Applying Memory Test to Embedded Systems

César Augusto M. Marcon[1], Alexandre de Moraes Amory[1], Marcelo Lubaszewski[1], Altamiro Amadeu Susin[1], Ney Laert V. Calazans[2], Fernando Gehm Moraes[2], Fabiano Hessel[2]

[1]*Universidade Federal do Rio Grande do Sul (PPGC-UFRGS), Porto Alegre – RS – BRASIL*
[2]*Pontifícia Universidade Católica do Rio Grande do Sul (PPGCC-PUCRS), Porto Alegre – RS – BRASIL*
marcon@inf.ufrgs.br

Abstract

Embedded systems designers are faced today with stringent performance requirements on, e.g. power and area consumption. This work analyses power consumption and execution time of memory test algorithms, considering both hardware and software implementations. Software implementation is carried experimentally on four different processors. Comparative results are presented to guide the designer to select the best solution for applications.

1. Introduction

Embedded systems have a broad spectrum of requirements depending largely on the applications specificity. For applications on mobile telephony, for example, low power dissipation leads to an increase of telephone set autonomy. For applications of high-speed data communication, the execution speed of data manipulation algorithms is an essential requirement. For most embedded applications, silicon area consumption and number of input/output pads are requirements that guide the design flow. In general, embedded systems design must always be concerned with area consumption and power dissipation. Area consumption directly affects the system production cost. Power dissipation, on the other hand, is an additional requirement in the system total cost, due to the power dissipater sizing. Besides, power is a limiting factor for all battery-powered systems. For some embedded systems, the execution speed is also a limiting design requirement, mainly to embedded systems that operate in *critical real time applications* [1].

An embedded system design flow must include verification steps to guarantee its correct operation. Memory circuits are usually present in embedded systems and are very important in the power and area budgets, implying that memory test procedures are one of the main concern for system verification.

Memory verification can be executed during the design flow, manufacturing or system operation. The verification during the design searches for errors induced for an inadequate specification. The verification during the manufacturing process looks for coarse errors making impracticable the system use or at least restricting its applicability. On the other hand, verification during system execution seeks to evaluate the correct operation with regard to transitory or permanent faults. Device fatigue is generally responsible for such faults.

Two test classes perform the verification during system operation for random access memories (RAM). The first one is the *destructive tests class*, which is not concerned with keeping the information integrity; the second one is the *non-destructive tests class*, which preserves the original memory contents after test.

The destructive tests class allows greater flexibility to test various memory faults. On the other hand, to perform such tests during system operation implies data loses. The non-destructive class is less flexible and implies extra care with the system global design, because these tests are designed to be executed during system operation. Both test classes can be integrated on systems that will be tested, i.e. BIST [2].

The motivation for this work is the characterization of memory test algorithms in hardware/software domains, having as target embedded systems applications.

This work addresses the evaluation of destructive and non-destructive memory tests. These costs are area consumption, power dissipation and execution speed. Section 2 presents the algorithms used for memory tests. Section 3 presents the implementations of the algorithms. Section 4 shows a comparative analysis of the results and Section 5 presents some conclusions.

2. Memory Test

Several types of faults can occur in memory devices, as stuck-at-faults, transitions faults, coupling faults and address decoder faults.

A *stuck-at fault* (SAF) happens when a cell or line is always at the value 0 or 1, and it is not possible to modify this value. Tests to detect and locate SAFs must satisfy the following requirement: "from each cell or line a 0 and a 1 must be read" [3].

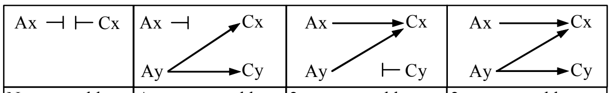
A *transition fault* (TF) is a special case of SAF. It occurs when a write in a cell or line does not generate a correct transition. A test that allows detecting and locating all the TFs must submit all the cells or lines to transitions 0 → 1 and 1 → 0, and all cells or lines must be read after each transition and before being submitted to a new transition.

A *coupling fault* (CF) involves two or more cells. This fault occurs when a writing operation, which induces a cell transition, modifies the content of another cell. Tests to detect and locate all the CFs must satisfy the following requirement: "For the entire cell that is a coupled cell, each cell should be read after a series of transitions that could cause coupling faults, with the condition that the number of transitions in the coupled cells be odd" [3].

Address decoder faults (ADF) occur when one or more of the four following faults occur.

1. For a given address, no cell will be accessed;
2. There is no address accessing a given cell;
3. With a certain address, multiple cells are accessed;
4. A certain cell is accessed by more than one address.

Tests to detect and locate all the ADFs must test all fault combinations illustrated in Figure 1, for all the cells.



No address accesses a cell	An address accesses more than one cell	2 or more addresses access the same cell, and there are cells never accessed	2 or more addresses access the same cell
Faults 1 or 2	Faults 1 or 3	Fault 2 or 4	Fault 3 or 4

Ax and Ay are addresses;
Cx and Cy are cells addressed by Ax and Ay, respectively.

Figure 1 – Combinations of address decoder fault

2.1 Memory Test Algorithms

This work analyses 3 March algorithms, which have a destructive behavior, and an algorithm that preserves the memory contents, named in this work Ndestructive. The following notation is used to describe the memory test algorithms:

- Curly brackets determine the beginning and the end of the algorithm;
- An algorithm is composed of subalgorithms separated by semicolon;
- The ↑ and ↓ arrows represent the access order to the memory elements, either increasing or decreasing addressing;
- Parentheses represent actions performed in the same memory address. These actions are separated by comma;
- A subalgorithm is composed of a direction arrow and actions delimited by parentheses;
- The actions are: r, r_s, w0, w1 and w_s, where:
  - r represents a memory reading operation. Such read implies the comparison to the value previously written;
  - w0 – write a pattern into memory (0 in this work);
  - w1 – write the complemented pattern into memory;
  - r_s represents the reading and saving of a memory cell or line;
  - w_s represents the writing of the last saved value by the r_s action.

2.1.1 March Algorithms

A *March algorithm* consists of a sequence of *March elements*. A March element contains reading and/or writing that are applied to all **n** memory cells in increasing (for 0 until **n**-1) or decreasing (for **n**-1 down to 0) order. The class of March algorithms is very useful for memory BIST implementation; therefore, the complete address range is executed sequentially many times. This operation allows the use of a simple counter, counting up or down depending on the memory access direction. The complexity of the selected March algorithms is O(**n**). However, this complexity measurement is not sufficient to evaluate the algorithms execution speed.

2.1.1.1 MATS

The *Modified Algorithmic Test Sequence* (MATS) is explained in Figure 2 and represented in Figure 3. It is the simpler possible March algorithm with reasonable fault coverage. The algorithm detects all the SAFs in memory cell and in the write and read logic. It also finds all ADFs, if the cell placement corresponds to the address sequence, i.e. the access to an address **i** can affects the cell addresses **i-1** and **i+1**. For the ADFs it is assumed that the reading of two or more coupled cell will result in the "or function" of the contents of these cells. MATS requires 4 * **n** memory accesses.

```
      (I)                (II)                (III)

for i ←0 to n-1   for i ←0 to n-1   for i ←0 to n-1
  cell[i] ← 0       if cell[i] ≠ 0    if cell[i] ≠ 1
                      ERROR             ERROR
                    cell[i] ← 1
```

Figure 2 – MATS algorithm description

(I) (II) (III)

{ ↑(w0);   ↑ (r, w1);   ↑ (r) }

Figure 3 – MATS algorithm representation

2.1.1.2 March C Algorithm

The March C algorithm is used to test coupling fault of unlinked cells. Moreover, this algorithm detects SAFs and TFs, because all the cells are alternately read in 0 and 1 states. The March C algorithm requires 11 * **n** memory accesses.

{ ↑(w0); ↑(r, w1); ↑(r, w0); ↑(r); ↓(r, w1); ↓(r, w0); ↓(r) }

Figure 4 – March C algorithm representation

2.1.1.3 March B Algorithm

March B is a test algorithm that detects SAFs, TFs, ADFs and CFs. March B also detects all the combinations of SAFs, TFs, ADFs and CFs. Figure 5 illustrates the March B algorithm. This algorithm requires 17 * **n** memory accesses.

{↑ (w0); ↑ (r, w1, r, w0, r, w1); ↑ (r, w0, w1); ↓ (r, w0, w1, w0); ↓ (r, w1, w0);}

Figure 5 – March B algorithm representation

2.1.2 Non-Destructive Algorithm

There are many algorithms for non-destructive test implementation as it is presented in [2]. This work uses a minimum non-destructive algorithm, namely NDestructive, which essentially test SAFs. The algorithm, depicted in Figure 6, saves the tested cell value while it performs some actions to test the cell. The NDestructive algorithm requires 6 * **n** memory accesses.

{ ↑ (r_s, w0, r, w1, r, w_s) }

Figure 6 – NDestructive algorithm representation

3. Target Components of Memory Test

This Section presents the architectures chosen for the comparison of memory test implementations taking into account software and hardware components.

3.1 Memory Test Implementation with Hardware

The hardware implementation of the memory test algorithms were described with VHDL, creating a common interface for all tests, such as described in Figure 7. The difference between two algorithms is only the control part, while auxiliary elements, used for memory communication and control and data verification keeps unchanged. The advantage of this approach is the code reuse: only the description part that changes from one algorithm to the other needs to be evaluated.

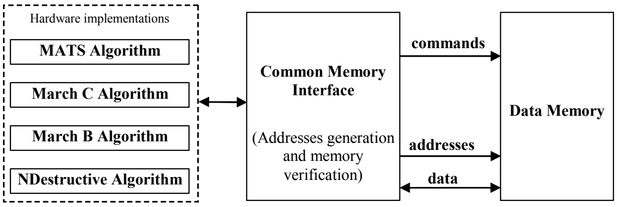


Figure 7 – Memory test implementations with hardware

3.2 Memory Test Implementations with Software

Four IP soft-core processors were chosen to analyze software implementations: 8051, MIPS, FemtoJava and R8. Two of them are commercial processors and two are academics. The choice of these processors has two reasons: the first one is the availability of the VHDL description, and the second one is that they are target to embedded systems. These VHDL descriptions allowed an accurate comparison of the architectures with respect to execution speed, area consumption and power dissipation. Table 1 summarizes the processors features.

Table 1 – Features of the processors used to run memory test algorithms

MIPS	Commercial processor with load-store architecture and pipeline with three stages. Supports interrupts and all MIPS-I user mode instructions, except unaligned load and store operations. The processor word is 32 bits long. IP-core available at http://www.opencores.org/projects/mips.
8051	The Intel 8051 is an 8-bit micro-controller. This micro-controller is able to address 64 Kbytes of program and 64 Kbytes of data memory. To be Intel 8051 cycle compatible, the IP-core available at http://www.cs.ucr.edu/~dalton/i8051/i8051syn was partially modified.
FemtoJava	Academic microcontroller [4] based on stack architecture. It was developed for embedded applications, running a subset of java bytecodes. Address/data is 16 bits long. IP-core is available at http://www.inf.ufrgs.br/~gme.
R8	Academic processor [5], with load-store architecture. Low complexity control unit. 16 general-purpose register. Address/data is 16 bits long. Soft-core, including specification, several implementations and software development tools are available at http://www.inf.pucrs.br/~gaph.

Memory test algorithms are stored in the program memory of each processor. The memory to be tested is the data memory. The processor job is reading instructions sequence dictated by the algorithm and generating addresses and data for the data memory. We presume that instruction memory is previously tested with similar approach to [6].

4. Evaluation Results

This Section presents the comparison of area consumption, power dissipation and execution speed for the test algorithms developed to hardware and software implementations, considering three memory sizes. The chosen memory sizes are 128, 1024 (1K) and 65536 (64K) words. The reduced memory size reflects the small memory sizes found in embedded systems.

4.1 Methods Used to Results Comparison

The four memory test algorithms (MATS, March C, March B and NDestructive) have each one five implementations (one for each processor and one for hardware), which are evaluated in this Section. For hardware implementations, all memory tests were described with VHDL RTL. For software implementations, the assembly generation was performed manually. The reason for this approach is that memory tests compilation cannot be done with optimisations switch; because of the compiler removes some sequences of instructions like, for example, double write at the same memory address. Without optimisations, the generated code and the test execution time were four times bigger than the manual implementation.

The availability of VHDL soft-core of all processors allowed the comparisons of power dissipation, area consumption and execution speed, taking into account the available devices by the synthesis tools.

The execution speed of the memory tests implemented in software was evaluated with Active-HDL™ simulator. For each processor was implemented a VHDL test-bench able to read text files containing the object-code corresponding to each test algorithm. Each test-bench was compiled together with the correspondent VHDL processor code. The simulation flow is depicted in Figure 8. The evaluation process of hardware implementations was similar. However, instead of having a file describing the object-code program, the algorithm was described in a VHDL module, one module for each test algorithm. For all the implementations were considered the number of clock cycles and the operation frequency reported by the simulation and synthesis tools.

The evaluation of power dissipation was done using the Quartus™ synthesis tool of Altera and the ModelSim™ simulator. The Quartus was used to synthesize all the implementations for a common device, which was the APEX20KE (APEX20K200EFC484-2X) of Altera. The ModelSim, in turn, generates files with signal transitions produced by simulation of switching activity. These files are used by Quartus to evaluate power dissipation carried out by synthesis step.

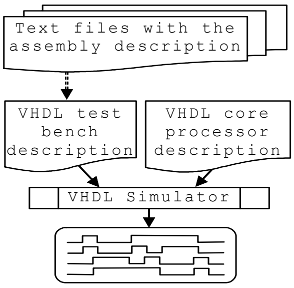

Text files with the assembly description
VHDL test bench description
VHDL core processor description
VHDL Simulator

Figure 8 – Evaluation of execution speed for software implementations

The area consumption evaluation was performed with Leonardo Spectrum™ synthesis tool target to a 0.35 micron TSMC ASICs standard cell library. The synthesis was executed with typical values and low effort.

4.2 Area Consumption Analysis

Table 2 and Table 3 show processors and hardware implementations area, respectively. FemtoJava processor presented the smaller area consumption, while MIPS had the biggest area occupation. Hardware implementations achieved a great variation of area consumption between different algorithms. An important aspect is that the area consumption of the smaller processors is six times bigger than the average of hardware implementations. Since processors are present in mostly embedded systems, their area is not really an overhead; only the memory used to store test code counts. The aim of this comparison is to select the well-suited processor regarding the memory test procedures.

Table 2 – Comparison of processors area consumption (in equivalent gates)

MIPS	8051	FemtoJava	R8
19,243	6,512	4,023	4,742

Table 3 – Comparison of hardware area consumption (in equivalent gates)

Algorithm	128 words	1K words	64K words
MATS	380	403	471
March C	603	628	718
March B	681	706	795
NDestructive	396	409	477

Figure 9 illustrates the number of bytes for algorithms implemented in software. Figure 9 also shows that MIPS expend more bytes of code, it happens manly due to the instruction memory size. On the other hand 8051 consumes less bytes because of the processor instruction word is 8 bits long.

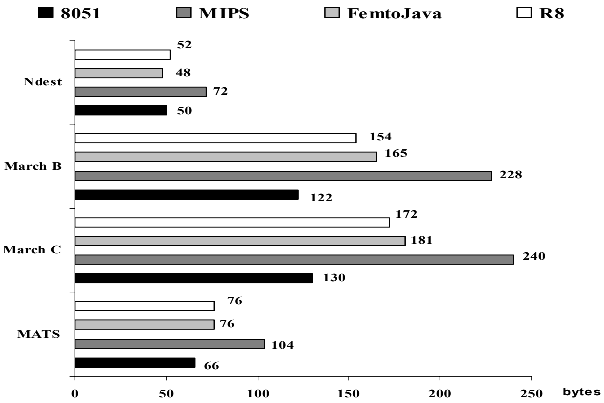


Figure 9 – Comparison of code size

4.3 Execution Speed Analysis

The tests execution speed for all the implementations showed an almost linear behavior with the memory size variation.

Figure 10 depicts that hardware implementations always achieve the smaller clock cycle result for all test algorithm. Figure 10 also shows that MIPS has the best performance if compared to others processors due to its pipeline architecture. 8051 has the worst performance because of its CISC architecture, which implements many instructions with more than 12 machine cycles. FemtoJava has lower performance either due to its stack architecture. The MIPS showed to have an average of cycles per instruction (CPI) equal to 1.7, 8051 has a CPI near of 14.9, while R8 has a CPI near of 3.8; the CPI of FemtoJava is 5.9.

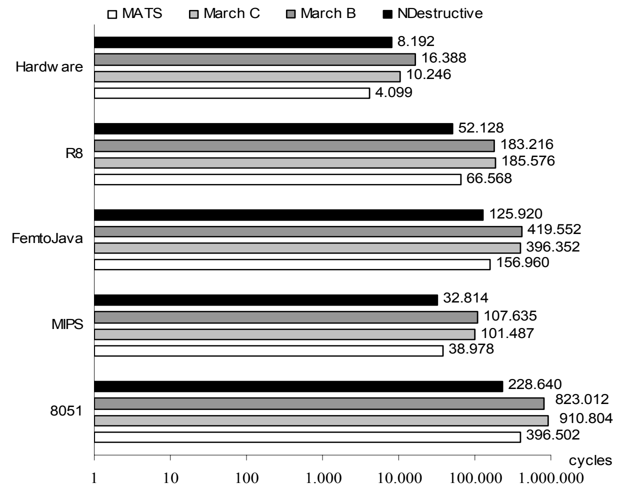


Figure 10 – Comparison of execution time for a 1K words memory (logarithmic scale)

The clock frequency, estimated by the synthesis tool, is shown in Figure 11.

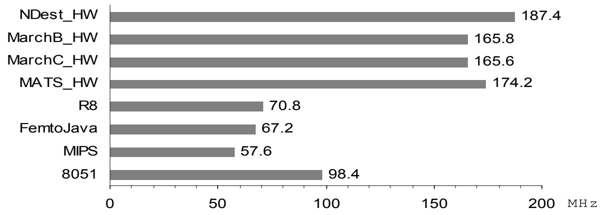


Figure 11 – Clock frequency obtained from synthesis

Applying the estimated frequency to all implementations, as it is depicted in Figure 12, the hardware execution speed is still bigger, which gives to the hardware the ability to execute fast memory test.

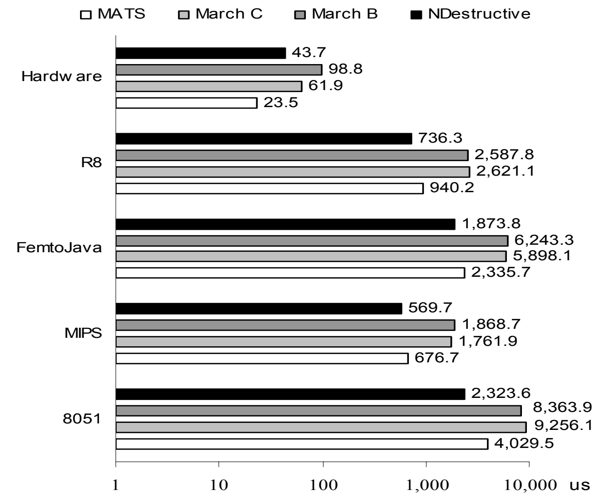


Figure 12 – Comparison of execution time for a 1K words memory (logarithmic scale)

4.4 Power Dissipation Analysis

Figure 13 presents the result of the power consumption comparison in mW, obtained form association of Quartus and ModelSim tools. The test-bench input frequency for all implementations was normalized to 10 MHz.

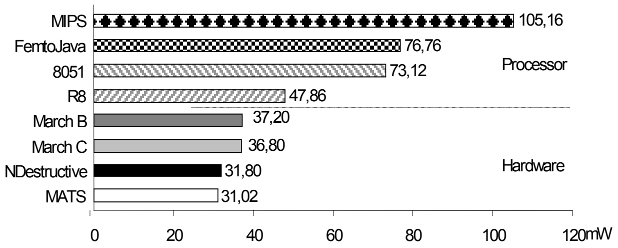


Figure 13 – Power dissipation (in mW)

This work performs power analysis not taking into account the memory power consumption, just the test circuit. Therefore, it was verified that the tested memory size scale does not affect significantly the power consumption of hardware and software implementations. The reason of this is that the signals transitions of the test circuits depend weakly on the memory size.

Figure 13 highlights that hardware implementations have around of 20% of power consumption differences due to the algorithm variations. However, software implementations showed little variation on the average power consumption by modifying the test algorithms. This occurs because the four test algorithms use similar processor resources, i.e. registers bank, arithmetical and logical operations. This explanation is corroborated by *code coverage* analysis, available in Active-HDL and ModelSim simulators. The code coverage analysis shows the number of times that a code line is executed during a simulation. Power dissipation is function of $\alpha.C.V^2.f$, where α, C, V and f represent the switching activity, capacitance, voltage and operation frequency, respectively. For the same C, V and f, the power dissipation depends only on α. In turn, α does not depend on the code execution order. Indeed, it depends on the number of times that the code is executed and the input data for each code line. For all test algorithms, it was observed that the input data was very similar (test pattern are FFFFh and 0000h) and the code coverage shows less than 3% of differences among algorithms. This roughly demonstrates that the power consumption did not have to change from one algorithm to the other.

Proportional analysis concerning power dissipation and used area shows that the implementations of hardware have higher costs of power dissipation if compared to software, mainly for the R8 processor. This demonstrates that for processors only small part of transistors is switching during the memory test.

The absolute value of power dissipation of the memory access is not presented in this work. Instead of, it is presented a comparative evaluation, which allows qualifying the implementation results. The first aspect is observed in Figure 11: hardware implementations allow greater clock frequency and consequently more power dissipation. The second aspect is that hardware implementations were performed in such a way to allow practically one memory access per cycle. It means that the data memory is always been read or written implying power dissipation in all clock cycles.

Software implementations imply power dissipation for data and instruction memory. However, it does not necessarily imply more system power dissipation, for some processors the power dissipation caused by data and instruction memory access is smaller than the power dissipation caused by data memory access in hardware implementations. MIPS processor has CPI equal to 1.7, 32 bits of instruction memory and, in average; it is necessary 6 instructions for each data memory access. It means that for each 10 cycles (1.7 * 6) there are 6 instruction memory accesses and 1 data memory access. Considering that the instruction memory consumes two times more than the data memory (double word size), the average power costs due to memory access is 13[1] for 10 cycles while for hardware the average cost is just 1 for cycle. Therefore, memory tests with MIPS expend around of 30% more power than memory tests with hardware. However, hardware operation frequency is 3 times greater than MIPS and the memory reading operation dissipate less power than the memory writing operation. These two reasons lead to the conclusion that MIPS memory power dissipation is smaller than hardware memory power dissipation. The power dissipation difference is greater if hardware implementations are compared to others processors without pipeline and smaller instruction memory, like 8051 and R8. For example, the CPI of R8 is 3.7, 16 bits of instruction memory and it is necessary 8 instructions for each load/store instruction. It means that each 30 cycles (3.7 * 8) exists 8 instructions memory access and 1 data memory access. In this case, the memories of hardware implementations dissipate around of 230% more power than the software equivalent, without considering the frequency effects.

Finally, Figure 14 shows the processor and hardware power dissipation considering the synthesis estimated clock depicted in Figure 11. It is possible to observe that when it is used the highest hardware frequency, the software power dissipation is proportionally reduced, and in some cases is smaller.

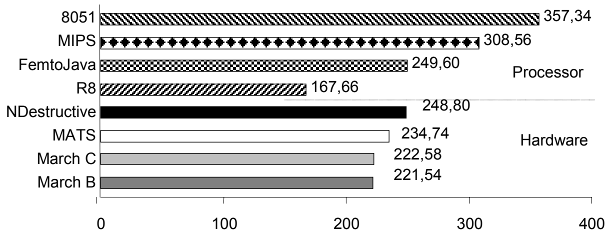


Figure 14 – Power dissipation (in mW) for synthesis estimated clock

5. CONCLUSIONS

The hardware implementations have significant advantage in the execution speed requirement. Software implementations achieve good results with MIPS processor that uses pipeline techniques and bad results with FemtoJava and 8051 processors due to the stack and CISC architecture, respectively.

It is possible to observe that the processors power dissipation has the same magnitude order of the hardware implementations power dissipation, and for a dedicated embedded processor, like R8, the power dissipation is smaller if considered the synthesis estimated clock. This fact is more salient concerning the analysis of power dissipation.

For the software implementations, modification in the memory test algorithms changes insignificantly the processor power dissipation. Another important aspect, related to the power dissipation, is the low deviation due to the memory size changing.

As it was expected, the area consumption analysis shows that the hardware implementations achieved the smaller number of equivalent gates. It happens because of the memory test algorithms are not complex, being able to be implemented with little logic. However, the majority of the embedded systems use some type of processor, independent of having or not memory tests. Thus, if the memory test will be executed in software, the power dissipation and area consumption requirements will be well fulfilled, being the execution speed the only restrictive factor.

6. REFERENCES

[1] C. A. M. Marcon, N. L. V. Calazans and F. G. Moraes – “Requirements, Primitives and Models for Systems Specification”. SBCCI 2002.

[2] M. L. Bushnell and V. D. Agrawal – "Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits". Kluwer Academic, 2000.

[3] A. Van de Goor and C. Verruijt – “An Overview of Deterministic Functional RAM Chip Testing”, ACM Computing Surveys, 1990.

[4] S. A. Ito, L. Carro, R. P. Jacobi - System Design Based on Single Language and Single-Chip Java ASIP Microcontroller. Design, Automation and Test in Europe (DATE '00), 2000.

[5] N. L. V. Calazans, F. G. Moraes, C. A. M. Marcon – “Teaching Computer Organization and Architecture with Hands-On Experience”. 32nd ASEE/IEEE Frontiers in Education Conference. November, Boston, 2002.

[6] Y. Zorian, A. Ivanov – An effective BIST scheme for ROM's. IEEE Transactions on Computers, p.: 646 -653, Volume: 41 Issue: 5, May 1992.

[1] (6 instruction memory access and 7 data memory access)

Session 3 – Fault Tolerant Architectures and Techniques