Design of a Totally Self Checking Signature Analysis Checker for Finite State Machines

M. Ottavi, G. C. Cardarilli, D. Cellitti, S. Pontarelli, M. Re, A. Salsano

Department of Electronic Engineering University of Rome “Tor Vergata”, Italy

da.ce@tiscalinet.it
{ottavi, pontarelli, salsano}@ing.uniroma2.it
{marco.re, g.cardarilli}@ieee.org

Abstract

This paper describes the design of a totally self-checking signature analysis checker to be used to implement self-checking finite state machines. The application of the signature analysis method is studied taking into account trade off criteria concerning area and timing constraints requested in specific applications. In this paper, we propose a novel VHDL realization of this methodology suitable for the implementation of the SSMM for satellite applications described in [1][2]. Finally, we will present a general criterion to evaluate the optimal solution in terms of area overhead between the proposed method and a typical duplication and compare strategy.

1. Introduction

In the field of digital systems design, the growth of the system level integration and of complexity is a trend that is dramatically exploding in recent years. The use of computer aided design (CAD) tools and Hardware Description Languages (HDL) allows the design of systems where the granularity is moving from the gate level to more complex levels like the Finite State Machines (FSM). A straightforward consequence of this trend is the introduction of higher level methodologies to achieve self-checking and fault tolerant systems. In particular, this paper will describe a self-checking signature analysis checker suitable to control the correct operations of a finite state machine. In chapter 2 we will firstly introduce some basic concepts about Finite State Machines, the fault models expected, how they affect the behavior of a FSM and an overview of the methods to achieve self-checking FSMs. In chapter 3 we will introduce the trade-off criteria between the different solutions taking into account the constraints requested by specific applications. In chapter 4 we will then describe the signature analysis checker we developed for the SSMM design. In chapter 5 we will produce the simulation results. Finally, in chapter 6 we will generalize the result of our study to evaluate the optimal solution in terms of area overhead between the proposed method and a typical duplication and compare strategy.

2. Finite State Machines

Finite State Machines are automata with a finite number of states and are also called sequential machines. The main entities of a FSM are the inputs X the outputs Z and the State

S, both the state and the input affect the output. We consider the two main implementations of the FSMs called Moore (figure 1) ad Mealy (figure 2) Machine. In FSMs we can distinguish two combinatorial blocks and a state register. The combinatorial blocks are the Next State Logic (NSL) and the Output Logic (OL). When the register is enabled the next state S(t+1) becomes the actual state S(t) allowing the machine to follow a sequential behavior. In a Moore machine we have:

$S(t+1) = f[X(t),S(t)]$

$Z(t) = f[S(t)]$

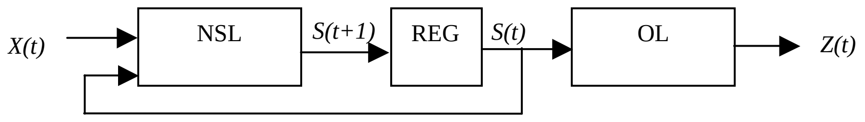

figure 1 Moore FSM

In a Mealy machine we have

$S(t+1) = f[X(t),S(t)]$

$Z(t) = f[X(t),S(t)]$

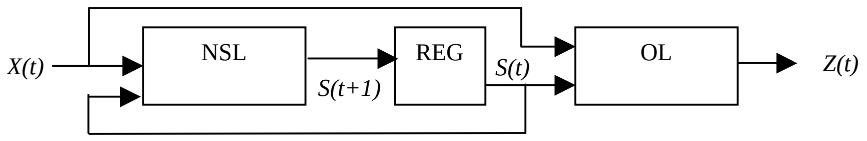

figure 2 Mealy FSM

Can be demonstrated that the two implementations are equivalent. Therefore we will focus our attention on Moore machines without loss of generality.

2.1. Fault Models

The fault modeling is an important step in the design of self-checking circuits. Physical faults occurring in an FSM realized with electronic components can be modeled with changes in the state diagram executed by the state machine [6].
In particular, the changes that can occur in the state diagram can be:

- Generation of a new arc between two states
- Generation of unattended sequences
- Infinite loops in a state

For example, a bit-flip in the state register caused by heavy ion impact, can be modeled by the generation of a new arc between two states, while a fault in the NSL or delay faults on the interconnection lines can generate the occurrence of unexpected sequences.
With this fault model, we can develop a methodology realizing the FSM checking with a certain level of abstraction from the physical implementation.

2.2. Self Checking FSMs

In literature, the proposed methodologies to achieve the self checking property in the FSMs are mainly divided into two approaches: control of the logic blocks (OL, NSL and registers) or control of the state diagram properties. Therefore, we can summarize these approaches as follows:

Control of the logic blocks
1. FSM decomposition and states coding [3]
2. Introduction of monitoring FSM [4]

Control of the state diagram properties
3. FSM states coded with the Signature Invariance Property (SIP) [5] [7]
4. On-line Signature Analysis [6]

3. Trade-Off Criteria

The scope of this work is to implement a monitoring system for FSM that detects unexpected behavior in the evolution of state diagram.
The choice of the methodology depends on a set of parameters under optimization.
The trade-off criteria to be considered can be summarized as follows:

- Hardware overhead of the controlling system with respect to the controlled one
- Latency time in the fault detection
- Requested reliability of the system
- Hardware complexity in the controlling system
- Number of requested modifications to be applied to the controlled system

Often these parameters are in contrast, therefore a trade off must be made to achieve a sub optimal solution. For example, solutions 1 and 2 have in general no latency time but they introduce a consistent overhead. Moreover, solution 1 needs the decomposition of the controlled FSM. Solutions 3 and 4 have lower hardware redundancy but solution 4 introduces latency time in the fault detection and solution 3 needs the modification of the state diagram.
In applications where is important to achieve low hardware overhead and low power a solution based on the signature analysis could be preferred. We investigated the application of this method in the design of some (non time critical) functional blocks of the SSMM system for space applications described in [1].

4. Implementation of a Self-Checking Signature Checker

The state diagram of a FSM is partitioned like described in [5] and the signatures are calculated. For each pre-calculated signature, a two-rail comparator compares it with the run-time calculated signature. The result of comparison is evaluated only in the check point states corresponding to the convergence vertices and the branch ones of the state diagram.
If the run-time calculated signature corresponds with one of the pre-calculated ones the controller will signal that the system is running correctly otherwise an error signal will be produced. To detect the occurrence of infinite loops a downward counter is initialized at the start of each sequence with the value of the to the length of the longer sequence.
If the counter reaches zero before a checkpoint is detected an error signal will be generated.

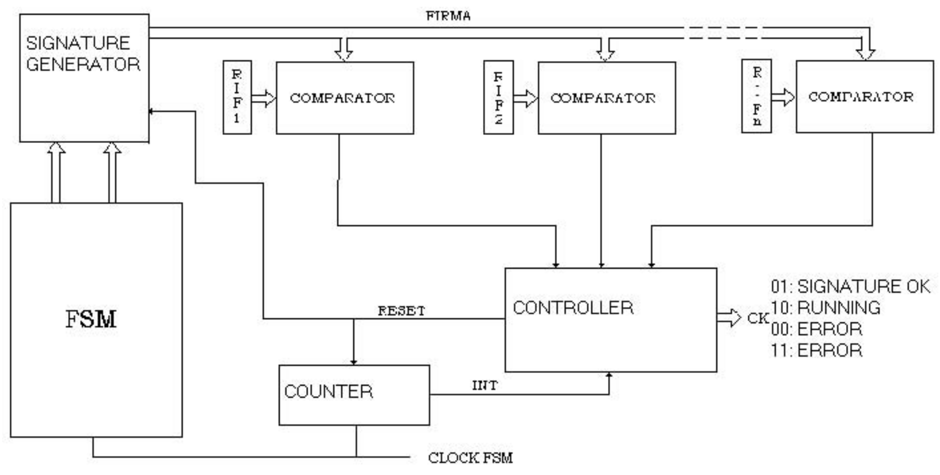

figure 3 signature analysis checker

In figure 3, we show the schematic of the self-checking checker we realized. We can distinguish four main blocks:

- Comparator: this blocks concurrently check the run time generated signature by the signature generator and send the result to the controller
- Signature generator: this block generates the signature of the sequence of states coming from the FSM under test. The signature is calculated by using a linear feedback shifter register.
- Counter: detects the infinite loops like explained above
- Controller: supplies the error signal by controlling the outputs of counter and comparators. The meaning of the signal is explained in the figure.

It has to be noticed that all the functional blocks are self-checking and their outputs are 1 out of 2 codified.

5. **Simulation Results**

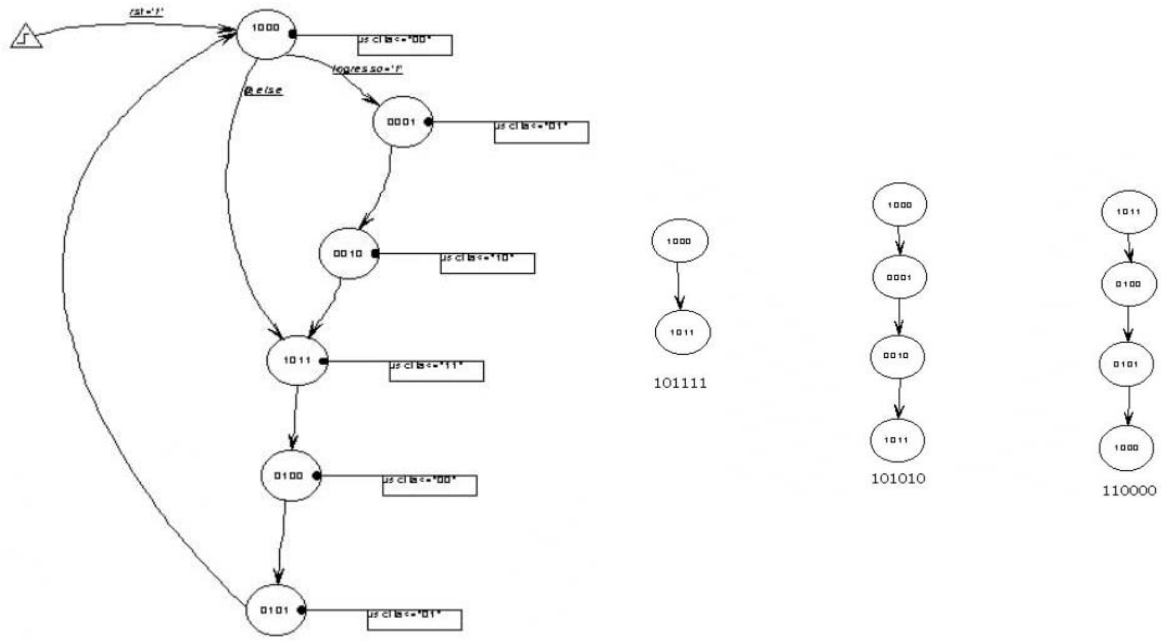

figure 4 example state diagram and sequence decomposition

In this section, we describe the simulation results obtained by the architecture explained above. We take into account an FSM under test (UT) with the state diagram reported in figure 4. In the same figure we also report the decomposition of the diagram in sub-sequences.

In figure 5 we report the outputs generated during the normal operation of the FSM-UT. We can notice that the output OK is always 01 or 10 because no faults occur.

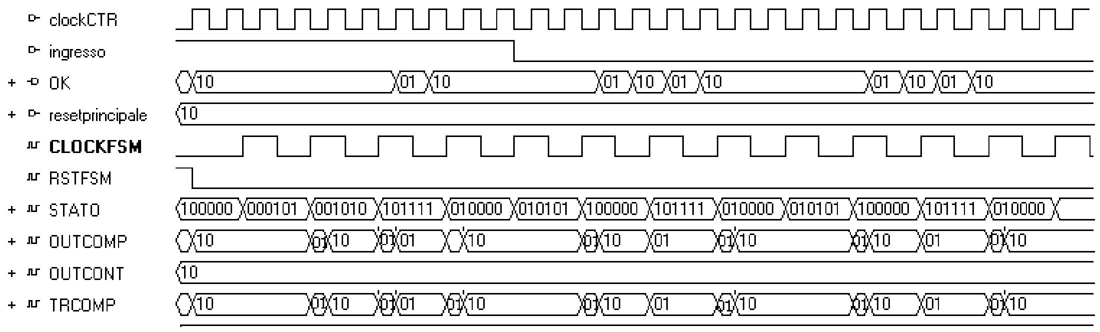

figure 5 waveform during normal operation

The signal OUTCOMP reveals that a signature generated run-time has been recognized in at one of the parallel comparators assuming the value 01 otherwise remaining 10. The controller evaluates this signal only in the checkpoint states.

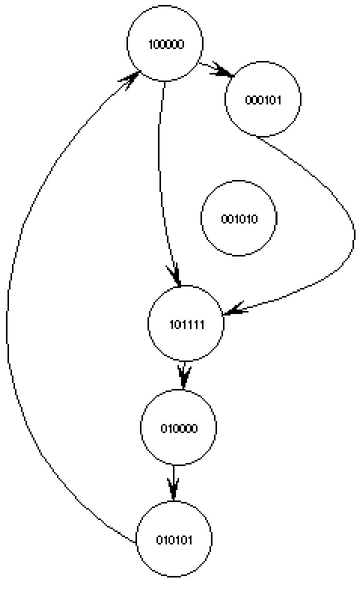

figure 6 erroneous jump

In figure 6, we show the occurrence of a fault generating a new arc between the states 000101 and 101111. In figure 7 we show the simulation results: we can notice that error signal 00 is generated when the STATO signal assumes the value 101111.

In fact the signal OUTCOMP remains 10 (no signature recognized) in a checkpoint and so, an error signal, OK = 00, is generated. Consequently the FSM UT is stopped by setting CLOCKFSM 0 and RSTFSM to 1.

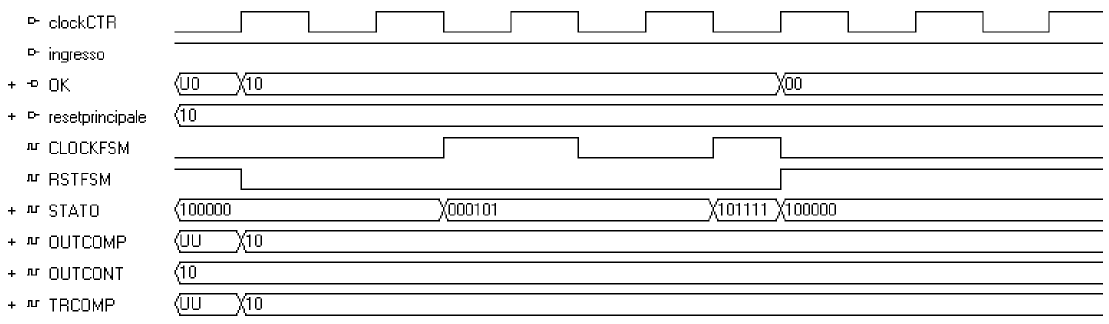

figure 7 fault in the FSM

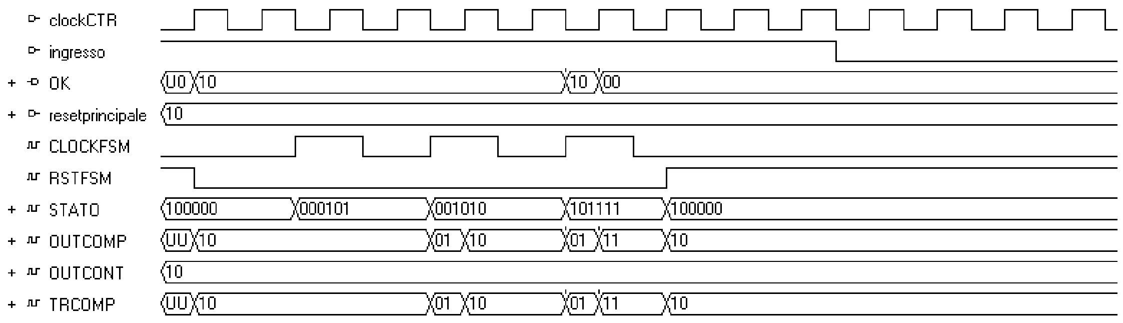

figure 8 fault in a comparator

Finally in figure 8, we show the simulation results concerning the occurrence of a fault in a comparator. When the signal OUTCOMP is 11 the fault is detected in a comparator. With the detection of the fault the controller sets OK at 00 signaling the occurrence of an error and stops the FSM UT.

5. Estimation of Optimal Solution

In order to estimate the area overhead of the proposed checker, it is necessary to have informations about the state diagram of the FSM to be controlled. The estimated results are confronted with the area overhead generated with the duplication and comparison method.
We use the following symbols:

n: number of states
m: number of checkpoints
b: number of state bits
u: number of output bits
i: number of input bits
r: number of signature bits
S: number of sequences

The number of sequences S in a state depends on the number *m* of checkpoints. In fact, the number of sequences in a graph with *m* checkpoints is equal to the sum of outgoing arcs from every checkpoint. Therefore we can approximate the number of sequences in the interval

$m - 1 <= S <= m^2 + (n-m)$

where $m-1$ is the minimum number of sequences and $m^2 + (n-m)$ is the maximum. We can suppose a particular graph composed by $m-1$ checkpoints (source) with only one outgoing arc and one checkpoint that is well of all the generated sequences. The maximum is obtained for a fully connected graph including the loopback on the nodes. In this case, we have m sequences generated by each node, therefore the number of sequences is m^2. If the graph has nodes that are not checkpoints, one sequence must be added to the sum. Because the number of nodes that are not checkpoints is $n - m$, the above formula is demonstrated.
The complexity of the state diagram in function of m is shown in figure 9 where the lighter curve represents the minimum number of sequences and the darker represents the maximum.

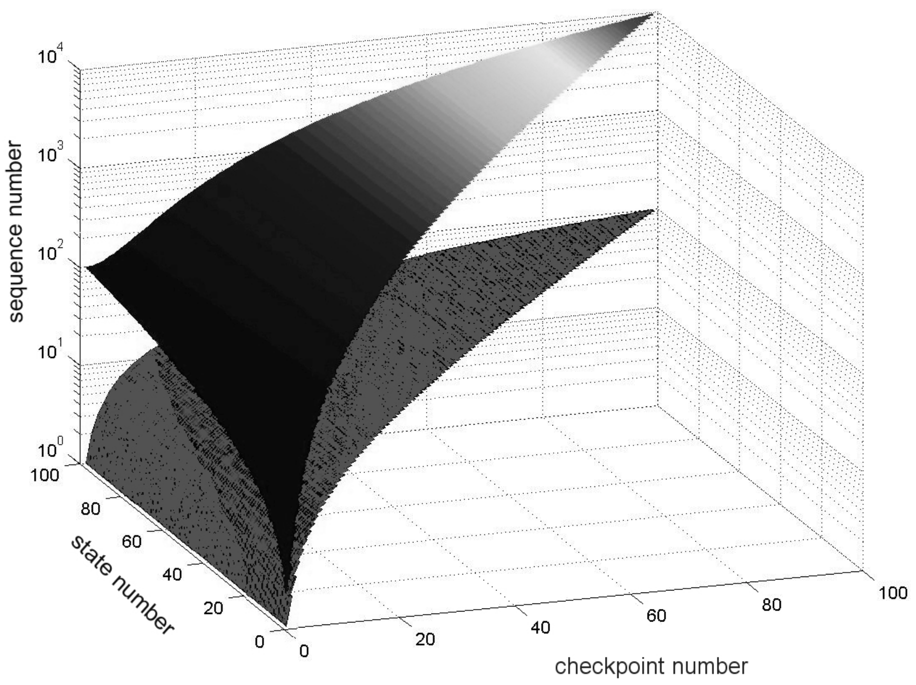

figure 9 estimation of the number of sequences in a state diagram

We estimated the area overhead G (in terms of equivalent gates) introduced with the signature analysis method by using the approximated formula

$$G \sim 6 S r$$

We can see that the overhead depends linearly on the number of sequences and on the number of bits of the signature, therefore, with the signature analysis method, the overhead is strictly depending on the characteristics of the state diagram.

The overhead G_D (in terms of equivalent gates) introduced with the duplication and comparison method can be expressed as

$G_D = G_{Cmp} + G_{NS} + G_{Out} + G_{Reg}$

Where G_{Cmp} is the number of gates of the comparator, G_{NS} is the number of gates of the next state logic of the FSM G_{Out} is the number of gates of the output logic of the FSM G_{Reg} is the number of gates of the state register.
The approximated values for each of these terms were evaluated by using semi-empirical models and by validating them with real implementation data. We obtained:

$$G_{\mathrm{Cmp}} \propto (b+o)$$

$$G_{NS} \approx \frac{(i+b)^2 b}{2}$$

$$G_{Out} \approx \frac{b^2 u}{2}$$

$$G_{\mathrm{Re}g} \propto b$$

We can explain these formulas as follows: the number of gates in the comparator grows linearly with the number of lines to be compared (in this case we compare the outputs and the state bits) a similar behavior is assumed for the number of gates in the registers. We estimated the number of gates of a combinatorial block with *o outputs* and *i inputs*, with a formula derived fitting a set of *Mcnc* benchmarks minimized with MIS [8].

Finally, we compared the overhead results obtained in the two cases and we traced a comparative diagram as shown in figure 10. We show two surfaces representing the number of gates required in function of the number of input bits of the FSM UT and of the number of sequences of the state diagram. The gray surface represents the overhead introduced with the signature checker while the black one represents the duplication method.

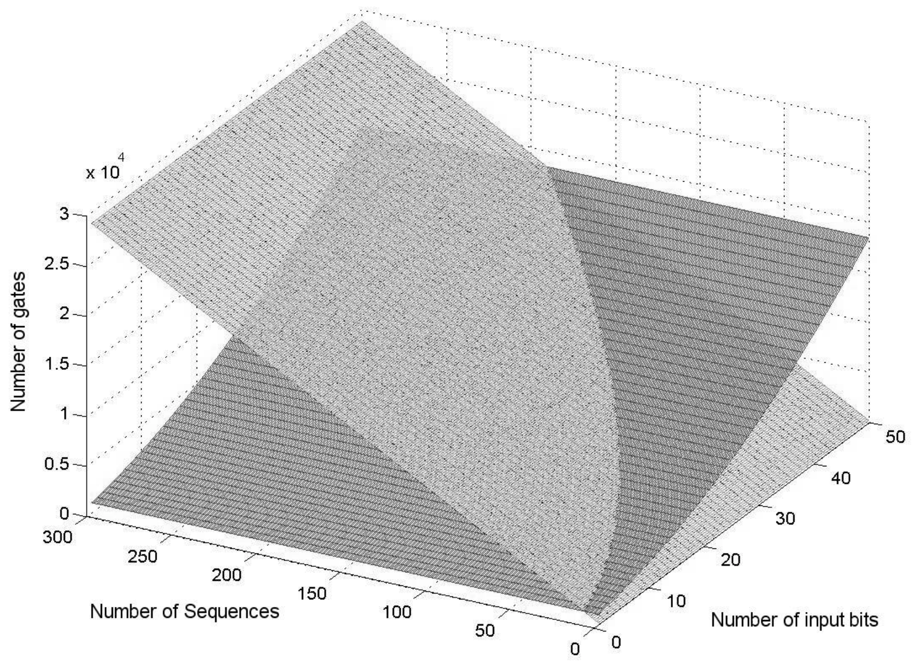

figure 10 overhead comparison

We can see that, where the gray surface is lower than the black one the use of the signature analysis method is more convenient in terms of area overhead.
From this evaluation we can derive an algorithm to visit automatically a state diagram and to evaluate the more convenient approach to achive a self checking FSM from the given state diagram.

7. Conclusions

In this paper we described a study of the methods to achieve fault tolerant Finite State Machines.
We investigated the application of the signature analysis method taking into account trade off criteria concerning area and timing constraints.
A totally self-checking signature analysis checker has been developed to achieve Fault tolerance in the FSMs of the SSMM for satellite applications [1] [2].
The simulation results have been obtained through a VHDL realization of the checker. Finally, we presented the evaluation of the area overhead introduced with this method and we compared it with the area overhead of a duplication and compare checker.

8. References

[1] M. Ottavi, G.C. Cardarilli, P. Marinucci, S. Pontarelli, M. Re, A. Salsano, - "Development of a dynamic routing system for a fault tolerant solid state mass memory" - IEEE International Symposium on Circuits and Systems, ISCAS 2001, Sydney, Australia, May 6 - 9, 2001.

[2] G.C. Cardarilli, P. Marinucci, M. Ottavi, A. Salsano, "A Fault-tolerant 176 Gbit Solid State Mass Memory Architecture", IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, DFT 2000, Yamanashi, Japan, October, 2000.

[3] Busaba, F.Y.; Lala, P.K. – “Design of self-checking interacting FSMs for multiple faults” - Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on Pages: 607 - 611 vol.1.

[4] Parekhji, R.A.; Venkatesh, G.; Sherlekar, S.D. – “Design of monitored self-checking sequential circuits for enhanced fault models” - Test Symposium, 1993., Proceedings of the Second Asian on Pages: 298 – 303.

[5] Robinson, S.H.; Shen, J.P. – “Evaluation and synthesis of self-monitoring state machines”- Computer-Aided Design, 1990. ICCAD-90. Digest of Technical Papers., 1990 IEEE International Conference on Pages: 276 – 279.

[6] El Guibaly, H.; Muzio, J.; El Guibaly, F. – “On concurrent error detection of finite state machine systems” - Circuits and Systems, 1994., Proceedings of the 37th Midwest Symposium on Pages: 213 - 216 vol.1.

[7] Robinson, S.H.; Shen, J.P. – “Direct methods for synthesis of self-monitoring state machines” - Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on Pages: 306 – 315.

[8] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, “MIS: A multiple-level logic optimization system,” IEEE Trans. Computer-Aided Design, vol. CAD-6, pp. 1062–1081, Nov. 1987.