OPTIMIZING MARCH ALGORITHM FOR TESTING EMBEDDED MEMORY: A CASE STUDY

Nitin Sharma, Dr. Pradeep Kumar, Mr. Sunil Kumar

ASET, AMITY UNIVERSITY, Noida (India)

ABSTRACT

The increase in integrity of the recent VLSI technology has enabled a trend of a lot of new, small and portable applications. The popularity of these portable applications, for ex notebook computers and cellular phones demand even higher performance and lower-power consumption. Embedded semiconductor memories form a big portion of the overall hardware on these portable devices. Testing the semiconductor memories is increasingly important today because of the high density of current memory chips. March algorithms are well known for memory testing because of their simplicity and good fault coverage. This is the reason that they are the dominant test algorithms implemented in most modern memory BIST. The paper describes the proposal to optimize one of the most popular march elements based algorithm c-. The proposed march algorithm is modified march c- algorithm which uses concurrent technique. The basic purpose of this proposal is to reduce the complexity of implementation and the test time. Because of concurrency in testing the sequences the test results were observed in less time than the traditional March tests. This technique is applied for a memory of size 1024x16 and is generic enough to be extended to any other memory size.

Keywords: Embedded memory, fault coverage, March test

I INTRODUCTION

With every techno node and new product lines being attacked in semiconductor landscape, due to ever growing sizes and density, Embedded Memories are using more and more complex design structures resulting in the chances of occurring manufacturing defects to be more compared to any other embedded core on SOC. Hence testing of embedded memory is becoming an ever growing real challenge for every design architect. For SOCs, to have direct access to the core is already one of the major problems in the field of testing and diagnosis [1-4]. Further because of the limited available bandwidth between the primary inputs of the system chip and the embedded core, the external access for test purpose is becoming increasingly infeasible. This has prompted a very strong interest in the self-test of the embedded memory arrays. In particular, mostly because they provide defined detection properties for classical memory array faults such as stuck at faults and transition faults, functional March tests have found wide acceptance in this domain.

During Memory tests, each location in a memory device is tested to be working. This testing involves writing a specific set / pattern of data to each memory address location and verifying this data by reading it back and
comparing with an already anticipated response. Depending on the values readback, if they are the same as those that were written, the memory device is said to pass the test, otherwise device fails. There are various test methodologies that have evolved from the years to identify the memory defects. One such test methodology is memory built-in self-test which involves built-in self-test circuitry for each memory array to apply some data pattern in a pre-defined manner, reading back the response, comparing the response and reporting the status. The memory BIST is implemented embedding various march algorithms to test the memories.

The advantage of March tests, which makes it acceptable from industrial point of view, lay in the fact that high fault coverage can be obtained and the test time were usually linear with the size of the memory. March based algorithms were capable of locating and identifying the fault types ultimately helping to catch design and manufacturing errors. In order to further improve this acceptance of march based tests, the method proposed in this paper is Modified March C- algorithm with concurrent technique. The intent is of this modified algorithm is to retain the high fault coverage of March C but the test run should finish in a reduced time frame or reduced number of cycles. The paper describes the functional fault models in the memory, classical and March based tests in section II. The proposed Modified March C- algorithm is discussed in section III. Results and comparisons are discussed in section IV. Section V concludes the work.

II HISTORY OF FUNCTIONAL FAULT MODELS

For testing purpose, various functional faults are represented as fault models modeled after observing the behavior of faults in memories so that functional tests to detect these faults can be used. This modeling helps to clarify, simplify and generalize the testing approach of a memory. The fault model of the faults determines the quality of tests strongly in terms of its fault coverage, its test vector length as well as the test time.

There are various fault models to test the functional faults. Some most popular ones are such as stuck at faults. While dealing with SRAMS, coupling faults are also considered. Other faults to consider are Address decoder faults and bridging faults when dealing with DRAM. The most commonly occurring faults in general are stuck at faults [5-7].

**Stuck at fault (SAF):** The stuck-at fault (SAF) considers that the logic value of a cell or line is always 0 (stuck-at 0 or SA0) or always 1 (stuck-at 1 or SA1). To detect and locate all stuck-at faults, a test must satisfy the following requirement: from each cell, a 0 and a 1 must be read.

**Transition Faults (TF):** The transition fault (TF) is a special case of the SAF. A cell or line that fails to undergo a $0 \rightarrow 1$ transition after a write operation is said to contain an up transition fault. Similarly, a down transition fault indicates the failure of making $1 \rightarrow 0$ transitions. According to van de Goor [8, 9], a test to detect and locate all the transition faults should satisfy the following requirement: each cell must undergo an up transition (cell goes from 0 to 1) and a down transition (cell goes from 1 to 0) and be read after each transition before undergoing any further transitions.

The algorithms targeting fault detection for both SAFs and TFs are MATS++ and March C- algorithm. Now with the new modified C- Algorithm, although different in test length, there is no fault coverage loss compared to MATS++ and traditional March C- for capturing stuck at and transition fault. In additions the fault coverage in context of detecting other faults is as good as well. The detection process can be understood by examining the March C- algorithm as indicated in expression below.
March Test Notation: A March test consists of a finite sequence of March elements [10-12]. A March element is a finite sequence of operations or primitives applied to every memory cell before proceeding to next cell. For example, \( \downarrow (r1, w0) \) is a March element and \( r0 \) is a March primitive. The address order in a March element can be increasing (↑), decreasing (↓), or either increasing or decreasing (↕). An operation can be either writing a 0 or 1 into a cell (w0 or w1), or reading a 0 or 1 from a cell (r0 or r1). Accordingly notation of March C-test is described as follows:

\[
\{ \downarrow (w0); \uparrow (r0, w1); \uparrow (r1, w0); \downarrow (r0, w1); \downarrow (r1, w0); \uparrow (r0) \}
\]

March C-algorithm has 6 elements as shown with a complexity of 10n.

III MODIFIED MARCH C-ALGORITHM

The proposed Modified March C-algorithm almost similar to March C but it follows concurrency in testing the sequences. The steps for following the concurrency are as follows:

- Group entire memory into subgroups.
- For each subgroup, generate all test vectors for the first fault in the group.
- Simulate all faults in the subgroup to select one vector that detects most faults in that subgroup. If more vectors than one detect the same number of faults within the group, then select the vector that detects most faults outside the group as well.
- Apply the final test vectors to all subgroups concurrently

In the proposed method the memory is divided into two subgroups such as M1 and M2. Then, the algorithms are applied for concurrency. The following are the elements in Modified March C-algorithm.

\[
M1: \{ \uparrow (w0); \uparrow (r0, w1); \uparrow (r1); \downarrow (w0); \downarrow (r0, w1); \downarrow (r1); \}
\]

\[
M2: \{ \uparrow (w1); \uparrow (r1, w0); \uparrow (r0); \downarrow (w1); \downarrow (r1, w0); \downarrow (r0); \}
\]

The number of March elements is same as traditional March C and is 6 but because of concurrency the complexity is reduced to 8n.

The pseudo code for modified march C is as follows:

```plaintext
//for writing 0s in block 1 and writing 1s in block 2, let n and m are rows and columns
for(i=0;i<=(n-1)/2;i=i+1)
begin
for(j=0;j<(m-1);j=j+1)
mem[i][j]=0;//write 0 in m1
end
for(i=(n-1)/2;i<(n-1);i=i+1)
begin
for(j=0;j<(m-1);j=j+1)
mem[i][j]=1;//write 1 in m2
end
```

www.ijarse.com
//for reading background and for writing alternatively
for(i=0;i<(n-1)/2;i=i+1)
begin
for(j=0;j<(m-1);j=j+1)
begin
if(mem[i][j]==0)
mem[i][j]=1;
else return;
end
end
for(i=(n-1)/2;i<(n-1);i=i+1)
begin
for(j=0;j<(m-1);j=j+1)
begin
if(mem[i][j]==1)
mem[i][j]=0;
else return;
end
end

According to Modified March C- elements, when 0s are written in one memory group, 1s will be written in another group concurrently. So the test sequence can be taken through an inverter hence true form will go to one block of memory and complement form will go to another block of memory. The method directly reduces the time required to write and read the bit concurrently. Ultimately this reduces the logic to be put on the address decoder path and operation controller path of the memory BIST circuit thereby resulting in lesser logic on the path to memory. This reduces the test time and test costs also. Finally, there may be additional design cost in terms of some hardware gates added to generate the test sequence to perform the test concurrently.

IV RESULTS AND COMPARISONS

Table 1 indicates delay introduced by performance for each element present in traditional March C- algorithm under faulty condition. Using SA fault models the overall delay observed as 11.602ns.

Table 2 shows the delay performance using Modified March C- algorithm. In this also delay performance were calculated under faulty conditions. Under faulty condition the overall delay was observed as 11.494ns. Hence it is proved that using Modified March C- algorithm using concurrency the overall delay is greatly reducing. It is giving at speed test performance than any other traditional algorithm. The result tables also provide the information on minimum input arrival time before clock and maximum output time after the clock. Simulation was carried using Xilinx ISE 10.1i tool for the
device XC3S4004tq144 and tested on Spartan 3 kit. Fig 1 and 2 shows the simulation results respectively for modified march elements I and II when fault is imposed.

**Table 1: Result of traditional March C- Algorithm**

<table>
<thead>
<tr>
<th>March Element</th>
<th>Min period</th>
<th>Min i/p arrival time before clock</th>
<th>Min o/p required time after clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>M₁: ↑(w₀)</td>
<td>1.483</td>
<td>3.439</td>
<td>6.314</td>
</tr>
<tr>
<td>M₂: ↑(r₀,w₁)</td>
<td>1.585</td>
<td>3.504</td>
<td>6.318</td>
</tr>
<tr>
<td>M₃: ↑(r₁,w₀)</td>
<td>1.585</td>
<td>3.504</td>
<td>6.318</td>
</tr>
<tr>
<td>M₄: ↓(r₀,w₁)</td>
<td>1.585</td>
<td>3.504</td>
<td>6.318</td>
</tr>
<tr>
<td>M₅: ↓(r₁,w₀)</td>
<td>1.585</td>
<td>3.504</td>
<td>6.318</td>
</tr>
<tr>
<td>M₆: ↓(r₀)</td>
<td>2.196</td>
<td>3.955</td>
<td>6.318</td>
</tr>
</tbody>
</table>

All the delay times mentioned are in ns.

**Table 2: Result of traditional March C- Algorithm**

<table>
<thead>
<tr>
<th>March Element</th>
<th>Min period</th>
<th>Min i/p arrival time before clock</th>
<th>Min o/p required time after clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>M₁: ↑(w₀)</td>
<td>1.483</td>
<td>3.439</td>
<td>6.28</td>
</tr>
<tr>
<td>M₂: ↑(w₁)</td>
<td></td>
<td></td>
<td>6.441</td>
</tr>
<tr>
<td>M₃: ↑(r₀,w₁)</td>
<td>2.132</td>
<td>4.755</td>
<td>6.318</td>
</tr>
<tr>
<td>M₄: ↑(r₁,w₀)</td>
<td>2.132</td>
<td>3.96</td>
<td>6.318</td>
</tr>
<tr>
<td>M₅: ↑(r₁)</td>
<td>2.132</td>
<td>3.96</td>
<td>6.318</td>
</tr>
<tr>
<td>M₆: ↑(r₀)</td>
<td>2.132</td>
<td>3.96</td>
<td>6.314</td>
</tr>
<tr>
<td>M₇: ↓(10)</td>
<td>2.132</td>
<td>3.96</td>
<td>6.314</td>
</tr>
</tbody>
</table>
### Table 3: Comparison

<table>
<thead>
<tr>
<th>Type of Algo</th>
<th>Complexity</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional C-</td>
<td>10N</td>
<td>13.782</td>
</tr>
<tr>
<td>Modified C-</td>
<td>8N</td>
<td>11.783</td>
</tr>
</tbody>
</table>

### V CONCLUSION

This paper defines the functional fault model and compared the traditional march C-algorithm with modified march C-algorithm in terms of speed of the test sequence and complexity of the number of test sequences. The crucial part in testing is how well the test can be completed in minimum time with minimal test length. The modified march algorithm has proved that the test length is minimal as well as the time required to test SAF also minimum when compared with traditional march C-. Hence this modified march is much comparable and could be used for detection of various faults other than SAF as future work.

### REFERENCES


