PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100-ft Ring Road, Bengaluru – 560 085, Karnataka, India

Report on

Implementation of March Algorithms for Memory Testing

Submitted by

Vaishnavi V C (PES1201800591)
Rajath J (PES1201801707)
Chetan S R (PES1201801911)

Jan-May 2021

under the guidance of

Prof. Sunitha M S
Associate Professor
Department of ECE
PES University Bengaluru -560085

TABLE OF CONTENTS

Objective

To design a state machine that uses the march algorithm to determine addressing order and data written into cells. Permanent faults like stuck-at-faults, transition faults, coupling faults, multiple select address faults are introduced in a memory model and detected by running March patterns.

Comparison between few march patterns and their respective fault coverage will also be depicted using simulation graphs.

Abstract

The purpose of memory systems is to store massive amounts of data and hence different fault models and test algorithms are required to test memories.

Fault models such as stuck-at faults, transition faults, coupling faults, neighborhood pattern sensitive faults (NPSF),address decoder faults are sufficient for memory testing.

Some of the popular algorithms to test the memory model are:

- Exhaustive test pattern
- Walking pattern
- March patterns
- Checkerboard algorithm

March tests have proved to be simpler and faster, and have emerged as the most popular ones for memory testing.

March Algorithm

A March test applies patterns that “march” up and down the memory address while writing values to and reading values from known memory locations. March algorithms are a set of deterministic pattern generator algorithms used in testing of memory systems as they have a good fault coverage.

It involves changing the data at a given address and leaving the address in the changed state when proceeding to the next memory address in order.

Generally, before a march sequence begins, a background data type is written into the memory. (i.e. All addresses are either written to 0 or 1 before the process begins)

March Patterns

There are various types of March tests with different fault coverages :

1. March C
2. March C-
3. Enhanced March C-
4. March A
5. March X
6. March Y

Fault Models

Fault refers to a defect or imperfection in a system. They are of various types such as : Permanent faults, Intermittent faults, and Transient faults.
Permanent faults are those that persist indefinitely after their occurrence and are discussed in this report.

Stuck-At-Fault

Stuck at faults in memory is the one in which the logic value of a cell (or line in the sense amplifier or driver) is always 0 or 1

Transition Faults

In transition faults a cell fails to make a transition (0 to 1) or a (1 to 0) when it is written. Memory cells will retain either 0 or 1 state once written and cannot transition back.

Down transition fault : Cell transitions only from 0 to 1 but not otherwise

Up transition fault :Cell transitions only from 1 to 0 but not otherwise

Coupling Faults

Coupling faults are faults in which fault occurs in a cell because of coupling with other cells.

Generally, in coupling fault models it is assumed that any “two” cells can be coupled together leading to irregular behavior in these two cells, it is called 2-cell coupling fault model where one cell is the aggressor cell (undergoes coupling) and the other is a victim cell (undergoes transition cell).

State Coupling fault : If aggressor cell is in a particular state then victim cell is forced to be in error state

Inversion coupling fault : Data stored in victim cell is inverted when there is either an upward or downward transition in the aggressor cell

Idempotent coupling fault : Due to upward or downward transition in the aggressor cell victim cell is forced to be in error state of 0 or 1

Decoder Faults

Row and column decoder comprises the address decoder of a memory. From the context of memory testing four types of faults are considered in address decoder.

Disconnected address fault: No address from which a particular word can be accessed.

Mis directed address fault: Address accesses wrong word

Multiple select address fault: With certain addresses multiple words can be accessed simultaneously.

Multiple select cell fault: A certain word can be accessed with multiple addresses.

Code of Memory Module:

```
module memory#(parameter RAWIDTH = 2,  CAWIDTH = 2) //RAWIDTH=Row Address Width and CAWIDTH=Column
address width
(// Clock and Reset    //Column and Row address    // Write Interface     // Read Interface
  input clk,             input [RAWIDTH-1:0]RA,       input we,             input re,
  input rst,             input [CAWIDTH-1:0]CA,       input datain,         output reg dataout
);
//create memory model
localparam rDEPTH = 2**RAWIDTH;
localparam cDEPTH = 2**CAWIDTH;

reg [cDEPTH-1:0] memory [rDEPTH-1:0];

integer i,j;

always @(posedge clk)
begin
  if(rst) begin
    for(i=0; i < 2**RAWIDTH ;i=i+1) begin
      for(j=0; j < 2**CAWIDTH;j=j+1) begin
                  memory[i][j] <= 0;
               end
           end
    end
    else
        if(we)
            memory[RA][CA] <= datain;
end



always @ (posedge clk)
begin
    if(re)
        dataout <= memory[RA][CA];
end

endmodule
```

Codes used to insert different faults to the Memory:

```
//Stuck at fault :
memory[0][1] <= 1;
memory[0][0] <= 0;
memory[3][0] <= 1;

//Transition fault
always @(negedge memory[1][3])
begin
    memory[1][3] =~ memory[1][3];
end


// Multiple Select Address Fault
always @(memory[3][3])
begin
    memory[0][3] <= datain;
end
```

```
// State Coupling Fault
// victim cell address < aggressor cell address
always @(memory[2][2])
if(memory[2][2]==1)
begin
    memory[2][0]=1;
end

// Inversion Coupling Fault
// victim cell address < aggressor cell address
always @(memory[2][3])
begin
    memory[0][2] = ~memory[0][2];
end

// Idempotent Coupling Fault
// victim cell address > aggressor cell address
always @(memory[1][2])
begin
    memory[2][1] = 1;
end
```

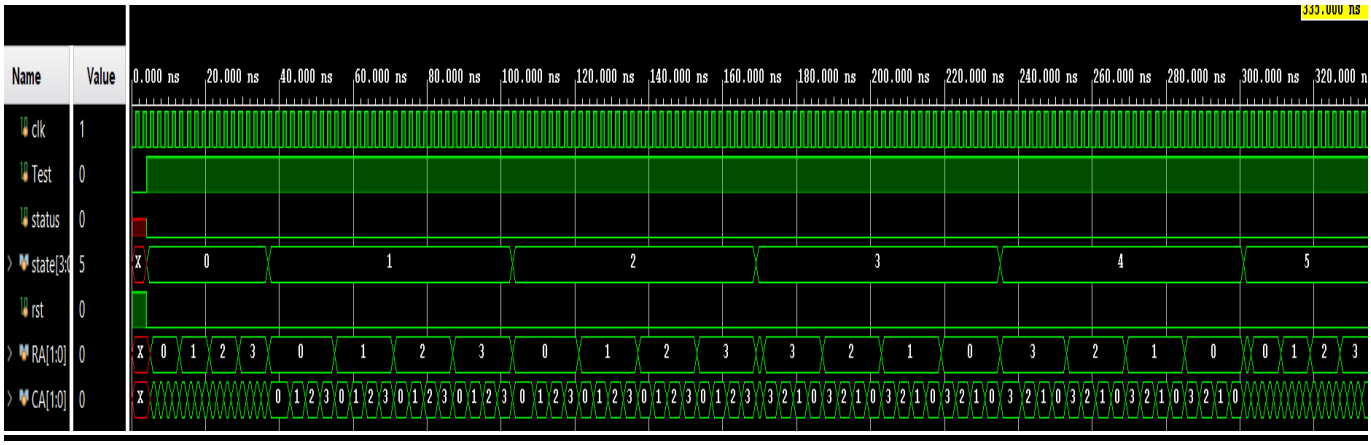
Pseudo Code/ Algorithm used:

```
//State Flow Model
For every positive edge of the clock: if(Test) then continue
    Case ( current_state )
    //do the following
    if(first iteration)
        RA = 0; CA = 0; //if ascending order is followed
        Print ("beginning");
    else
        if(CA reached the Max column size)
            RA = RA+1; CA = 0;
        else
            CA = CA+1;
        Count++; // increment count
    we/re= 1; // Set Read or Write to high based
              // on the Element operation to be performed
    datain = 0/1; // Based on Element operation
    if ( Count exceeds the cell size of the memory )
        Print ("done");
        Current_state = Next_state;
```

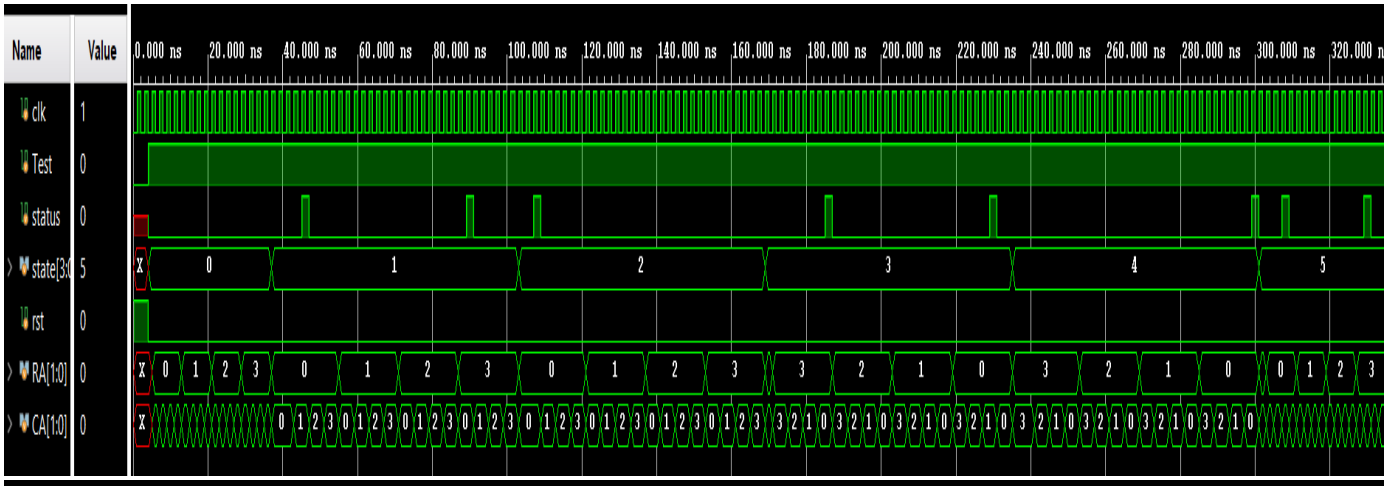
Note: Nested case statements are used for states with multiple read or write stages.

Simulation Results

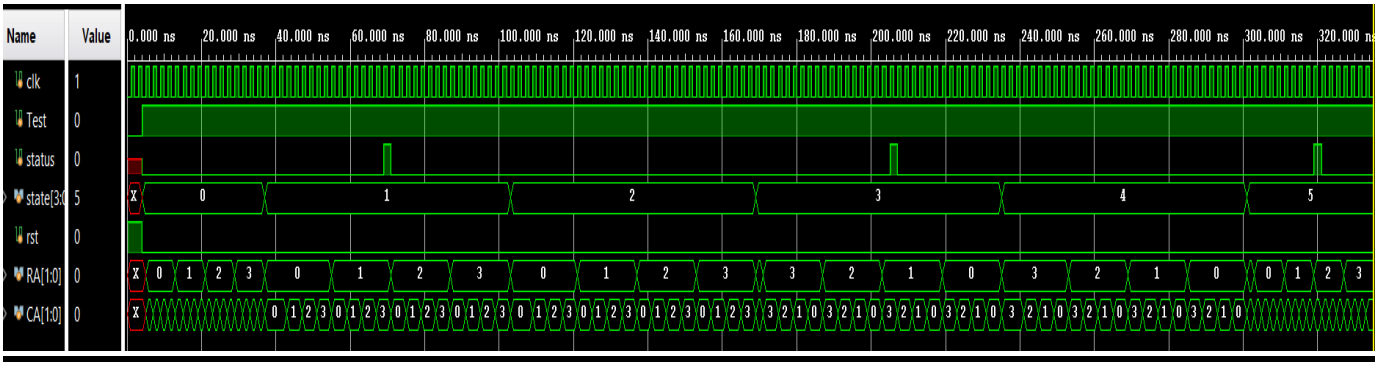
March C- against a fault free memory



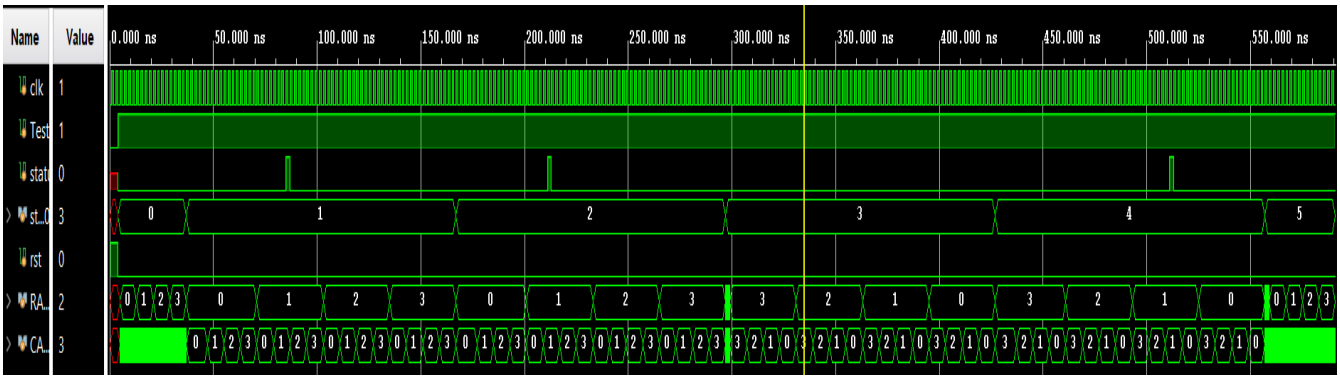
March C- detecting Stuck-at fault



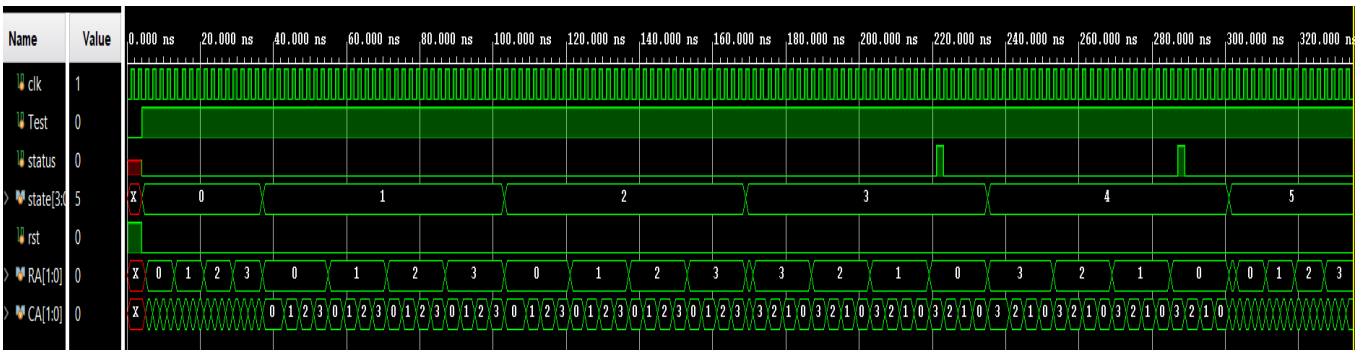
March C- detecting a Transition fault



March C- enhanced detecting a NPSF fault

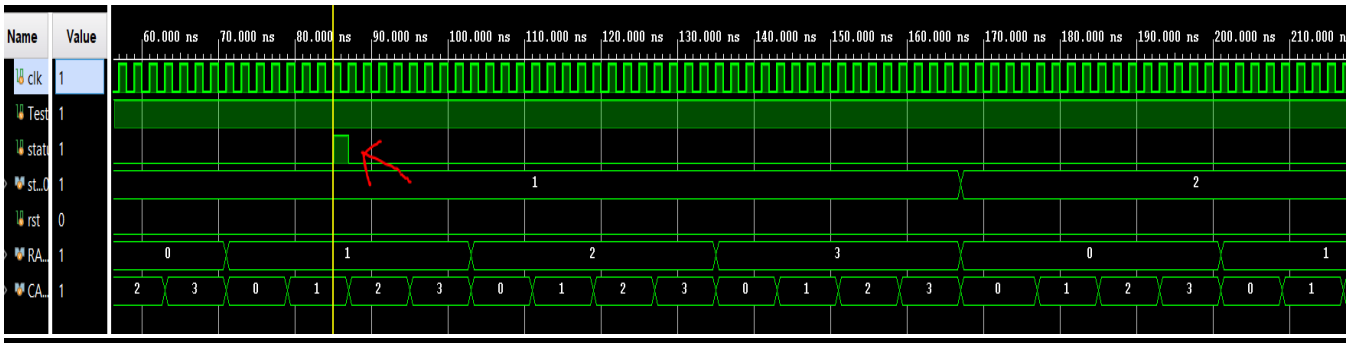


March C- detecting a Multiple Select address fault

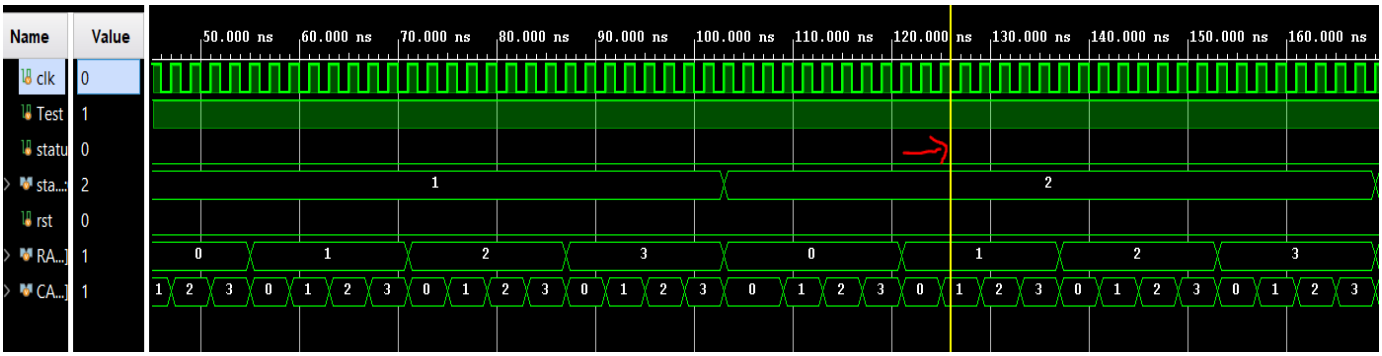


Console Outputs for a few selective trials

Enhanced March C- detecting a late update in a defective cell

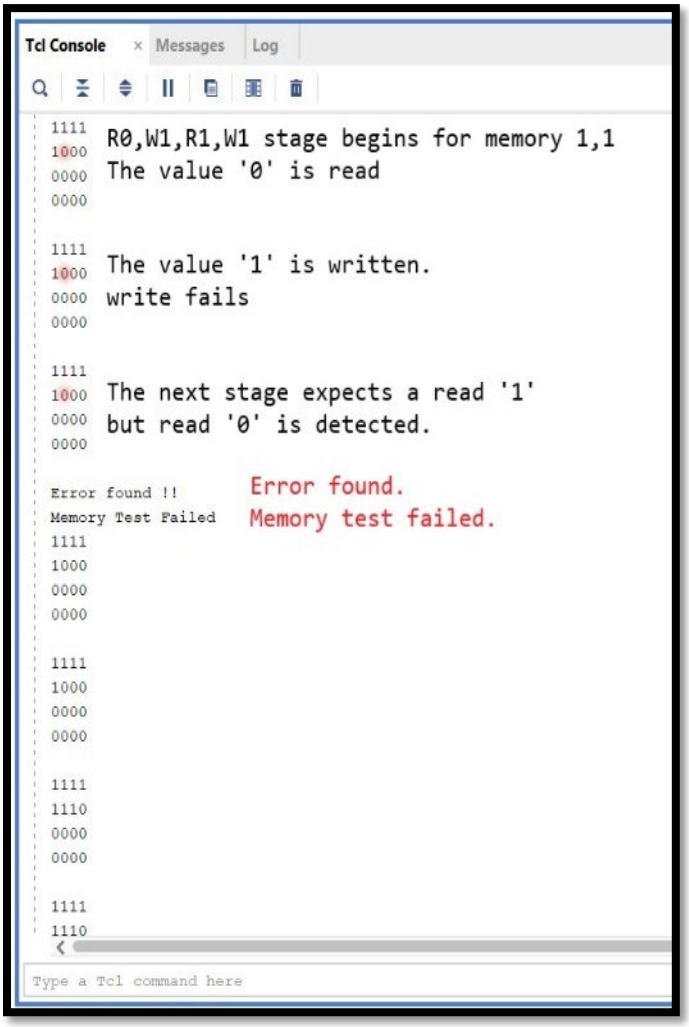


March C- unable to detect a late update in a defective cell

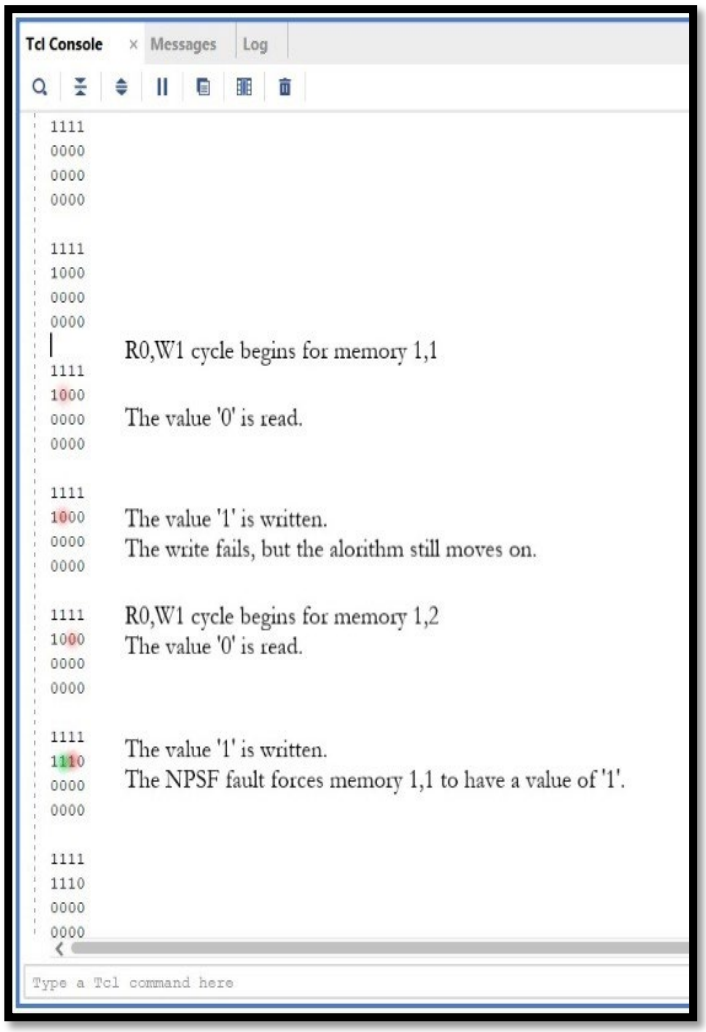


Comparison of Console Output:

Enhanced March C - :



March C - :



The fault gets covered up and March C- fails to detect it.

Fault Coverage Table

March algorithm	Stuck-at fault	Transition fault	State coupling fault	Inversion coupling fault	Idempotent coupling fault	Multiple select address fault	NPSF	Late write
March C	yes	yes	yes	yes	yes	yes	yes	no
March C-	yes	yes	yes	yes	yes	yes	yes	no
March C-enhanced	yes	yes	yes	yes	yes	yes	yes	yes
March A	yes	yes	yes	yes	yes	yes	yes	no
March X	yes	yes	yes	no	yes	yes	yes	no
March Y	yes	yes	yes	no	yes	yes	yes	yes

Conclusion

March algorithms are used to determine addressing order and data written into cells. Permanent faults like stuck-at-faults, transition faults, coupling faults, multiple select address faults were introduced in a memory model and were successfully detected by running March patterns.

The implementation is done in Verilog and follows state flow modelling. The code is highly modular and follows a simple structure. Thereby allowing future experimentation.

Simulation graphs are obtained for respective march patterns and their fault coverages are tabulated.

The code file also displays the update of the memory at each operation. This helps in understanding the behavior of the Memory and working of the Algorithm under consideration.

References:

[1] P. C. B. Common, “Fault Modeling Why Model Faults ?,” pp. 1–31, 2002.

[2] M. S. Sunita, “Memory design and testing.”

[3] “Memory Testing – An Insight into Algorithms and Self Repair Mechanism.”

[4] F. Models, “Memory Market Share in 1999,” pp. 1–27.