Designing Circuits Detecting Different Types of Faults[1]

I. LEVIN[a], M. KARPOVSKY[b], S. OSTANIN[a], V. SINELNIKOV[a],
[a]Tel-Aviv University, Ramat Aviv, 69978, Israel
[b]Boston University, 8 Saint Mary's Street, Boston, MA 02215, USA
i.levin@ieee.org, markkar@bu.edu, ostanin@fpmk.tsu.ru, sinel@hait.ac.il

Abstract: - This paper presents methods for designing totally self-checking Mealy type synchronous sequential circuits (SSCs). We use implementations of the output and next state functions that are monotonic in state variables. The monotony enables the SSC to react to permanent faults differently than it does to transient faults. If the fault is permanent, the SSC will produce a non-code output after a number of clock cycles (latency), and the non-code output will be detected as an error by the checker. In the case of a transient fault, the SSC is able to become fault free after a number of clock cycles (so-called self-healing time). In the case of a permanent fault the minimal latency is desired. Though the self-healing of SSC after a transient fault takes its time. Thus, requirements to the period of the SSC functioning in cases of permanent and transient faults are contradicting.

We propose a universal architecture of self-checking SSCs enabling to overcome the above contradiction. Namely, this architecture can be adapted either for reduction of the latency of a permanent fault or for increasing the SSC self-healing ability after a transient fault. In the proposed architecture, a method for SSC synthesis is presented. This method is oriented to FPGA based implementation.

Key-Words: - On-line testing, sequential circuits, fault detecting, latency, transient fault, self-healing circuits.

1 Introduction

Two different approaches can be considered for handling an error detected in a circuit concurrently with its functioning. The first approach is based on the immediate marking of the circuit as erroneous when an error is detected. An alternative approach concentrates on increasing the survivability of the circuit, which means, "to give a chance" to the circuit to continue to work during a number of clock cycles after the error detection, and marking it as erroneous only after a steady error is indicated. Sometimes these several clock cycles are sufficient for the recovery of a transient fault and for returning the circuit to its proper functioning.

The first of the above-mentioned approaches has an advantage in the low hardware overhead required for its implementation. The second one offers a way for circuits to survive in cases of transient faults but require a considerable overhead.

In the present paper we investigate architectures that enable circuits to survive without introducing any additional overhead. This property of circuits to become fault free after disappearance of a transient fault is called self-healing.

We deal with microcontrollers described by Finite State Machines (FSMs) and implemented as Synchronous Sequential Circuits (SSCs). Methods for concurrent error detection within the SSC-based microcontrollers have received wide attention, since the control part of a digital system is usually the most critical part from the testability point of view. Complexity and irregularity of the control structure on one hand, and its central role in the functioning of the whole controlled digital system on the other hand, puts the problem of synthesis of self-checking SSCs onto the theoretical and practical agenda.

Usually, when the latency requirements are strict, a memory checker is introduced

[1] This research was supported by BSF under grant No. 9800154

to allow immediate detection of any fault whenever the test vector is applied [1]. Other self-checking SSC architectures which does not contain the memory checker provide either immediate, or the next clock fault detection. The solutions without the memory checker utilize the logic capacity of a combinational part of SSC for the checking of state codewords [2]. The requirement of immediate (or next clock cycle) detection of a fault is reasonable for permanent faults, since such faults have to be detected as soon as possible.

However, as will be shown in our paper, in the case of transient faults the approach may be different. A sequential circuit may pass through several incorrect states due to a fault, while maintaining correct outputs before the error is detected. Such a situation is suitable for transient faults since, before the error is detected, both the fault and its consequences may disappear, and the SSC may become fault-free again (self-healing property). Should we use any of the known approaches, we would often mark an SSC as erroneous prematurely, although it could successfully survive.

We will use the on-set realization of the SSC's output and next state functions [8]. Furthermore, we deal with implementations where both the next state and output equations are unate [2] in state variables and binate in primary input variables. SSCs that are implemented according to such a scheme we call state monotonic SSCs. In most cases, the mentioned realizations result in a considerable reduction of overhead [2, 7]. These realizations are also significant for the proposed approach being precisely those allowing the self-healing property [3].

Reduction of the fault latency is one of well-known challenges of self-checking design. The above-mentioned self-healing property is related to the period of time when the fault is present but has not been detected yet, i.e. with the latency. Indeed, if any fault is detected either immediately, or on the next clock pulse after its manifestation, the latency seems to be minimal, but the SSC is unable to recover. Reduction of the latency leads to declining of the self-healing ability.

In the present paper we propose techniques to overcome the above contradiction. We develop architecture for implementing self-checking SSCs, which is applicable to both types of the faults: permanent and transient. Moreover, it can be adapted to each type of the faults. This adaptation can be achieved within the framework of the same architecture by using different implementations of certain blocks of the architecture.

The proposed architecture is based on: a) the state monotonic implementation of self-checking SSCs [7]; b) the Match Detector architecture that does not require any coding of output vectors [4].

We investigate the architecture from the point of permanent fault latency and the self-healing ability for a transient fault. We assume that the resulting SSC will be implemented by FPGA.

This paper is organized as follows. Section 2 deals with state monotonic SSCs. Universal Match-Detector (UMD) architecture is proposed in Section 3. Implementing specific blocks of the UMD architecture are given in Section 4. Benchmarks results are given in Section 5. Conclusions are presented in Section 6.

2 State Monotonic SSC

2.1. Definitions

Let us describe a sequential machine according to the Mealy model.

Let **I**, **O**, **Q** - the sets of input, output and state vectors accordingly. N_I, N_O and N_Q - numbers of elements in these sets.

Let q_1 be the initial state.

δ - the next state function: $\delta : \mathbf{Q} \times \mathbf{I} \rightarrow \mathbf{Q}$,

λ - the output function: $\lambda : \mathbf{Q} \times \mathbf{I} \rightarrow \mathbf{O}$.

A schematic diagram of the synchronous sequential circuit is shown in Figure 1.

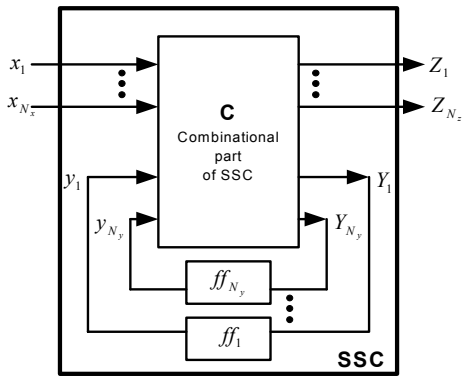

Figure 1. Synchronous Sequential Circuit

We will use the following notations for the SSC: C is a combinational part of the SSC,

and $ff_1,...,ff_{N_y}$ are D-flip-flops. Inputs of C are $x=\{x_1,...,x_{N_x}\}$ (input variables of the SSC) and $y=\{y_1,...,y_{N_y}\}$ (current state variables). Outputs of C are $Z=\{Z_1,...,Z_{N_z}\}$ (output variables of the SSC that realize the output function λ) and $Y=\{Y_1,...,Y_{N_y}\}$ (next state variables that realize the next state function δ). The combinational circuit C implements a system Φ of N_z+N_y Boolean functions $\{\delta,\lambda\}$ on N_x+N_y Boolean variables $\{x\cup y\}$.

Let us consider a specific realization of the SSC. We will call it a state-monotonic SSC.

Definition. A SSC having a combinational part defined as a system of Boolean functions that are partially monotonic in state variables is defined as a state monotonic SSC.

Any monotonic system of Boolean functions can be presented in the unate sum-of-products form [6].

Obviously, any system of Boolean functions partially monotonic in state variables can be presented in the sum-of-products form, which is unate in state variables.

Consider system Φ corresponding to the combinational part of SSC. Suppose that Φ is partially monotonic in state variables for a state monotonic SSC. Let Φ_f be a system corresponding to the combinational circuit **C** in presence of a fault *f*. Also suppose that Φ_f is partially monotonic in state variables. If $\Phi_f \prec \Phi$, we denote this fault as f_0. If $\Phi \prec \Phi_f$, we denote this fault as f_1. Construct a set of faults $\Im$ consisting of f_0 and f_1 faults and describe properties of these faults. Obviously, any unidirectional fault is either f_0 or f_1. We assume that any single stuck-at fault on the gate poles and state lines of the circuit is either f_0 or f_1 fault [7].

Theorem 1 [9]. *The state monotonic SSC is fault-secure for any permanent fault* f_0 *or* f_1.

Let us describe the behavior of the SSC in the case of a transient fault. As distinct from a permanent fault, the transient fault affects one or several clocks, and after that disappears. Notice, that during the period when the transient fault does appear, the fault's influence is similar to that of the permanent fault. The transient fault can result in:

1. A non-codeword will appear on the SSC's outputs at a certain clock after the disappearance of the fault.
2. Consequences of the fault will disappear and the SSC becomes fault free again.

The following Theorem 2 proves that an erroneous codeword cannot appear on outputs of the state monotonic SSC after disappearance of the transient fault.

Theorem 2 [9]. *The state monotonic SSC realization is fault-secure for any transient fault* f_0 *or* f_1.

2.2. Self-healing SSC

A state monotonic SSC may function in four different modes: fault free (**F**) mode; latent (**L**) mode; silent (**S**) mode and erroneous (**E**) mode. The SSC behavior is described in presence of either permanent or transient faults, using a graphical representation. We introduce a *Mode Transition Graph* (MTG) for this purpose.

An MTG describing the SSC behavior in a presence of a permanent fault is shown in Figure 2.

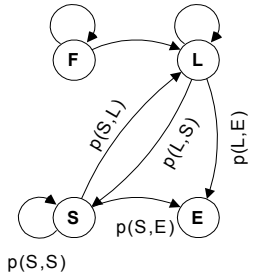

Fig. 2. Mode Transition Graph for a Permanent Fault

F - Fault free mode. The circuit remains in the fault free mode until a fault occurs.

L - Latent mode. It is a mode where the presence of a fault cannot be detected since a test vector detecting the fault has not yet appeared at the circuit's inputs. The circuit moves to the latent mode from the fault free mode when a fault occurs, and leaves the latent mode when a test vector is applied to the circuit.

S - Silent mode. It is a mode where a fault does not manifest itself in the form of a non-code output, although the presence of

the fault could potentially be detected if the next state lines (or the memory) can be observed. In this case only values of state variables $Y_1,...,Y_{N_y}$ are distorted. The circuit moves to the silent mode from the latent mode when a non-code state vector appears at the flip-flop inputs or outputs. From the silent mode the circuit is able either to move to an erroneous mode (**E**), or to revert to the latent mode.

E - Erroneous mode. It is a mode in which the circuit terminates its proper functioning, i.e. when a non-code output vector has been produced. The circuit is able to move to this mode either from the silent mode or from the latent mode.

In the case of transient fault, the dynamics of the above-mentioned modes are shown in Figure 3.

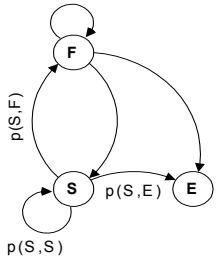

Fig. 3: Mode Transition Graph for a Transient Fault

Obviously, the latent mode, **L,** is absent in the case of a transient fault. When the transient fault occurs, the SSC moves from Fault mode, **F,** either to the Erroneous mode, **E**, (if a non-code output vector is produced) or to the silent mode, **S**, if a non-code next state vector is produced, while the output vector is a codeword. Note, that the sequential circuit is able to revert to the **F** mode after it's functioning in the **S** mode, which means that the SSC has become fault free again. Such a transition of the SSC from the **S** mode to the **F** mode we will call self-healing. We note that a SSC may have a self-healing property for a given fault and a given input sequence even if it does not have equivalent states.

Now, after we have described graphically behavior of the state monotonic SSC in presence of different faults we may formulate the main objective of the present paper as follows:

Developing a universal architecture of a self-checking sequential circuit, which would provide:

• maximizing a probability p(L, E) of the SSC transition from the latent mode to the erroneous mode for permanent faults;

• maximizing a probability p(S, F) of the SSC transition from the silent mode to the fault free mode for transient faults.

3 Universal Match Detector Architecture

The proposed universal self-checking architecture is based on the Match Detector (MD) architecture presented in [4]. The MD architecture does not require encoding of output vectors and consequently provides low overhead solutions. The checker of the MD architecture includes the Match Detector, and is respectively called an MD-checker.

The Universal Match Detector (UMD) architecture proposed in this paper is a generalization of the architecture described in [4]. As mentioned in the introduction, the main feature of the UMD architecture is its adaptability to either permanent or transient faults. A schematic diagram of the UMD architecture is shown in Figure 4.

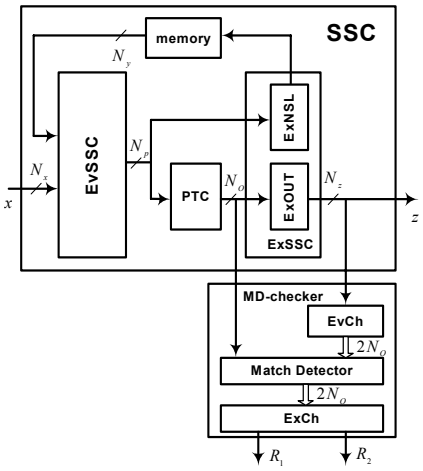

Fig. 4. The Schematic Diagram of UMD Architecture

Like the MD-architecture, the UMD architecture consists of two parts: a self-checking SSC and an MD-checker. In turn, each of these parts contains two main blocks: an “evolution” block and an “execution” block. In addition, the SSC contains a Product Terms Compressor (PTC). The function of the PTC is to form

1-out-of- N_O code from the output codeword.

Self-checking SSC. The evolution block of the SSC (EvSSC) implements all the product terms, while the execution block of the SSC (ExSSC) implements outputs functions and next state functions of the SSC. In turn, the ExSSC consists of two sub-blocks: an execution block for the next state logic (ExNSL) and an execution block for the output logic (ExOUT).

Inputs of the evolution block of the SSC (EvSSC) comprise primary inputs, $X = \{x_1,\ldots x_{N_x}\}$ of the SSC and output memory signals $Y = \{Y_1,...,Y_{N_y}\}$. Outputs of the EvSSC correspond to product terms **P**. The EvSSC is implemented as a tree, wherein each of the nodes is either a LUT or a fan-out. The memory signals are coded by codewords of the *1-out-of-* N_y code. The Product Terms Compressor (PTC) transforms the vector of product terms into *1-out-of-* N_O.

MD-checker. The MD-checker consists of the evolution block (EvCh), the execution block (ExCh) and the Math Detector (MD) situated between them. EvCh implements all minterms, while ExCh assembles these minterms to implement the checker's function.

The EvCh is built as a tree with "AND" nodes for implementation of product terms and "fork" nodes actually implemented by regular fan-outs. The ExCh comprises either *1-out-of-* N_O, or (N_O-1)*-out-of-* N_O LUTs combining all minterms, coming from the EvCh.

EvCh is implemented in the form of a self-checking two-rail tree. This tree is constructed in such a way that, in the case of proper functioning of both the SSC and the checker, one and only one dual-rail output (S_i,V_i) will have the value (1,0). All the remaining outputs will have the value (0,1). Outputs of the EvCh tree are inputs of the ExCh of the checker. Each of the two-rail outputs of the EvCh corresponds to a certain output vector of the original SSC.

All S-outputs of EvCh serve as inputs of the first component of the ExCh. This component is implemented as a converging *1-out-of-* N_O multilevel tree. All V-outputs of the EvCh are inputs to the second component of the ExCh, which is implemented as a converging (N_O-1)*-out-of-* N_O multilevel tree.

The Match Detector compares outputs of the PTC and outputs of the evolution block of the checker (EvCh). Any output vector of the PTC is formed by N_O binary one-rail outputs. Output vectors of EvCh are N_O dual-rail-coded outputs. In Figure 4, the checker is shown as MD-checker. If the two compared vectors are equal, the resulting vector will be equal to the EvCh output vector. If they are not, the ExCh will receive a predetermined faulty dual-rail vector.

An example of the match detection function is shown in Table 1.

Table 1. True table of the Match Detector

$\rho(i)$	$S^1(i)$, $V^1(i)$		$S^0(i)$, $V^0(i)$	
1	**1**	**0**	**1**	**0**
0	**0**	**1**	**0**	**1**
1	**0**	**1**	**1**	**1**
0	**1**	**0**	**0**	**0**
-	**0**	**0**	**0**	**0**
-	**1**	**1**	**1**	**1**

In this table: $S^1(i)$, $V^1(i)$ - dual-rail code of bit *i* of an output vector of the EvCh; $S^0(i)$, $V^0(i)$ - dual-rail code of bit *i* of the corresponding output vector of the match detector (MD); $\rho(i)$ - the state of bit *i* of the PTC single-rail output vector.

The main idea of the proposed approach is based on the property that an output vector of the PTC and the vector that is applied to ExCh are equal and both belong to the *1-out-of-* N_O encoding of the corresponding output codeword. These vectors have to be equal for the proper functioning of the SSC, and different in the case of a fault. Comparison of these two vectors by the Match Detector (MD) provides for the TSC property of the SSC.

4 Implementing PTC and ExSSC

In this section, we discuss the influence of different implementations of blocks PTC, ExOUT and ExNSL on to both the self-healing property and the latency of the UMD architecture. We show that change of the basis of implementation between the *1-out-of-n* function and the *OR* function

allows adapting the UMD architecture to permanent/transient faults.

All the blocks discussed (PTC, ExNSL and ExOUT) can be implemented by using either the *OR* or the *1-out-of-n* assembling elements. In both cases, the Self-checking Property of the UMD architecture is satisfied. As they are functionally equivalent in the fault free mode, these solutions behave differently if a fault occurs. The *1-out-of-n* based solution generally provides a lower fault latency than the *OR* based solution. In turn, the *OR* based implementation provides the self-healing property for transient faults.

If a fault affects an output vector of the ExOUT, the fault will be detected immediately and self-healing cannot be achieved. Thus, the ExOUT block can always be implemented by using *1-out-of-n* functions, regardless the orientation of the UMD architecture (to permanent faults and/or transient faults).

The self-healing may happen in the case when a fault occurs and does not affect the SSC's outputs, but appears on the SSC next state lines. To provide the self-healing, in the case of transient fault orientation (maximization of the self-healing ability), we propose to implement both the PTC and the ExNSL by *OR* elements. In turn, in the case of permanent fault orientation (minimizing the fault latency) we propose to implement both of the blocks by *1-out-of-n* elements.

Hence, the UMD architecture can be adapted to a particular type of fault by choosing the basis of implementation of two its blocks: PTC and ExNSL. The *1-out-of-n* based implementation of these blocks leads to an architecture that has the minimal permanent fault latency and lacks the self-healing ability. On the other hand, the *OR* based implementation leads to the high permanent fault latency and the self-healing ability with respect of faults.

One implementation of an exemplary SSC is illustrated by Figure 5. Elements of ExNSL and PTC blocks are either the *OR* functions ($\vee$) or the *1-out-of-n* functions ($\oplus$).

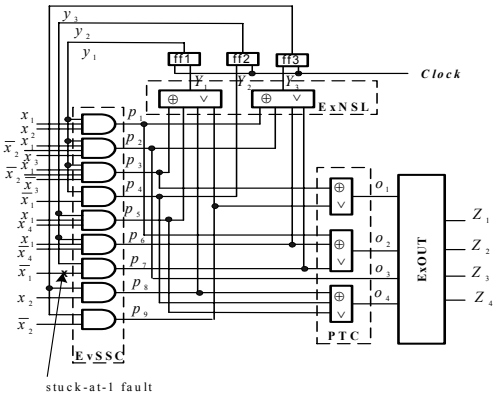

Fig. 5. Exemplary SSC implemented according to UMD architecture

We received values of the latency and the self-healing time for a certain fault of the SSC shown in Figure 5 experimentally (experiments are described below in Section 5). Let a *stuck-at*-1 fault occurs on the input $\bar{x}_1$ of the element realizing the product term p_7. In the case when the fault is permanent, the average latency for the *OR*-based implementation is equal to 16.77; for the *1-out-of-n* – based implementation the latency is equal to 7.17. The second solution provides a substantial reduction in the latency (more then 50%). If the fault is transient, then the *OR*-based implementation provides the self-healing property. In our example the probability of self-healing is 16.57%.

5 Experimental Results

To arrive at values of fault latencies and self-healing for both of the proposed implementations of the UMD architecture, we performed experiments with benchmarks circuits [5] in which both permanent and transient fault were injected. Behavior of the faulty and the fault-free circuits was simulated on random input sequences. We assumed that the input vectors of the SSC are equally probable. Random single *stuck-at*-1 faults were injected into an arbitrary input or output bit position of blocks (EvSSC, ExNSL, PTC and ExOUT). The time selected for starting the injecting of faults was the steady-state time. We assumed that the duration of a transient fault is equal to one clock cycle.

Experimental results are presented in Table 2.

Table 2. Benchmarks results for the proposed architecture

Example	Permanent Faults			Transient Faults		
	OR-based	1-out-of-n based	latency reduction (%)	*OR*-based		1-out-of-n based
	T_1	T_2	Ω	T_{heal}	Ψ	T_{det}
ex1	6589	4876	25.998	1.331	10.608	0.194
ex6	2124	1577	25.753	1.366	12.931	0.226
s386	11243	4529	59.717	1.42	14.14	0.226
s820	5324	4324	18.783	1.215	5.16	0.221
sse	12340	9807	20.527	1.41	16.94	0.226
beecount	4253	581	86.339	1.614	48.72	0.23
bbtas	698	323	53.725	3.151	24.96	0.235
bbara	9798	5854	40.253	5.284	42.68	0.248
dk15	1720	994	42.209	1.466	14.32	0.22
Average	6009.	3651	41.48	2.03	21.16	0.23

Column T_1 corresponds to the fault latency for a permanent fault within the *OR*-based implementation, and column T_2 - for the *1-out-of-n* based implementation.

$\Omega = 100 * (T_1 - T_2)/T_1$ is the percentage of the latency reduction for the *1-out-of-n*-based implementation in comparison with the latency for the *OR*-based implementation.

T_{heal} is the average time of the self-healing after a transient fault within the *OR*-based implementation (time of staying in the silent mode S before returning to the fault free mode F);

T_{det} is the average time of a transient fault detection for the *1-out-of-n*-based implementation (time of staying in the silent mode S before moving to the erroneous mode E).

ψ is a percentage of input sequences that lead to the self-healing in the case of the *OR*-based implementation.

Experimental results show that:

1. The *1-out-of-n*-based implementation provides reduction of permanent faults latencies of about 42% as compared with the OR-based implementation. This reduction can be explained by the fact that the permanent faults latency for *1-out-of-n*-based implementation comprises a single presence of SSC in the latent mode (L), while in the case of using the OR-based implementation SSC is able to be in the latent mode repeatedly. In the case of the *1-out-of-n*-based based implementation SSC arrives to the latent mode only once and never returns there again after leaving the latent mode.

2. Using the *OR*-based implementation of the UMD architecture provides a capability of self-healing in the case of a transient fault. On random input sequences introduced to benchmarks, the average probability of self-healing is about 21%. The "future" of a circuit is determined during a small number of clocks. Several clocks are sufficient for SSC both for the healing and for marking it as faulty. The time of remaining in the silent mode S is very small.

6 Conclusions

In this paper, we present architecture of self-checking Synchronous Sequential Circuits (SSCs), based on Universal Match Detector (UMD). We noticed a phenomenon of self-healing of such circuits. Circuits with the self-healing ability are partially monotonic in their state variables. On one hand, the state monotony enables the circuit to be self-healing with respect to some transient faults. On the other hand, using state monotonic SSCs leads to a considerable latency increase for permanent faults.

The proposed UMD architecture can be adapted either to permanent or to transient faults. In other words, the same solution can be applied to both types of faults. Two alternative implementations are proposed, relating to functions of specific blocks in the proposed architecture:

a) *1-out-of-n* based implementation for cases of predominance of permanent faults;

b) *OR* based implementation for cases of predominance of transient faults.

The above two implementations of the same architecture are equal from the point of overhead. Moreover, these implementations can be converted one to the other by reconfiguring blocks in the UMD architecture.

In practice, the UMD architecture can be efficiently applied for situations when predomination of the specific type of a fault (permanent or transient) is known in advance. In many cases such information is available or can be achieved by analyzing a certain application of the SSC.

It is pertinent to note that the two implementations provide the TSC property and detect both permanent and transient faults. The proposed adjustment of the UMD architecture allows improving characteristics of the SSC with respect to a particular type of faults.

Since adaptation of the UMD architecture for a certain type of faults can be performed just by reconfiguring its blocks, in future it could be utilized in novel self-checking solutions, if such would distinct between permanent and transient faults in real time.

References:

[1]. Jha, N.K., and S.J. Wang, Design and synthesis of self-checking VLSI circuits and systems, *IEEE Trans. CAD*, 12, no. 6, June 1993, pp. 878-887.

[2]. Lala, P., Self-checking and Fault-Tolerant Digital Design, Morgan Kaufmann Publishers, San-Francisco / San-Diego / New-York/ Boston/ London/ Sydney/ Tokyo, 2000.

[3]. Levin I., Matrosova A., Ostanin S., Survivable Self-checking Sequential Circuits. *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*. DFT2001. 24-26 October 2001, San Francisco, California, USA, pp. 395-402.

[4]. Levin, I., V. Sinelnikov, Self-checking of FPGA based Control Units, *Proceedings of 9th Great Lakes Symposium on VLSI*, Ann Arbor, Michigan, 1999, IEEE press, pp. 292-295.

[5]. Lisanke, R., Logic Synthesis Benchmark Circuits, *International Workshop on Logic Synthesis,* Research Triangle Park, NC, May 1989.

[6]. Mago, G., Monotone Functions in Sequential Circuits, *IEEE Transactions on Computers,* vol. C-22. October 1973, pp. 928-933.

[7]. Matrosova A., S. Ostanin., Self-checking FSM Design with Observing only FSM Outputs, *Proc. The 6-th Int. On-Line Testing Workshop*, July 2000, pp.153-154.

[8]. Tohma, Y., Y. Ohyama, R. Zakai, Realization of fail-safe sequential machines by using a k-out-of-n code, *IEEE Trans. Computers,* Vol. c-20, N11, November 1971.

[9]. Levin I., Sinelnikov V., Karpovsky M., Ostanin S. Sequential Circuits Applicable for Detecting Different Types of Faults, *Proceedings of the 8th IEEE International On-line Testing Workshop,* July 2002, Isle of Bendor, France, pp. 44-48.