A Low-Cost FPGA-Based Test and Diagnosis Architecture for SRAMs

A Low-Cost FPGA-Based Test and Diagnosis Architecture for SRAMs

Authors: Di Carlo S., Prinetto P., Scionti A., Figueras J. Manich S., Rodriguez-Montanes R.,

Published in the Proceedings of the IEEE First International Conference on Advances in System Testing and Validation Lifecycle (VALID), 20-25 Sept. 2009, Lisbon, PT.

DOI: 10.1109/VALID.2009.29

A Low-Cost FPGA-Based Test and Diagnosis Architecture for SRAMs

Stefano Di Carlo, Paolo Prinetto, Alberto Scionti
Control and Computer Engineering Department
Politecnico di Torino
Torino, Italy
Email: {stefano.dicarlo,paolo.prinetto,alberto.scionti}@polito.it

Joan Figueras, Salvador Manich, Rosa Rodriguez-Montañés
Departament d'Enginyeria Electronica
Universitat Politècnica de Catalunya
Barcelona, Spain
Email: {figueras,manich,rosa}@eel.upc.es

***Abstract*—[1] The continuous improvement of manufacturing technologies allows the realization of integrated circuits containing an ever increasing number of transistors. A major part of these devices is devoted to realize memory blocks. Test and diagnosis of memory circuits are therefore an important challenge for improving quality of next generation integrated circuits. This paper proposes a flexible platform for testing and diagnosis of static random access memories. The architecture is based on the use of a low-cost FPGA based board allowing high diagnosability while keeping cost at a very low level.**

***Keywords*—Memory Diagnosis, Memory Testing, March Test.**

I. Introduction

With the increasing of memories' density and area, yield of most System-On-Chip (SOC) is dominated by embedded memories [1]. Manufacturing errors and defects should be therefore detected, diagnosed, and localized, to improve memory quality, reliability, and yield [2] .

Although diagnosis has been widely used for Static Random Access Memories (SRAMs), it is still considered an expensive process due to long test time, and complex fault analysis procedures. Efficient test and diagnosis algorithms, as well as low-cost diagnosis platforms will play an ever increasing role in the semiconductor industry.

While high-end Automatic Test Equipments (ATE) characterized by high grade of automation, digital and analog test capability, and high-speed test execution, are nowadays used by manufacturers at the end of production, their high cost and complex requirements in terms of setup make the introduction of low-cost ATE systems mandatory during the preliminary chip evaluation phase.

This paper proposes an efficient, easy-to-use, and flexible solution for test and diagnosis of faults in SRAM circuits. Test and diagnosis stimulus are applied through a low cost hardware platform controlled by a FPGA soft core microprocessor. The possibility of easily configuring the interface between the microprocessor and the target circuit, as well as the flexibility in terms of stimulus application and diagnostic data collection that are actually managed by a software running on the soft core microprocessor make this system a viable solution for preliminary chip evaluation.

The use of a modified March C- is proposed in this paper to diagnose typical memory array fault models, and bus fault models. Moreover, a very compact data structure to store diagnostic data is proposed. The proposed platform has been successfully applied in the diagnosis of a set of commercial memory devices.

The paper is organized as follows: Section II overviews related works in the field of memory test and diagnosis. Section III describes the adopted diagnosis solution while Section IV describes the main architecture of the proposed hardware platform and presents experimental results. Finally Section V concludes the paper.

II. Related Works

Among the different types of algorithms proposed for testing SRAMs, march tests have proven to be faster, simpler and regularly structured [2]. Several diagnosis march tests have been proposed in the literature, e.g., [3], [4], [5], [6], [7], [8], [9], [10]. Bergfeld et al. [5] proposed a 12N march test able to distinguish single-cell faults from multiple-cell faults for a N-bit memory. In [6], Niggemeyer et al. proposed a diagnosis schema based on a combination of faults decomposition, and output tracing of the memory outputs. In [7], Li et al. proposed a three-phase diagnosis schema able to locate the aggressor bit of coupling faults. In [4], a 12N march CL algorithm for fault detection and partial diagnosis was reported. Also, a 4N march-like algorithm is used to locate the aggressor bits (words) of some CFs (inter-word CFs) in bit-oriented (word-oriented) memories. However, this diagnosis schema cannot achieve full diagnosis. In [8], a 15N march test, and an adaptive 3N march-like test were proposed to achieve full diagnosis on coupling faults. In [9], [10], the authors proposed an efficient fault location and full diagnosis algorithm for dynamic faults.

Even if the proposed solutions proved to provide high diagnosability, due to their complexity, their integration into low-cost test and diagnosis platforms is still challenging.

[1]This work was supported by the Integrated Actions Italy Spain project IT09A142F3 "DEFECT BASED TESTING AND DIAGNOSIS OF SRAM MEMORIES"

TABLE I
FAULTY DICTIONARY FOR SINGLE AND TWO-CELLS MEMORY ARRAY FAULTS: STUCK-AT-FAULTS (SAF), TRANSITION FAULTS (TF), STATE COUPLING FAULTS (CF_{st}), IDEMPOTENT COUPLING FAULTS (CF_{id}), INVERSION COUPLING FAULTS (CFf_{in})

	M_0	M_1	M_2		M_3	M_4		M_5	M_6		M_7	M_8		M_9
Fault type	-	-	R_0	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}
SAF(0)	0	0	0	1	1	1	0	0	0	1	1	1	0	0
TF(0)	0	0	0	1	1	1	0	0	0	1	1	1	0	0
SAF(1)	0	0	1	0	0	0	1	1	1	0	0	0	1	1
TF(1) v_0=1	0	0	1	0	0	0	1	1	1	0	0	0	1	1
TF(1) v_0=0	0	0	0	0	0	0	1	1	1	0	0	0	1	1
$CFst_{0;0}$ - $a < v$	0	0	0	0	0	1	0	0	0	1	0	0	0	0
$CFst_{0;0}$ - $a > v$	0	0	0	1	0	0	0	0	0	0	0	1	0	0
$CFst_{0;1}$ - $a < v$	0	0	0	0	0	0	1	1	1	0	0	0	0	1
$CFst_{0;1}$ - $a > v$	0	0	1	0	0	0	0	1	0	0	0	0	1	1
$CFst_{1;0}$ - $a < v$	0	0	0	1	1	0	0	0	0	0	1	1	0	0
$CFst_{1;0}$ - $a > v$	0	0	0	0	1	1	0	0	0	1	1	0	0	0
$CFst_{1;1}$ - $a < v$	0	0	1	0	0	0	0	0	0	0	0	0	1	0
$CFst_{1;1}$ - $a > v$	0	0	0	0	0	0	1	0	1	0	0	0	0	0
$CFid_{0w_1;1}$ - $a < v$	0	0	1	0	0	0	0	0	0	0	0	0	0	0
$CFid_{0w_1;0}$ - $a < v$	0	0	0	0	0	0	0	0	0	0	1	1	0	0
$CFid_{1w_0;1}$ - $a < v$	0	0	0	0	0	0	0	0	0	0	0	0	0	1
$CFid_{1w_0;0}$ - $a < v$	0	0	0	0	0	1	0	0	0	0	0	0	0	0
$CFid_{0w_1;1}$ - $a > v$	0	0	0	0	0	0	0	0	1	0	0	0	0	0
$CFid_{0w_1;0}$ - $a > v$	0	0	0	0	1	1	0	0	0	0	0	0	0	0
$CFid_{1w_0;1}$ - $a > v$	0	0	0	0	0	0	0	1	1	0	0	0	0	0
$CFid_{1w_0;0}$ - $a > v$	0	0	0	0	0	0	0	0	0	0	0	1	0	0
$CFin_{0w_1}$ - $a < v$	0	0	1	0	0	0	0	0	0	0	1	1	0	0
$CFin_{1w_0}$ - $a < v$	0	0	0	0	0	1	0	0	0	0	0	0	0	1
$CFin_{0w_1}$ - $a > v$	0	0	0	0	1	1	0	0	1	0	0	0	0	0
$CFin_{1w_0}$ - $a > v$	0	0	0	0	0	0	0	1	1	0	0	1	0	0

III. DIAGNOSIS ALGORITHM

Memory diagnosis involves fault detection, fault identification in terms of fault models, and fault localization. March tests [2] are widely used for fault detection and localization due to their linear complexity with respect to the number of memory cells.

A march test consists of a finite sequence of *march elements* delimited by curly brackets. Each march element is composed of a sequence of read (r_d where d is the expected data value), or write (w_d, where d is the data value written in the cell) operations, delimited by round brackets and applied to all memory cells according to a predefined *address order*. Two address orders are usually considered: the ascending order ($\Uparrow$), and the descending one ($\Downarrow$).

If a fault f is detected by a march test containing k read operations, the *march syndrome* for f is defined as a k-tuple:

$$S_f = (R_0, R_1, \ldots, R_k) \tag{1}$$

where $R_i \in \{0, 1\}$ is equal to 1 if the i^{th} read operation of the march test detects f, and is equal to 0 otherwise. The set of march syndromes for the detected faults represents the *fault dictionary* of the test. A fault can be correctly diagnosed if it is identified by a unique syndrome in the dictionary. Diagnostic march tests are therefore constructed by adding additional operations able to introduce distinct syndromes in the fault dictionary. The *Diagnosability Ratio* (DR), defined as the ratio between the number of faults that can be correctly identified over the total number of faults in the dictionary, measures the efficiency of the diagnosis.

In this work we consider the modified March C- algorithm reported in Figure 1, first proposed in [7] and here extended. While this algorithm was mainly considered for test and diagnosis of single stuck-at and coupling faults, here its use is extended to the diagnosis of bridging and data bus faults.

$$\begin{aligned}
&\{\underset{M_0}{\Uparrow (w0)};\ \underset{M_1}{\Uparrow (w0)};\ \underset{M_2}{\Uparrow (r0, w1, r1)};\\
&\underset{M_3}{\Uparrow (r1)};\ \underset{M_4}{\Uparrow (r1, w0, r0)};\\
&\underset{M_5}{\Uparrow (r0)};\ \underset{M_6}{\Downarrow (r0, w1, r1)};\ \underset{M_7}{\Uparrow (r1)};\\
&\underset{M_8}{\Downarrow (r1, w0, r0)};\ \underset{M_9}{\Uparrow (r0)};\}
\end{aligned}$$

Fig. 1. $18n$ diagnostic march test for cell-array faults, and bus faults

Table I shows the fault dictionary of the proposed march test for typical memory cell array fault models. The test algorithm can detect different types of coupling faults between two different cells of the memory array, as well as transition and stuck-at faults (see [11] for a complete definition of each fault model). The second march element (M_1) is useful to avoid CF_{id} and CF_{in} caused by an unknown state (metastability) of the memory circuit before the reset. Being all march syndromes in Table I different except SAF(0) with TF(0) and SAF(1) with TF(1) $v_0 = 1$, this algorithm allows an high diagnosability ratio on the considered fault dictionary. Considering TF(1) two cases are possible, depending whether the previous value of the faulty cell was 0 ($v_0 = 0$) or 1 ($v_0 = 1$). In the first case the fault is detected the first time by R_4 (w_0 in M_4 does not work correctly), while in the second case by R_0 since the cell is blocked at 1 (as in the case of a SAF(1)).

Considering coupling faults, the location of both the aggressor cell (a), and the victim cell (v), should be located. This can be accomplished by running the additional $3n$ march test proposed in [7], and reported in Figure 2.

At the end of the execution of the algorithm in Figure 1 the relative position of the aggressor cell w.r.t. the victim cell is known, together with the address of the victim cell. $Loc_{(L)}$ is selected when $a < v$, while $Loc_{(H)}$ is selected when $a > v$. Let $\Uparrow$ and $\Downarrow$ denote here the application of the memory operations from cell 0 to $v-1$, and from $n-1$ to $v+1$, respectively, and A_s, V the aggressor state after the execution of the diagnostic algorithm, and the fault free state of the victim, respectively. Table II shows the relationship between the value A of the aggressor used in the additional $3n$ march test and the A_s state activating the coupling fault. The procedure starts selecting the proper algorithm (i.e. $Loc_{(L)}$ or $Loc_{(H)}$) and the proper data value A. The first march element initializes the content of the lower (higher) portion of the memory in which the aggressor cell is present with value $\bar{A}$ ($w_{\bar{A}}$), and then initializes the victim with value V (w^v_V). The second march element checks the changes in the victim cell (r^v_V) caused by writing the coupling fault activation state in the aggressor cell (w_A). The last address used in the second march element, causing a change in the victim state, is the location of the aggressor cell.

For example considering the $\text{CFid}_{0w_1;0}$ - $a > v$ the aggressor cell has an address higher than the victim one, thus the $Loc_{(H)}$ is used to determine the address of the aggressor cell, and the value $A = 1$ is selected.

$$Loc_{(L)}: \{ \Uparrow (w_{\bar{A}}); w^v_V; \Uparrow (w_A, r^v_V); \}$$

or

$$Loc_{(H)}: \{ \Downarrow (w_{\bar{A}}); w^v_V; \Downarrow (w_A, r^v_V); \}$$

Fig. 2. $3n$ march test for the identification of aggressor and victim cells in a coupling fault.

TABLE II
RELATIONSHIP BETWEEN AGGRESSOR ACTIVATION STATE A_s AND THE ADDITIONAL MARCH TEST VALUE A

Fault	**A_s**	**A**	**Fault**	**A_s**	**A**
CF_{st}	0	0	CF_{st}	1	1
CF_{id}	$\uparrow$	1	CF_{id}	$\downarrow$	0
CF_{in}	$\uparrow$	1	CF_{in}	$\downarrow$	0

A. Bridging faults

By considering a single fault scenario, the proposed march test can be extended to correctly diagnose bridging faults in the cell-array. Bridging faults are caused by a short circuit among two or more lines that manifests itself as a bidirectional couplings between cells. Bridging faults are a common type of faults in modern chips, and therefore represent a concern also for memory devices.

Bringing faults can be categorized into: (i) AND Bridging Faults (ABF), where the faulty behavior is given by the logic AND among the faulty cells, i.e., if one of the cells is at low level all the others are forced to a low level, (ii) OR Bridging Faults (OBF), where the faulty behavior is given by the logic OR among the faulty cells, i.e., if one of the cells is at high level, all other cells are forced to the high level, and (iii) defects where bridged nodes do not behave as ABF or OBF. In this paper we consider the first two categories of BFs, being these faults the most common in memory devices.

Considering 2-cells ABFs, similarly to SAF(0) faults are detected by $r1$ operations, since it is not possible to force 1 in a faulty cell with the second faulty cell initialized to 0. 2-cells ABFs can be therefore identified by two concurrent SAF(0) detected in the faulty cells. In a similar way, 2-cells OBFs are detected similarly to a SAF(1) by $r0$ operations, since it is not possible to write 0 in a faulty cell when the other faulty cell is initialized to 1. 2-cells OBFs can be therefore identified by two concurrent SAF(1) detected in the involved cells. There is only one exception. If the previous value of the two cells is 0, the R_0 read operation on the first faulty cell works correctly and the fault is detected only by the remaining operations, similarly to a TF(1).

Table III summaries the fault dictionary for bridging faults.

B. Data and address bus faults

By considering a single fault scenario, the proposed march test can also be applied to diagnose address bus faults (AF), and data bus faults (I/OF). Faults on the address bus and on the data bus lead to faulty behaviors involving several cells.

For example, a SAF(0) on the data bus affects all write and read operations on the memory. In order to distinguish from bridging faults (see Section III-A) it is enough to verify if more than two consecutive cells manifest the fault. In particular, when addressing the memory in ascending order, it is enough to verify if the first three cells manifest the fault detected by R_1, R_2, R_3, and R_8, and, on the other hand, when addressing the memory in descending order, it is enough to verify if the

TABLE III
FAULT DICTIONARY FOR BRIDGING FAULTS (TWO FAULTS PER TEST)

	M_0	M_1	M_2		M_3	M_4		M_5	M_6		M_7	M_8		M_9
Fault type	-	-	R_0	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}
ABF	0	0	0	1	1	1	0	0	0	1	1	1	0	0
OBF	0	0	0 (1)	0	0	0	1	1	1	0	0	0	1	1

last three cells manifest the same fault detected by R_7, and R_9. In case of SAF (1) on the data bus the situation is similar, but faults are detected by R_0, R_4, R_5, and R_{11} for the first three cells and R_6, R_{10} for the last three cells.

To diagnose a SAF on the address bus we have to identify when faults occur. For example, in a 4-bit memory with a SAF(0) on the second bit of the address, by addressing the memory in ascending order, the first fault occurs with address 0010 (cell 0000 is erroneously addressed), and the fault can be detected by R_0 and R_3. In a similar way, when addressing the memory in descending order, the fault first arises with address 1111 (cell 1101 is erroneously addressed), and the fault is detected by R_6 and R_9. Unfortunately, it is not possible to distinguish between SAF(0), and SAF(1). Similar considerations can be applied to diagnose ABFs and OBFs both on the address and on the data bus. Table IV summarizes the fault dictionary for data and address bus faults.

IV. TEST PLATFORM ARCHITECTURE AND EXPERIMENTAL RESULTS

The proposed test architecture consists of an hardware platform based on a low-cost FPGA board. In particular the board used in our experiments is a Xilinx™ ML-403.

The FPGA, connected to a slave board used to accommodate the SRAM circuit under test, is used to execute the diagnostic algorithm proposed in Section III and to collect test information. The board is accessible through different communication channels including a 10/100 ethernet link allowing remote testing sessions and an USB channel used to program the FPGA for the specific test, and to collect diagnosis information at the end of the test execution. The availability of these communication channels provides a simple and fast facility to manage all steps of the test and diagnosis process.

The connection between the main testing board and the slave board is obtained using standard expansion connectors. The use of a FPGA-based board allows an easy customization according to target the DUT (Device Under Test) saving time and money [12].

A Microblaze™ microprocessor mapped on the FPGA has been used to implement all diagnosis processes. The Microblaze is a soft-core microprocessor based on a 32-bit Harvard RISC architecture. It can access both internal FPGA resources and external blocks. Having all test and diagnosis activities implemented as software routines allows easy customization to the target DUT and to the target set of experiments.

A. Test Slave Board

The slave board has been developed to support different types of SRAM circuits. In particular, in our prototype, we adopted a dual-in-line SRAM socket. The socket is connected to the main testing board through a double communication channel. One of the two channels is directly connected to the expansion port, while the other one includes a bridge (e.g., resistors to simulate faults on the address or data buses) to connect the socket with the main board. To reduce the level of noise captured by the channels there is a pull-up resistor for each single channel line, and a decoupling capacitor between the voltage supply and the ground. Figure 3 depicts the slave board prototype used during the experiments. The memory power supply is provided through an external variable source thus allowing to perform diagnosis under different supply conditions.

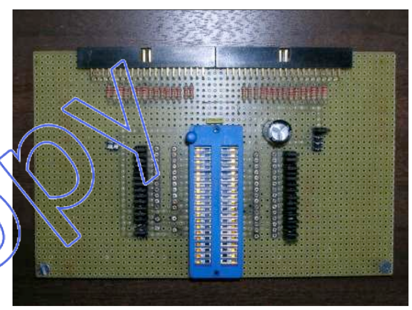

Fig. 3. Slave board for supporting DIP socket based SRAM circuits

B. Diagnosis software implementation

The full test introduced in Section III has been coded into a C program running on the Microblaze processor, and implemented resorting to four main test functions:

- *WriteByteOnMemory(address,data)*: write a byte to the specified address;
- *WriteBitOnMemory(address,bit,data)*: write a single bit value into a specific position in the memory byte specified by address;
- *ReadByteOnMemory(address)*: read a byte from the specified address;
- *ReadBitOnMemory(address,bit)*: read a single bit value from a specific position in the memory byte specified by address.

The two functions WriteBitOnMemory and ReadBitOnMemory allow us to avoid the use of data background se-

TABLE IV
DATA AND ADDRESS BUS FAULT DICTIONARY (MORE THAN TWO FAULTS PER TEST)

	M_0	M_1	M_2		M_3	M_4		M_5	M_6		M_7	M_8		M_9
Fault type	-	-	R_0	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}
SAF(0)$_{addr}$	0	0	1	0	0	1	0	0	1	0	0	1	0	0
SAF(1)$_{addr}$	0	0	1	0	0	1	0	0	1	0	0	1	0	0
ABF$_{addr}$	0	0	1	0	0	1	0	0	1	0	0	1	0	0
OBF$_{addr}$	0	0	1	0	0	1	0	0	1	0	0	1	0	0
SAF(0)$_{data}$	0	0	0	1	1	1	0	0	0	1	1	1	0	0
SAF(1)$_{data}$	0	0	1	0	0	0	1	1	1	0	0	0	1	1
ABF$_{data}$	0	0	0	1	1	1	0	0	0	1	1	1	0	0
OBF$_{data}$	0	0	1	0	0	0	1	1	1	0	0	0	1	1

quences, useful to detect intra-word coupling faults in word oriented memories [13].

The execution of the diagnostic algorithm involves collecting and storing data to be later analyzed. This represents a main issue due to the limited amount of memory available on the board. The proposed solution is to utilize an optimized data structure composed of a six elements vector, defined according to the C declaration in Figure 4.

```
struct Node
{
     int add;           // faulty cell address
     int bit;           // faulty cell bit
     int signature;     // fault signature
};
```

Fig. 4. Diagnosis data structure

Each node contains three fields used to store the address of the faulty cell, the position of the faulty bit within the memory word, and the march syndrome of the fault. The vector is divided in two groups in order to reserve space both for the first and for the second part of the test algorithm: the first three cells are used to store the diagnostic information when an ascending address order is used, while the other three cells are reserved for the part of the diagnostic algorithm using the descending addressing order. The *signature* field uses an integer value to efficiently code the syndrome of the fault. The i^{th} bit of this number is set to 1 if the i^{th} read operation detects the fault.

The algorithm used to store the information into the vector during the test execution is reported in Alg. 1. It is invoked every time, during the test execution, a fault is detected by a read operation. Four parameters (*AO*, *full*, *position*, and *exit*) are considered to identify the correct vector element where information has to be stored. The *AO* flag contains the considered address order, the *full* flag is set if the first half of the vector is full, *position* stores the first free entry in which the information must be written, and *exit* is set if there is a free entry. The algorithm checks for an existing entry in the first three positions of the vector for the same faulty address and faulty bit if the ascending address order is used or the descending one is used but the *full* flag is disabled (lines 1-2). If a valid position exists, i.e., variable *entry* contains a valid position in the vector, the signature field is updated. Otherwise, if an empty position is available, the information is stored (lines 3-7). Similarly, if the descending address order is used and the *full* flag is set, the same search is performed among the last three entries of the vector (lines 8-15).

Algorithm 1 Algorithm for storing diagnosis information

1: **if** ($AO = \Uparrow$) OR (($AO = \Downarrow$) AND ($full = 0$)) **then**
2: *entry* ← check_first_3_positions(fault$_{add}$, fault$_{bit}$);
3: **if** (($entry < 0$) AND ($exit = 1$)) **then**
4: store(*position*);
5: **else**
6: update(*entry*);
7: **end if**
8: **else**
9: *entry* ← check_last_3_positions(fault$_{add}$, fault$_{bit}$);
10: **if** (($entry < 0$) AND ($exit = 1$)) **then**
11: store(*position*);
12: **else**
13: update(*entry*);
14: **end if**
15: **end if**

The proposed data structure and allocation algorithm allow us to use a single vector location for 23 different types of memory array faults, two locations in case of BFs between two cells of the memory array, and more than three cells in case of SAFs or BFs into the address or data buses. This allows to perform diagnosis, using very limited resources and therefore low-cost hardware.

C. Experimental Results

In order to validate the proposed architecture, experiments were performed for a set of different SRAM circuits characterized by different size and internal organization of the memory array. The set is composed of 4 different SRAM circuits from different manufacturers:

- *Cypress CY7C128A-45PC*: 2048 words of 8-bit organized as an internal array of $128 \times 16 \times 8$ cells;
- *Cypress CY7C185-20PCX*: 8192 words of 8-bit organized as an internal array of $256 \times 32 \times 8$ cells;

- *Nec μPD43256BCZ-70LL*: a CMOS SRAM circuit with 32768 words of 8-bit;
- *Nec μPD431000ACZ-70LL*: a CMOS SRAM circuit with 131072 words of 8-bit.

Each circuit was connected to the system through the slave board, using a supply voltage equal to 5.00V. The FPGA has been programmed to use 5 GPIO channels to access address, control and data buses of the external SRAM circuit.

During the experiments all faults detectable by the modified march test C- have been injected both via hardware injection (address and data bus faults), and software injection (memory array faults). In this case, the faulty behavior has been simulated forcing a wrong result during the read operations according to the specific fault type. All faults have been correctly identified and diagnosed according to the specification of the test algorithm.

In addition, using the same board, we performed a set of experiments to test the memory behavior with decreased voltage supply simulating stand-by conditions minimizing power consumption. In fact, when the memory is not used, it is not necessary to supply it with the nominal voltage. A minimum voltage level is enough to retain the stored data.

By simply adding additional software functionalities the proposed platform was efficiently exploited to identify suitable voltage levels for stand-by conditions. Figure 5 summarizes the performed experiments. The test is divided into three phases: (i) using nominal voltage conditions a predefined pattern is written into the memory (writing phase), (ii) the memory is then forced into a stand-by phase where the voltage supply decreases at a minimum level, and finally (iii) after resuming to nominal voltage supply level the content of the memory is checked (reading phase) to understand if data were lost during the stand-by phase.

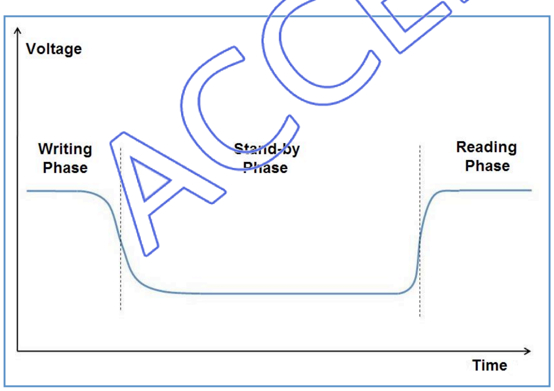

Fig. 5. Test with supply voltage variations

By running the proposed procedure on the four memory models with different voltage conditions, we have been able to identify the following minimum stand-by supply voltage for the four considered memory models:

- *Cypress CY7C128A-45PC*: 1.30V;
- *Cypress CY7C185-20PCX*: 0.82V;
- *Nec μPD43256BCZ-70LL*: 0.53V;
- *Nec μPD431000ACZ-70LL*: 0.52V.

V. Conclusion

The continuous improving of the manufacturing process poses several threats in the quality of integrated circuits, and in particular of memory devices. In this paper a low-cost and flexible architecture for SRAMs testing and diagnosis is described. The architecture allows to perform test and diagnosis of physical memory circuits storing diagnosis information in terms of location and fault type. The architecture is based on a reconfigurable hardware system which presents itself as an efficient and flexible low cost architecture if compared with standard ATE solutions. The flexibility of the platform has been efficient exploited to test and diagnose different models of commercial SRAMs, and to identify stand-by conditions to allow reducing power consumption. Moreover, being the full diagnosis process based on a software running on an embedded processor the same approach can be reused whenever memories are embedded in complex systems including a microprocessor, thus implementing Software-Based Self-Diagnosis solutions.

References

[1] International technology roadmap for semiconductors. [Online]. Available: http://www.itrs.net/ [Last Access: September 1, 2009]

[2] A. J. van de Goor, *Testing Semiconductor Memories: theory and practice*. John Wiley and Sons, Inc, September 1991.

[3] G. Harutunyan, V. Vardanian, and E. Zorian, "A march-based fault location algorithm with partial and full diagnosis for all simple static faults in random access memories," in *IEEE Design and Diagnostics of Electronic Circuits and Systems, 2007. DDECS '07*, 2007, pp. 1–4.

[4] V. Vardanian and Y. Zorian, "A march-based fault location algorithm for static random access memories," in *Memory Technology, Design and Testing, 2002. (MTDT 2002). Proceedings of the 2002 IEEE International Workshop on*, 2002, pp. 62–67.

[5] T. Bergfeld, D. Niggemeyer, and E. Rudnick, "Diagnostic testing of embedded memories using bist," in *Design, Automation and Test in Europe Conference and Exhibition 2000. Proceedings*, 2000, pp. 305–309.

[6] D. Niggemeyer and E. Rudnick, "Automatic generation of diagnostic march tests," in *VLSI Test Symposium, 19th IEEE Proceedings on. VTS 2001*, 2001, pp. 299–304.

[7] J.-F. Li, K.-L. Cheng, C.-T. Huang, and C.-W. Wu, "March-based ram diagnosis algorithms for stuck-at and coupling faults," in *Test Conference, 2001. Proceedings. International*, 2001, pp. 758–767.

[8] J.-F. Li and C.-D. Huang, "An efficient diagnosis scheme for random access memories," in *Test Symposium, 2004. 13th Asian*, 2004, pp. 277–282.

[9] G. Harutunyan, V. Vardanian, and Y. Zorian, "An efficient march-based three-phase fault location and full diagnosis algorithm for realistic two-operation dynamic faults in random access memories," in *VLSI Test Symposium, 2008. VTS 2008. 26th IEEE*, 27 2008-May 1 2008, pp. 95–100.

[10] A. Ney, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, and M. Bastian, "A history-based diagnosis technique for static and dynamic faults in srams," in *IEEE International Test Conference, 2008. ITC 2008*, Oct. 2008, pp. 1–10.

[11] A. J. van de Goor and Z. Al-Ars, "Functional memory faults: A formal notation and a taxonomy," in *Proc. 18th IEEE VLSI Test Symposium (VTS'00)*, Montreal, Canada, Apr.30–May3, 2000, pp. 281–289.

[12] C. Giaconia, A. Di Stefano, and G. Capponi, "Reconfigurable digital instrumentation based on fpga," in *3rd IEEE International Workshop on System-on-Chip for Real-Time Applications*, 2003, pp. 120–122.

[13] A. van de Goor and I. Tlili, "A systematic method for modifying march tests for bit-oriented memories into tests for word-oriented memories," *IEEE Trans. Comput.*, vol. 52, no. 10, pp. 1320–1331, 2003.