Self-checking Sequential Circuits with Self-healing Ability[1]

Ilya Levin[*] Vladimir Ostrovsky[*] Sergey Ostanin[*] Mark Karpovsky[**]

[*]Tel-Aviv University, Ramat Aviv, 69978, Israel

[**]Boston University, 8 Saint Mary's Street, Boston, MA 02215, USA

972-3-6407109 972-3-6407799 972-3-6407799 617-353-9592

ilia1@post.tau.ac.il vios@post.tau.ac.il sostanin@post.tau.ac.il markkar@bu.edu

ABSTRACT

In this paper we deal with totally self-checking (TSC) synchronous sequential circuits (SSCs), that are able to recover after an occurrence of a fault. We call SSC owing this property as a self-healing SSC. A concept of a partially monotonic SSC is used in the paper. It is shown that the partially monotonic SSCs satisfy the self-healing property. A novel *reduced m-out-of-n* code is developed. It is proposed applying this code to the synthesis of a TSC checker for the state monotonic SSCs. The proposed method of synthesis is based on a LUT implementation of monotonic functions.

For most circuits in a standard benchmark set, the proposed approach leads to a reduction of about 10-20% of the overhead as compared with the traditional methods.

Keywords

Self-checking, synchronous sequential circuit, self-healing, unordered coding, checker.

1. INTRODUCTION

Two different ideologies can be considered for handling an error detected in a circuit concurrently with its functioning. The first ideology is based on the immediate marking of the circuit as erroneous when an error is detected. An alternative ideology concentrates on increasing the survivability of the circuit, which means, "to give a chance" to the circuit to continue working during several 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. In the present paper we study precisely this case.

We propose a method for a synthesis of synchronous sequential circuits (SSCs) that have an ability to continue their proper functioning in the presence of affects of the fault and investigate an ability of such circuits to recover. Naturally, this self-healing ability can be archived by increasing a fault latency of the corresponding SSC. Consequently, a synthesis of self-checking checkers for such circuits becomes especially interesting.

The synthesis of self-healing SSCs and corresponding self-checking checkers are in the focus of the present paper.

GLSVLSI'02, April 18-19, 2002, New York, New York, USA.

2. RELATED WORKS AND RESEARCH AGENDA

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

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

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

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

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

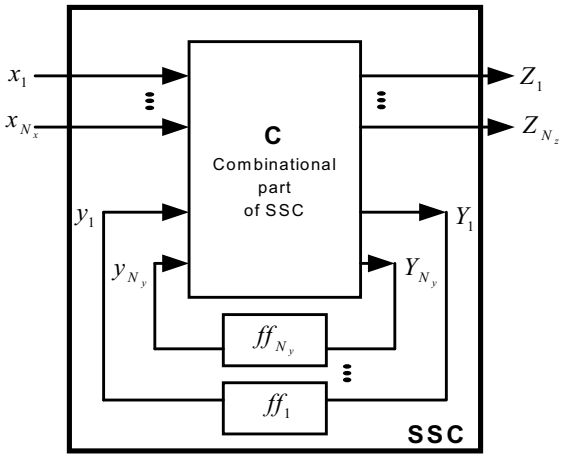


Figure 1. Synchronous Sequential Circuit

Let us introduce the following notations.

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

Let **C** be a combinational part of the SSC, and $ff_1,...,ff_{N_y}$ be *D*-flip-flops. Inputs of **C** be $x=\{x_1,...,x_{N_x}\}$ (inputs variables of the SSC) and $y=\{y_1,...,y_{N_y}\}$ (current state variables). Outputs of **C** be $Z=\{Z_1,...,Z_{N_z}\}$ (output variables of the SSC implementing 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 $\{\delta,\lambda\}$ of N_x+N_y Boolean variables $\{x\cup y\}$.

We consider all unidirectional errors [2] and assume that only output lines of the SSC are observable. It leads to the following definition of the Totally Self-Checking (TSC) sequential circuit [2].

Definition 1. A sequential circuit is self-testing if, for every fault in a fault set, there is an input/state code pair in the circuit such that a non-code output is produced.

Definition 2. A sequential circuit is fault secure if, for every fault from the faulty set the sequential circuit never produces an incorrect code output for a code input.

Definition 3. A sequential circuit is totally self-checking if it is both self-testing and fault-secure.

Two main approaches can be identified in the design of self-checking SSCs. The first one is based on applying special techniques to observing state transitions or a control flow. These techniques range from state coding by error-detection codes to control flow monitoring by signature analysis [3, 8] or special monitoring machines [9]. Usually, these techniques lead to considerable overhead.

The second approach is based on checking the SSC's outputs without direct checking of the memory. When these techniques are applied, a fault that leads to a non-code state vector is detected in the next clock cycle [2]. This property can be achieved by introducing an additional overhead. The present paper investigates the behavior of the SSC without memory checking. Such SSCs are free of additional overhead that would be required by the condition of immediate fault detection. We consider that the absence of immediate fault detection to be a significant advantage of the scheme, and utilize it for achieving a self-healing ability.

We use the on-set realization of output and next state functions [13]. 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 self-checking state monotonic SSCs. In most cases using such implementations leads to a considerable reduction in overhead. Moreover, if only output lines are handled in such schemes, SSCs can function properly with the presence of a fault and even recover from the fault [4]. Such SSCs we call self-healing SSCs.

Since the self-healing SSCs belong to self-checking circuits, designing such circuits is based of introducing a redundant circuitry that provides the TSC property. Development and investigation of methods for synthesis of TSC self-healing SSCs is the second objective of the present paper.

Major difficulties in designing self-checking devices are related to the complexity of coding (forming the check bits) and decoding (i.e. verification that a given output is a codeword). We deal with a SSC based controller where the number of possible output vectors is much smaller than 2^{N_z} (where N_z - is the number of output lines), while the set of possible vectors is known in advance. This property was used by [5] for designing checkers. The authors show that the checker for an SSC controller can be efficiently implemented in the form of "sum-of-minterms" (SOM) for output functions of the controller. An unordered code for output vectors is used since, for these codes, no unidirectional error can transform one codeword into another codeword. Note that the SOM-checker examines whether an output vector belongs to the set of possible codewords, and not to the Berger code (as in the case of standard design [2, 6]). An efficient unordered code was developed by Smith [12] specifically for the case when the number of possible codewords is essentially smaller then 2^{N_z}.

The main objective in the construction of unordered codes (Berger, Smith codes) is to achieve the required properties by using a minimal number of redundant bits. Nevertheless, decrease of the number of bits in many cases does not lead to an expected reduction in the checker's complexity. Thus, an approach that takes into account of the checker's complexity at the earlier stages of the code construction becomes quite topical. Moreover, the basis of implementation has to be taken into consideration. Note, that in our case we are interested in the checker's implementations by LUT based FPGA. Based on these requirements we propose a novel so-called *reduced m-out-of-n* code. The *reduced m-out-of-n* code differs from the standard code by having the two following features:

1) this code is a systematic one;

2) this code enables dividing the codeword into m fields in such a way that any acceptable codeword has exactly one bit that is equal to one in each of the fields.

The proposed reduced code allows checking each of the fields separately and therefore simplifies the checker.

Summarizing above-mentioned consideration we can formulate the aim of the paper as development of methods for synthesis of self-healing SSCs and self-checking checkers for such circuits. The paper is organized correspondingly. In Section 3, we describe so-called state monotonic SSCs and their self-healing property. The design of m-out-of-n checkers for the self-healing SSCs is presented in Section 4. Benchmarks results are given in Section 5. Conclusions and references are presented in Section 6 and 7, respectively.

3. STATE MONOTONIC SSC

3.1 Definitions

Let us consider an arbitrary system Φ of Boolean functions ϕ_i:

$$\Phi(x_1,...,x_k)=\{\phi_1(x_1,...,x_k),\phi_2(x_1,...,x_k),...,\phi_I(x_1,...,x_k)\}.$$

Let $A=(a_1,...,a_k)$ and $B=(b_1,...,b_k)$ be Boolean *k*- tuples.

Definition 4. The system Φ is monotonic if, for any pair of Boolean *k*-tuples *A*, *B* so that $A \le B$ (i.e. $a_i \le b_i$ for any $i = \overline{(1,k)}$), the condition $\Phi(A) \le \Phi(B)$ is satisfied.

Consider a subset x^* of Boolean variables, $x^* \subseteq \{x_1,...,x_k\}$.

Definition 5. Vector *B* covers *A* in variables from the set x^* ($A \overset{x^*}{\le} B$), for any component *i*, corresponding to a variable from x^*, $a_i \le b_i$, and for any other component *j* corresponding to a variable $\{\{x_1,...,x_k\} \setminus x^*\}$, $a_j = b_j$.

Definition 6. The system Φ is partially monotonic in x^* variables if for any pair of Boolean *k*-tuples *A*, *B* the condition $\Phi(A) \le \Phi(B)$ is satisfied for $A \overset{x^*}{\le} B$.

Notice that if Φ is monotonic it is partially monotonic in any subset of its variables.

Definition 7. Let Φ', Φ'' be two different systems having the same number of functions and the same variables if for any *k*-tuple $A \in \{0,1\}^k$: $\Phi'(A) \le \Phi''(A)$, then Φ'' implicates Φ' ($\Phi' \prec \Phi''$).

3.2 TSC property for the state monotonic SSC

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

Definition 8. SSC called a partially monotonic if the system of Boolean functions corresponding to its combinational part is partially monotonic in state variables.

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

Let combinational circuit **C** (the combinational part of an SSC) corresponds to the system Φ. System 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 $\Phi_f \prec \Phi$ (Φ implicates Φ_f), we will denote this fault as f_0. If $\Phi \prec \Phi_f$ (Φ_f implicates Φ), we will 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. Considering various synthesis methods and various sets of faults we can clarify whether the corresponding set contains only f_0 and f_1 faults or not. It has been proven in [7] that if the SSC is synthesized as the partially monotonic system by using either two-level or multi-level methods, then any single stuck-at fault on the gate poles and state lines of the circuit are f_0 and f_1 faults.

Theorem 1. The state monotonic SSC is totally self-checking for any fault from $\Im$.

The proof of Theorem 1 is based on properties of partially monotonic systems and the fact that faults occur one at a time while between any two faults a sufficient time elapses [11].

Obviously, any fault from $\Im$ is either undetectable or unidirectional by any input sequences $\tilde{I}$ If a fault remains undetectable on the SSC output lines during the time the sequence $\tilde{I}$ manifests itself on the SSC state lines, we will call the fault as a masked fault. The corresponding SSC property we call a fault masking property.

Note that only f_1 fault can be masked for the state monotonic SSC. Any f_0 fault will be manifested at least on the next clock. Precisely f_1 faults will be in the focus of our research.

3.3 Self-healing SSC

To formulate our approach, let us define four modes of functioning of the SSC and describe its behavior in presence of either permanent or transient faults, using a graphical representation. We introduce a Mode Transition Graph (MTG) for this purpose. Vertexes of the MTG correspond to specific modes of the circuit. Transitions between the modes correspond to specific events.

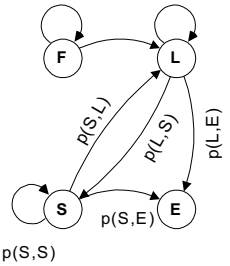


Figure 2. Mode Transition Graph for a Permanent Fault

An MTG describing the proposed concept with respect to a permanent fault is shown in Figure 2. Four different modes of functioning of the state monotonic SSC can be considered.

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 while the SSC is functioning since the test vector detecting the fault has not yet appeared on 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 the test vector is applied to the circuit.

S - Silent mode. It is a mode where the presence of a fault does not manifest itself in the form of a non-code output, although the presence of the fault could potentially be detected when the next state lines (or the memory) can be observed. The circuit moves to the silent mode from the latent mode when a non-code state vector appears on 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, transitions between above-mentioned modes are shown in Figure 3. Obviously, the latent mode **L** is absent in the case of a transient fault. When the transient fault occurs, the SSC moves from Fault free 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 return 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.

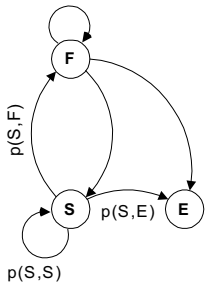


Figure 3. Mode Transition Graph for a Transient Fault

Hence, in this section we have introduced the self-healing property of SSCs and have illustrated this property by the example of the state monotonic SSCs. Furthermore, we have shown that the considered state monotonic SSCs satisfy the TSC property. In the next section we deal with methods for synthesis of self-checking checkers for such SSCs.

4. DESIGNING CHECKERS FOR THE STATE MONOTONIC SSC

The present section introduces a *reduced m-out-of-n* code and a method for synthesis of the corresponding checker. Being implemented by LUTs, this checker has a simple structure with relatively low overhead.

Define a relation of compatibility (incompatibility) on the set of acceptable output codewords **O**, as follows:

Definition 9. Two variables Z_i and Z_j are compatible if they are both equal to "one" in at least one codeword $o_r \in \mathbf{O}$. In other cases these variables are incompatible.

Definition 10. Subset $T_j \subset Z$ ($Z = \{Z_1,...,Z_{N_Z}\}$) is a set of compatible (incompatible) output variables if any two variables that belong to the set are compatible (incompatible).

Obviously, if the set T_j - is the set of incompatible output variables, then every acceptable output codeword may contain a maximum of one variable from the set (or does not contain any variable from the set at all).

Let set Z is divided into subsets $T_1,...,T_m$ of incompatible variables $Z = T_1 \cup ... \cup T_m$, $T_i \cap T_j = \varnothing$, for any $i \neq j$.

There are several well-known algorithms for optimal partitioning on the set Z [10]. We do not deal with this problem. We just mention here that it is reasonable to find the partition with the minimal number of the subsets.

For constructing the *reduced m-out-of-n* code we put each subset T_i into the one-to-one correspondence to a specific check bit c_i; c_i=1 only in those output codewords that have no output variables from T_i.

Obviously, every output word from the proposed code contains exactly *m* bits that are equal to one, according to the number of subsets, T_i. An example of such a coding is presented in Table 1.

Table 1. An Example of Reduced Coding

Output code words	Information bits					Check bits		
	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
	Z_1	Z_2	Z_3	Z_4	Z_5	c_1	c_2	c_3
o_1	0	0	0	0	0	1	1	1
o_2	1	1	0	0	1	0	0	0
o_3	0	1	0	1	0	0	0	1
o_4	0	0	1	0	1	0	1	0
o_5	0	0	1	0	0	0	1	1
o_6	1	1	0	0	0	0	0	1
o_7	0	1	0	0	0	1	0	1

The following partition has been constructed according to the above matrix: $\{Z_1,...,Z_5\} = \{\overline{Z_1,Z_3,Z_4}; \overline{Z_2}; \overline{Z_5}\}$. The proposed coding is based on the above partition.

Check bits are represented by three right hand columns in Table 1. Let us introduce continuous numbering for information bits. Denote a binary bit of the codeword by symbol b_i (see Table 1). Partition $B = \{B_1,B_2,B_3\} = \{\overline{b_1,b_3,b_4,b_6}; \overline{b_2,b_7}; \overline{b_5,b_8}\}$ corresponds to the *reduced 3-out-of-8* code.

We will call this partition a coding partition. Note, that each codeword includes exactly one variable from every subset. $B_i \subset B$, $i = 1,..., m$.

We propose to use symmetric functions for the checker's functional description. The symmetric functions are a convenient and acceptable form of representation for logic functions [1]. We will use the symbol $F_h(X)$ for such the notation of such functions, where h is a condition defining the number of arguments that are equal to "one" when the function is equal to "one". For example, $F_{=1}(X)$ is a so-called "1-hot" function.

Theorem 2. Let $B = \{B_1,...,B_m\}$ be a coding partition. The compatibility relation is defined as the set **O** of acceptable codewords. Hence:

$$R = F_{=1}(B_1) \& ... \& F_{=1}(B_m) = = \begin{cases} 1, & \text{on every } o_i \in O, \\ 0, & \text{on every unidirectional error.} \end{cases}$$

Based on Theorem 2, function R may be used for detecting unidirectional errors. Unfortunately, this function cannot be used

for the straightforward implementation by a LUT-based FPGA, since function $F_{=1}(B_i)$ does not satisfy the self-testing property.

Assume that the checker is implemented by the LUT-based FPGA, and the LUT has *s* inputs. In our examples assume that: *s*=4. We will estimate the scheme complexity by the number of LUTs.

According to the accepted practice in the design of checkers we represent an error signal by using two signals R_1 and R_2 (R_1, R_2 - implementation). If $R_1 \neq R_2$, the scheme is fault free, in the opposite case a fault has occurred either in the controller or in the checker.

Function *R* can be implemented by using the 1-out-of-n function and the product function. Function $F_{=1}(X)$ can be presented in the following form: $R_1 = F_{\geq 1}(X_1) \vee F_{\geq 2}(X_2)$ and $R_2 = F_{\geq 2}(X_1) \vee F_{\geq 1}(X_2)$, where $X_1 \cup X_2 = X$ and $X_1 \cap X_2 = \varnothing$.

If the number of arguments of $F_{=1}(X)$ equal *k* and the number of LUT's inputs *s* is greater or equal than *k* ($s \geq k$), then function 1-out-of-k can be implemented by two LUTs. Note that the schemes are testable on the set of acceptable codewords and that they do not include inverters. If $s < k$, the following simple decomposition can be used:

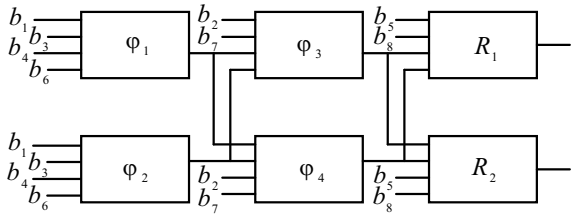
$F_{=1}(X) = F_{=1}(F_{=1}(X_1), X_2)$, where $X_1 \cup X_2 = X$ and $X_1 \cap X_2 = \varnothing$.

LUTs having an even number of inputs are appropriate for implementation of dual-rail products. Let it be required to implement the following product: $\gamma = \alpha \,\&\, \beta$, where each of the variables is represented in the dual-rail form: (γ_1, γ_2), (α_1, α_2), and (β_1, β_2) respectively. Two 4-input LUTs will be required for implementation of the function γ. These LUTs will be programmed for implementation of the following functions:

$\gamma_1 = \alpha_1 \,\&\, \beta_1 \vee \alpha_2 \,\&\, \beta_2$ and $\gamma_2 = \alpha_1 \,\&\, \beta_2 \vee \alpha_2 \,\&\, \beta_1$.

m-product is implemented by using 2(*m*-1) 4-input LUTs having a pyramidal structure.

The checker scheme that uses the proposed decomposition is shown in Figure 4.



$\varphi_1 = b_1 \vee b_3 \vee b_4 b_6$, $\varphi_3 = \varphi_1 b_2 \vee \varphi_2 b_7$, $R_1 = \varphi_3 b_5 \vee \varphi_4 b_8$,

$\varphi_2 = b_4 \vee b_6 \vee b_1 b_3$, $\varphi_4 = \varphi_1 b_7 \vee \varphi_2 b_2$, $R_2 = \varphi_3 b_8 \vee \varphi_4 b_5$.

Figure 4. An Example of the LUT-based on *the m-out-of-n* Checker

The checker detects errors on the set of acceptable codewords listed in Table 1. The 4-input LUT used in the scheme is shown in the form of boxes where the corresponding functions are indicated.

Now we estimate the checker's complexity by the number *L* of LUTs that have been used for implementing the checker. Let N_Z output signals be partitioned into m subsets for coding. Each of these subsets includes N_{T_i} elements. In this case:

$$L = 2\left(\sum_{i=1}^{m} \lceil (N_{T_i} - 1)/2 \rceil + m - 1\right).$$

Finally we have the following boundary conditions:

$$(N_Z + m - 2) \leq L \leq (N_Z + 2m - 2).$$

5. EXPERIMENTAL RESULTS

We applied the synthesis approach described above to several MCNC benchmarks to compute implementations for Altera series FPGAs. Comparisons of overheads proposed architecture with

Table 2. Benchmark's results

Circuit	N_X	N_Z	N_Q	*P*	Ω_1	Ω_2	Ω_3	Sequences, %
Cse	7	7	16	91	49.4	49.4	48.1	65
Ex1	9	19	20	138	80.1	57.3	14.6	18
Ex6	5	8	8	34	59.3	44.4	29.6	25
Pma	8	8	24	73	55.4	50	41.1	2
S386	7	7	13	64	56.9	50	41.4	25
S820	18	19	25	232	23.6	14	9.5	3
S832	18	19	25	245	33.3	19.2	13.4	4
Sse	7	7	13	56	50	51.9	51.9	31
Avrg.	9.88	11.75	18	116.63	51	42.03	31.2	21.63

various self-checking architectures are presented in Table 2. Columns "circuit", N_Q, N_x, N_z and P denote FSM name, the number of states, the number of primary inputs, the number of primary outputs and number of products of original FSMs, respectively. Columns Ω_1, Ω_2 and Ω_3 denote the overheads (%) for the Berger output encoding, the Smith output encoding and the proposed *reduced m-out-of-n* encoding, respectively.

We performed a statistical experiment of investigating the self-healing ability for the benchmarks' circuits. Behavior of the SSC in the presence of a transient fault was simulated on random input sequences of length 500. We assumed the inputs values are equally probable. A random fault occurs in the SSC's steady-state. The percentage of sequences on which a circuit survived among random 1000 sequences is shown in last column of the Table 2.

The table demonstrate that the newly proposed technique for designing totally self-checking SSCs based on the *reduced m-out-of-n* coding results in a hardware overhead of about 30%, while the Berger coding implementations require an overhead of about 50% and the Smith coding – about 40%. In other words, the authors' approach leads to a reduction of about 10-20% of the overhead as compared with the traditional methods.

The benchmark results show the average percentage of sequences, that lead the SSC to self-healing, to be about 22%.

6. CONCLUSION

In this paper, we investigate implementations of self-checking sequential synchronous circuits (SSCs) operating without checking their memory. A phenomenon of the self-healing of such circuits has been described. Circuits with the self-healing ability are partial monotonic in their state variables. Such circuits has been called state monotonic SSCs. On the one hand the state monotony enables the circuit the self-healing with respect to a transient fault, on the other hand, the state monotony opens the way for the SSC simplifying.

A novel *reduced m-out-of-n* code has been developed. The authors propose applying this code to the synthesis of a TSC checker for the state monotonic SSCs. The proposed method of synthesis is based on a LUT implementation of monotonic functions.

The self-checking approaches have been assisted on several SSC benchmarks. In most cases for the *reduced m-out-of-n* coding based architecture, overheads for the benchmarks are considerably lower in comparison with traditional approaches. Consequently, the proposed self-checking architecture of self-healing SSCs can be recommended for implementing circuits in a case of strong overhead requirements.

7. REFERENCES

[1] Akers, S. B., "A rectangular logic arrays," IEEE Trans. on Computers, Vol. C-21, August 1972, pp. 848-857.

[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] Leveugle, R., and G. Saucier, "Optimized Synthesis of Concurrently Checked Controllers", IEEE Transactions on Computers, Vol. 39, No. 4, April 1990, pp. 419-425.

[4] Levin, I., A. Matrosova, S. Ostanin, "Survivable Self-checking Sequential Circuits", Proc. of the Defect and Fault Tolerance in VLSI Systems Symposium, October 24-26, 2001, San Francisco, CA, USA, pp. 395-402.

[5] Levin, I., M. Karpovsky, "On-line Self-Checking of Microprogram Control Units", The 4th International On-line Testing Workshop, Capri, 1998, pp. 153-159.

[6] 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.

[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] Noubir, G., B. Y. Choueiry, "Algebraic Techniques for the Optimization of Control Flow Checking", Proc. of IEEE FTCS'96, 1996, pp. 128-137.

[9] Parekhji, R.A., G. Venkatesh, and S.D. Sherlekar, "Concurrent Error Detection Using Monitoring Machines", IEEE Design & Test of Computers, Vol. 12, No. 3, 1995, pp. 24-31.

[10] Paull, M.C., and S.H. Unger, "Minimizing the number of states in incompletely specified sequential switching functions", IRE Trans. Electron. Computers, V EC-8, No. 3, 1959, pp. 356-367.

[11] Smith, J.E., and G. Metze, "Strongly Fault Secure Logic Networks", IEEE Transaction on Computers, vol. C-27, June 1978, pp. 491-499.

[12] Smith, J., "On Separable Unordered Codes", IEEE Transaction on Computers, Vol. C-33, No. 8, August 1984, pp. 741-743.

[13] 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.