is no any March test of length 15N or low detecting $dCFds_{RRR}$. For example, when n=16, let us consider one of the integers obtained by means of the corresponding formula: $14171597=(0222122222202222)_3$, which corresponds to March-like sequence {W0, R0, R0, R0, W1, R1, R1, R1, R1, R1, W0, R0, R0, R0, R0} and we obtained March RRR from it by adding the addressing directions at the corresponding positions.

In Table 5, there are some experimental results. We see that the number of all March-like sequences is significantly larger than those obtained by the approach based on formulas

All March-like sequences	March-like sequences obtained by the formula for $dCFds_{RRR}$	Length
3188646	2	14
9565938	18	15
28697814	162	16
86093442	1170	17
258280326	6480	18
774840978	32886	19
2324522934	156492	20
6973568802	708588	21

Table 5: Experimental results for $dCFds_{RRR}$

and proposed in this paper. For example, for length N=14, there are 3188646 March-like sequences to be considered for generation of all March tests from those March-like sequences exhaustively. Meanwhile, our new method allows construction of a formula that generates only two March-like sequences to be used for generation of all March tests detecting all FFMs from the subclass $dCFds_{RRR}$. Thus, a great number of March-like sequences should not be considered at all when constructing the minimal March test for detection of all FFMs from $dCFds_{RRR}$.

4 Conclusions

A new approach for generation of March tests for detection or diagnosis of certain classes of static or dynamic FFMs is proposed. The essence of the approach is in that we do not generate all possible March tests of certain lengths during the search process, but our search is an oriented search. This allows us to reduce drastically the complexity of the generation process. The experimental results show that we can generate the minimal March tests for certain FFMs in several seconds. An example is brought for a subclass of 3-operation dynamic faults, namely $dCFds_{RRR}$. As a result of applying the new method, we have generated a few minimal March tests for detection of several subclasses of dynamic faults. Note that relatively small amount of March tests generated based on the corresponding formula should be simulated to assure that the generated test covers all FFMs from the considered subclass.

In the future, we are going to apply the new method to generation of minimal March tests for detection or diagnosis of the classes of static and dynamic FFMs that have not had minimal March tests so far.

References

[1] A.J. van de Goor, Testing semiconductor memories: Theory and Practice, *John Wiley & Sons*, 1991.

[2] A. J. van de Goor, B. Smit, "The automatic generation of march tests", *IEEE International Workshop Memory Technology Design and Testing*, pp. 131-136, 1993.

[3] K. Zarrineh, S. J. Upadhyaya, and S. Chakravarty, "A new framework for generating optimal march tests for memory arrays", *Proc. Int. Conf. (ITC)*, pp. 73-82, 1998.

[4] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, "An optimal algorithm for the automatic generation of march tests", *DATE 2002, IEEE Design, Automation and Test in Europe Conference and Exhibition*, pp. 938 939, 2002.

[5] S. M. Al-Harbi, S. K. Gupta, "An efficient methodology for generating optimal and uniform march tests", *IEEE VLSI Test Symposium*, pp. 231-237, 2001.

[6] C.-F. Wu, C.-T. Huang, and C.-W. Wu, "RAMSES: a fast memory fault simulator", *Proc. Int. Symp. Defect and Fault Tolerance in VLSI Systems (DFT), Albuquerque*, pp. 165-173, Nov. 1999.

[7] J.-F. Li, K.-L. Cheng, C.-T. Huang, and C.-W. Wu, "March-based RAM diagnostic algorithms for stuck-at and coupling faults", *Proc. IEEE ITC*, pp. 758-767, 2001.

[8] T. Gyonjyan, V. A. Vardanian, "An efficient algorithm for generating minimal march tests for fault detection and diagnosis in static random access memories", *International Design and Test Workshop, Dubai*, pp. 19-20, 2006.

[9] A. Benso, A. Bosio, S. Di Carlo, G. Di Natale, P. Prinetto, "Automatic march tests generation for static and dynamic faults in SRAMs",

[10] G. Harutunyan, V. A. Vardanian, Y. Zorian, "Minimal march tests for dynamic faults in random access memories", *Journal of Electronic Testing: Theory and Applications*, Vol. 23, Number 1, pp. 55-74, 2007.

[11] L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, M. Bastian, "Resistive-open defect injection in SRAM core-cell: analysis and comparison between 0.13 μm and 90 nm technologies", *Design Automation Conference*, pp. 857-862, 2005.

[12] L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, S. Borri, M. Hage-Hassan, "Dynamic read destructive ault in embedded-SRAMs: analysis and march test solutions", *Proc. IEEE European Test Symposium*, 2004.

[13] S. Hamdioui, A.J. van de Goor, M. Rodgers, "March SS: a test for all static simple faults", *Records of IEEE Int. Workshop MTDT*, pp. 95-100, 2002.

[14] S. Hamdioui, A.J. van de Goor, M. Rodgers, "Linked faults in random access memories: concept, fault models, test algorithms, and industrial results", *IEEE Trans. CAD*, vol. 23, No. 5, pp. 737-756, 2004.

[15] G. Harutunyan, V. A. Vardanian, Y. Zorian, "Minimal march tests for unlinked static faults in random access memories", *Proc. 23rd IEEE VLSI Test Symposium, Palm Springs, CA, USA*, pp. 53-59, 2005.

[16] A. J. van de Goor, Z. Al-Ars, "Functional memory faults. a formal notation and a taxonomy", *Proc. IEEE VLSI Test Symposium, Montreal, Canada*, pp. 281-290, 2000.

[17] Aho, Sethi, Ullman, Compilers: Principles, Techniques, and Tools, *Addison-Wesley*, ISBN 0-201-10088-6, 1986.

Բանաձևերի միջոցով մարշ տեստերի կառուցման արդյունավետ մեթոդ

Գ. Հարությունյան, Դ. Մելքումյան, Հ. Էլչյան, Վ. Վարդանյան

Ամփոփում

Նկարագրված է հիշվող սարքերում ստատիկ կամ դինամիկ անսարքությունների կամայական ենթաբազմությունը հայտնաբերող կամ ախտորոշող մինիմալ մարշ տեստերի կառուցման մեթոդ: Առաջարկված մեթոդում կառուցվում են որոշակի անհրաժեշտ պայմաններին բավարարող բոլոր մարշ տեստերը: Մարշ տեստերի և բնական թվերի միջև տրվել է արտապատկերում, որը հնարավորություն է տալիս կառուցել բանաձև: Բանաձևի միջոցով ստացվում են բոլոր այն մարշ տեստերը, որոնք հայտնաբերում կամ ախտորոշում են տրված անսարքությունները: Մեթոդը կիրառվել է երեք գործողությամբ դինամիկ անսարքությունների որոշակի ենթադասը հայտարերող մինիմալ մարշ տեստեր ստանալու համար: Մեթոդը հնարավորություն է տալիս կառուցել կամայական ստատիկ և դինամիկ անսարքություններ հայտաբերող կամ ախտորոշող մինիմալ մարշ տեստեր: