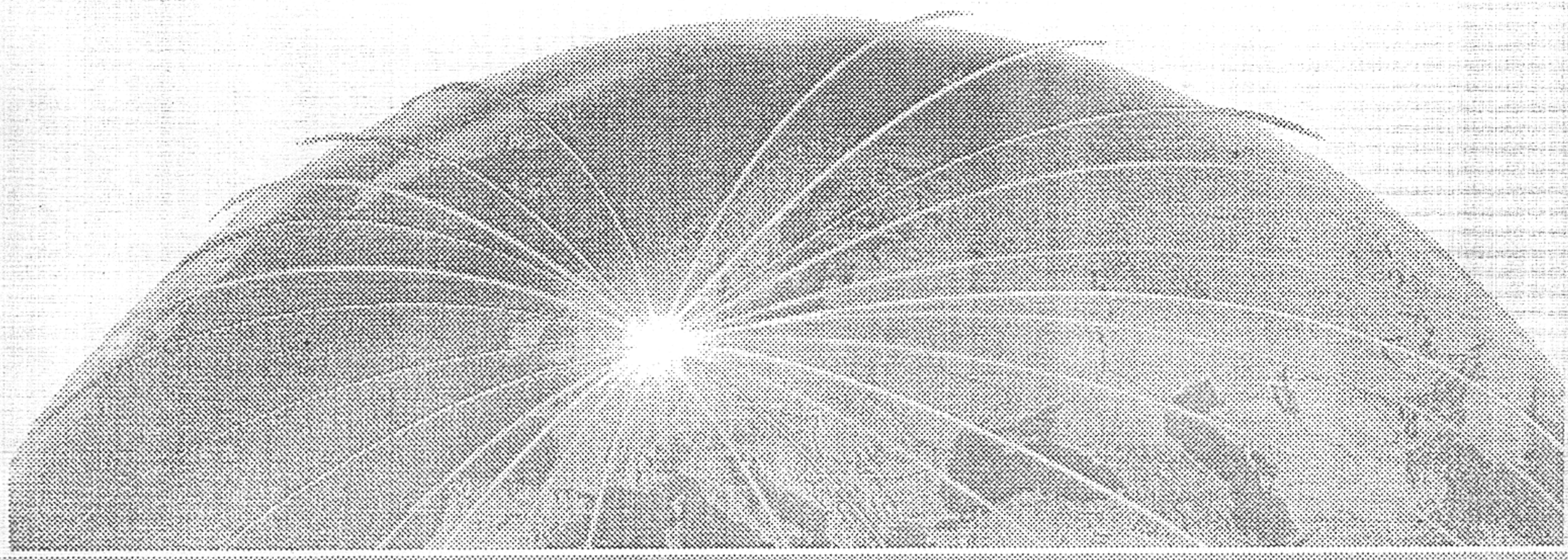
Proceedings

of the Fourth International Conference on

New Information Technologies

NITe'2000

Volume 1



Minsk, Belarus, December 5-7, 2000

NEW ARCHITECTURE FOR SEQUENTIAL MACHINES WITH SELF-ERROR DETECTION*

Mark Karpovsky[1], Ilya Levin[2], Vladimir Sinelnikov[3] and Roman Goot[3]

[1] --Boston University, Department of Electrical, Computer and System Engineering, Boston, MA 02215 USA
[2] –Tel-Aviv University, School of Education, Ramat-Aviv 69978, Israel
[3] –Academic Technological Institute, Department of Computer Science, Golomb, 52, Holon, Israel, e-mail: sinel @barley.cteh.ac.il

ABSTRACT

The present paper investigates a new technique for on-line checking of FPGA-based sequential devices defined by their finite state machines (FSMs). This technique is based on architecture comprising two portions: a self-checking FSM and a separate totally self-checking (TSC) checker. Each of these portions is implemented as a combination of an Evolution block and an Execution block. Comparison of code vectors transferred between these blocks provides for the totally self-checking property. The proposed technique does not require any encoding of output words and uses a one-rail design, thereby drastically decreasing the required overhead. The paper presents overhead estimations and results for benchmarks for the proposed architecture. The Markovian analysis is used for investigation of fault latencies.

1. INTRODUCTION

Techniques for concurrent error detection in finite state machine (FSM) controllers have received a wide attention since control part of a digital system is usually the most critical part from the testability point of view. Irregularity and complexity of the control structure on one hand, and its central role in functioning of the whole digital system to be controlled on the other hand, puts the problem of synthesis of self-checking FSM controllers onto the theoretical and practical agenda. Most of the faults that occur in VLSI circuits and systems are transient/intermittent in nature. The self-checking property allows both the transient/intermittent and permanent faults to be detected, thus preventing data contamination.

Existing general approaches to designing of self-checking FSMs are based on either duplication thereof or application of a specific error detecting codes (Berger code, constant weight code, etc.). In most cases these approaches require a hardware overhead of more than 100 percent.

According to [1], in the synthesis of totally self-checking (TSC) controllers, some basic properties of FSMs being a formal model of the controllers can be utilized for minimizing of overheads. Particularly, a totally self-checking (TSC) checker for a FSM can be implemented as a Sum-of-Minterms (SOM) of Boolean function $z(y_1,\ldots,y_N)$ of the FSM output variables $y_1,\ldots,y_N$. Function z is equal 1 if the FSM output word is a codeword (microinstruction), and equal 0 if the FSM output word is not a codeword. Each of minterms of $z(y_1,\ldots, y_N)$ corresponds to a specific codeword of the FSM. Since the number of microinstructions for most microcontrollers is not very large, this approach for designing of TSC controllers for FSMs may be rather efficient from the hardware point of view.

In [1], a method for a PLA implementation of FSMs based on this approach was proposed. This implementation uses orthogonality of transition functions for FSMs of microcontrollers. It results in a drastic reduction of the required overheads.

Approach [2] for synthesis of self-checking logical circuits by FPGA is based on dual-rail implementations of the hardware to be checked. According to this approach the FPGA-based self-checking FSM can be implemented in a form of a combination of the dual-rail FSM and the dual-rail SOM-based checker (SOM-checker). In this case, output words of the FSM have to be encoded by an error-detecting code for detecting unidirectional errors (such as the Berger code). Needless to say that such an implementation is critical from the point of view of resulting overhead due to both the dual-rail design and the Berger encoding.

A new architecture that does not require any encoding of codewords and allows a single-rail design of the FSMs was suggested in [3]. In this architecture, both the self-checking FSM and the

* This research was supported by BSF under grant No. 9800154.

SOM checker can be considered as a combination of two blocks: an "Evolution" block and an "Execution" block. The main advantage of the architecture [3], i.e., the minimized overhead, is based on comparing outputs of Evolution blocks of the FSM and it's SOM-checker. These two vectors must be equal and belong to the same 1-out-M code.

The present work can be considered as a continuation of [3]. Here we evaluate the architecture, proposed in [3] from two points of view: the required overhead and the fault latency. Estimations of the overhead and the latency, presented in this paper, provides for guidelines for synthesis of FSMs with required parameters.

2. ARCHITECTURE OF A SELF-CHECKING FSM

For the FSM to be implemented we denote a set of binary inputs as $X=\{x_1, \ldots, x_L\}$ and a set of binary outputs (microoperations) as $Y=\{y_1, \ldots, y_N\}$. An output codeword of the FSM (a microinstruction) is an N-component vector formed by N outputs. Each "1" in the vector reflects its corresponding microoperation; each "0" means that the corresponding microoperation is absent in the microinstruction. A set of possible output codewords of the FSM constitutes a set of microinstructions. The number of these microinstructions will be denoted M. The example of a transition table is presented in Table 1.

Table 1. Transition table of a FSM

a_m	a_s	$X(a_m, a_s)$	$Y(a_m, a_s)$	h
a_1	a_2	x_1x_2	y_1, y_2	1
	a_4	$x_1\bar{x}_2x_3$	y_4	2
	a_1	$x_1\bar{x}_2\bar{x}_3$	–	3
	a_3	$\bar{x}_1$	y_2	4
a_2	a_4	1	y_1, y_4	5
a_3	a_1	$x_4\bar{x}_1$	y_1, y_3	6
	a_3	$\bar{x}_4\bar{x}_1$	–	7
	a_4	$\bar{x}_4$	y_1, y_4	8
a_4	a_5	x_2	y_5, y_6	9
	a_1	$\bar{x}_2$	–	10
a_5	a_1	1	y_1, y_3	11

In this Table 1:

- a_m - present state,
- a_s - next state,
- $X(a_m, a_s)$ – a transition function, i.e. a Boolean function, which is equal to 1 when FSM makes the transition from state a_m to state a_s.
- $Y(a_m, a_s)$ – a list of microoperations, which are equal 1 on the transition of the FSM from a_m to a_s,
- h – a serial number of the FSM transition.
- We will use the following notations:
- L – number of input variables; L=4 for our example;
- N – number of output variables; N=6 for our example;
- H – number of transition functions; H=11 for our example;
- M – number of microinstructions; M=7 for our e;
- R – number of internal states; R=5 for our example.

A schematic diagram of the architecture of a totally self-checking FSM (TSC FSM) is shown in Figure 1. The TSC FSM comprises two portions: the self-checking FSM and the SOM-checker. In turn, each of these portions consists of two blocks: the Evolution block and the Execution block.

For this architecture outputs of the Evolution block of the FSM (EVFSM) and the Evolution block of the SOM checker (EVC) are to be compared for error detection. The output vector of EVFSM is formed by M binary one-rail outputs. Output vectors of EVC comprise M dual-rail-coded outputs.

For comparing, the EVFSM and the EVC output vectors are subjected to a match detection operation to produce a resulting vector, which is then fed to the Execution block of the SOM-checker. If the two vectors are equal, the resulting vector is equal to the EVC output vector. If they are not, the EXC will receive a predetermined faulty dual-rail vector. An example of the match detection function can be found in Table 1 of paper [3].

2.1. Self-checking FSM

Inputs of the Evolution block of the FSM (EVFSM) comprise working inputs X of the TFC FSM and output memory signals $T = (t_1 \ldots t_r)$. Outputs of the EVFSM correspond to signals of microinstructions. At each clock, one and only one microinstruction is equal to 1, which means that EVFSM outputs are codewords of the 1-out-of-M code.

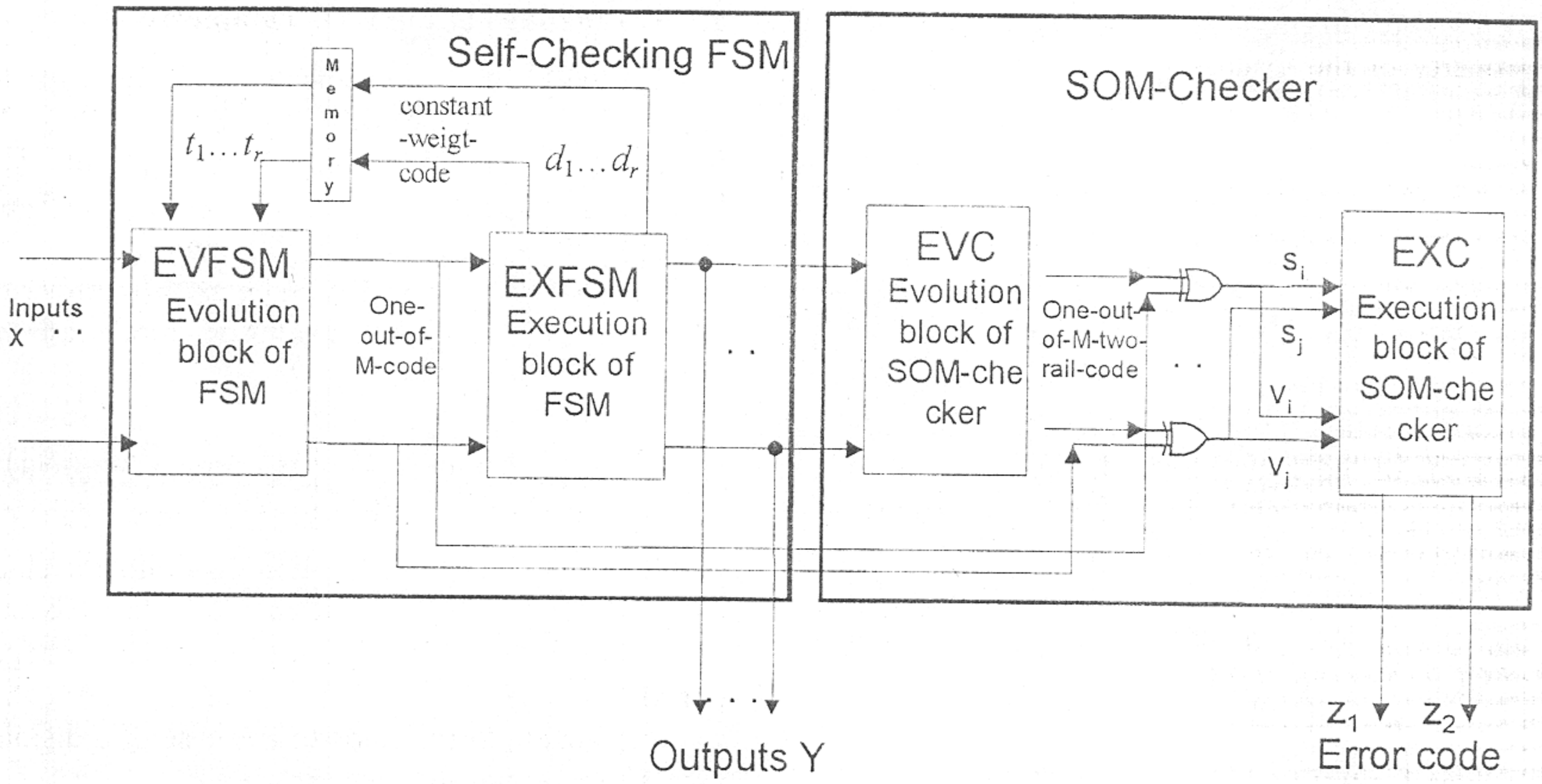


Figure. 1. A schematic diagram of the architecture of a totally self-checking FSM

The EVFSM is implemented as a tree, wherein each of the nodes is either a pre-designed Configurable Logic Block (CLB), or a fan-out. Each CLB is designed for implementation of either a sum of two product terms of g variables, or an AND-function of 2g variables. We use the Xilinx-4000 series architecture [4] for implementation of the proposed self-checking scheme. In this case the number of inputs of the CLB is equal to 8, which means g = 4. The Execution block of the FSM (EXFSM) comprises OR-assembling of EVFSM outputs. Outputs of EXFSM are output signals Y of the FSM and input memory signals D = $(d_1 \ldots . d_r)$. The memory signals are coded by a "constant weight code" [1].

2.2. SOM-based TSC checker

The SOM checker comprises the Evolution (EVC) and the Execution (EXC) blocks. The EVC implements all minterms of the SOM, while the EXC assembles these minterms to implement the checker's function. A schematic diagram of the SOM checker with the Evolution and the Execution blocks are shown in Figure 2.

The EVC is built as a self-checking tree with "AND" nodes for implementation of "long" product terms and "fork" nodes actually implemented by regular fan-outs. The EXC comprises either "1-out-of-2g", or "(2g-1)-out-of-2g" cells (CLBs) combining all minterms, coming from the EVC.

It is proposed to use the following pre-designed A-Cells and O-cells for implementation of the above-mentioned nodes of the checker.

A-Cell implements nodes of the "AND" type. It has two cascade inputs, four functional inputs for implementation of minterms of the SOM, and two cascade outputs.

O-Cell implements nodes of either "g-out-of-2g" or "(2g-1)-out-of-2g" types. These cells have 8 inputs and 1 output.

The EVC is a self-checking two-rail tree comprising a number of A-Cells. This tree is constructed in such a way that in the case of proper functioning of both the FSM and the checker one and only one dual-rail output (S_i, V_i) will have value (1,0). All the remaining outputs will have value (0,1). Outputs of the EVC tree are inputs of the EXC of the SOM-checker. Each of the two-rail outputs of the EVC corresponds to a certain microinstruction of the original FSM.

The EXC comprises two components consisting of O-Cells. All S-outputs of the EVC serve as inputs of the first component of the EXC. This component is implemented as a converging "1-out-of-M" multilevel tree. All V-outputs of the EVC are inputs to the second component of the EXC, which is implemented as a converging "(1-M)-out-of-M" multilevel tree (see Figure 2).

When, at a certain clock, the FSM generates a microinstruction, the EVC produces the corresponding dual-rail 1-out-of-M output vector. Each dual-rail output vector of the EVC is compared with a corresponding single-rail output vector of the EVFSM using the above-mentioned

match detection function. The totally self-checking property of the discussed architecture was proven in [3].

3. ESTIMATIONS FOR THE REQUIRED HARDWARE OVERHEADS

Overheads for the proposed architecture can be estimated by comparing numbers of CLBs, required for FSM implementations without self-error detection ability and for the self-checking versions of the same FSMs. It should be noted that the proposed architecture does not require any redundancy for the FSM itself. Therefore, the difference between the two above-mentioned FSM implementations equals to a complexity (number of CLBs) of the SOM-checker.

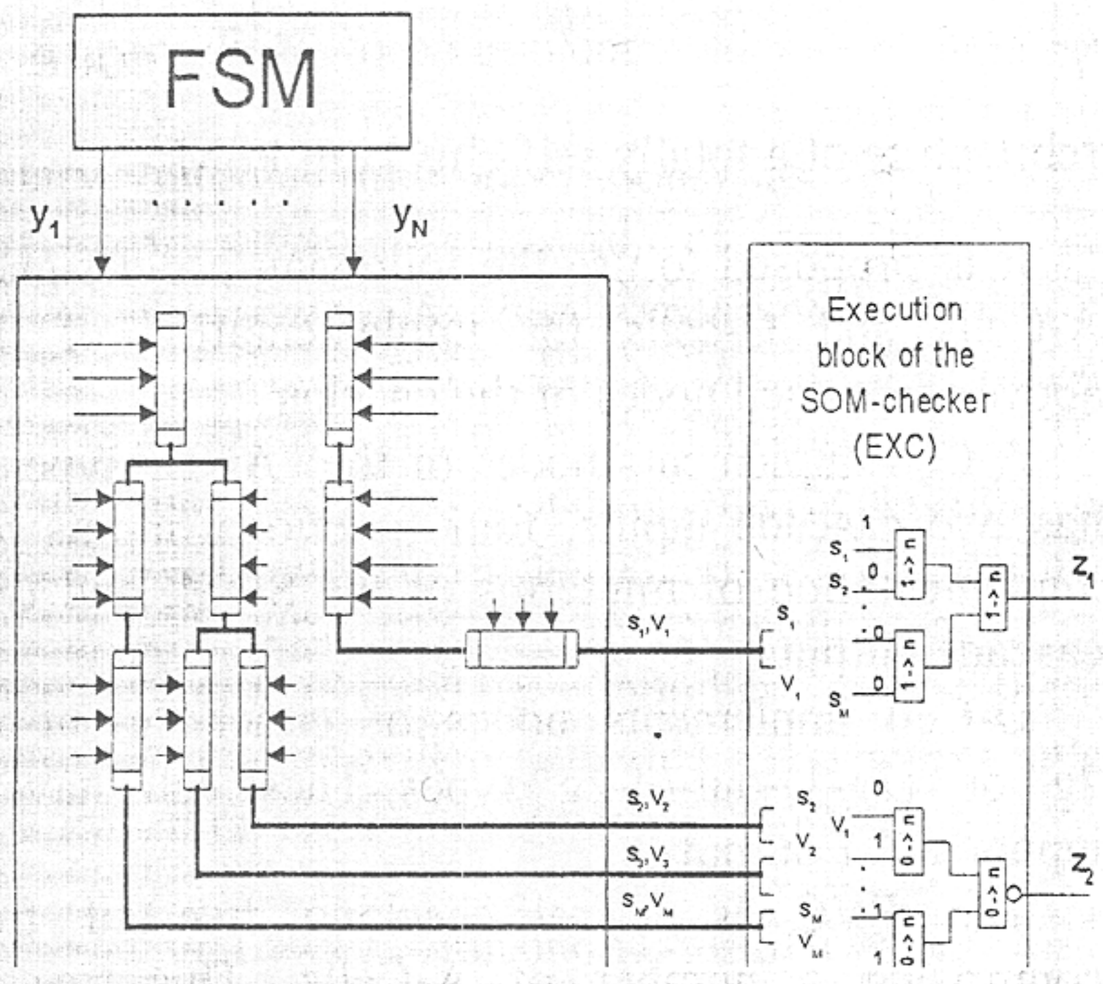


Figure 2. Schematic diagram of the SOM-checker

The goal of this chapter is to estimate the checker's expected complexity as a function of two parameters: number M of microinstructions and length N of microinstructions. Obviously, we cannot give the exact formula of the overhead, since the checker's complexity depends on the SOM function. But approximate formulae presented below allow estimation the required overhead even before performing the synthesis of the FSM.

The complexity of the SOM-checker will be estimated as a sum of complexities of its two components: EVC and EXC.

3.1. Estimations of the EVC complexity

The EVC tree implements a system of M logical functions of N variables. Each of these M functions is defined by a unique minterm. The EVC tree comprises CLBs having g dual-rail inputs (g=4) [4] and one dual-rail output.

The upper bound of the complexity (number of CLBs) S(EVC) of the Evolution block of the SOM-checker is:

$$S_{max}(EVC) = \left[\frac{N}{g-1}\right] M. \qquad (1)$$

(here g is decreased by 1 in denominator since one dual-rail input is used for the cascade connection between CLBs; [a] denotes an integer part of a.

This bound corresponds to the case of a disjoint implementation of the minterms.

For the lower bound on complexity we assume that:

- the number of CLBs at the first level of the EVC tree is equal to M: $A_1 = M$.
- each level of the EVC tree should comprise CLBs implementing the maximal number of different (g-1)-minterms. Then the lower bound on a number of CLBs at level i of the EVC tree can be computed as follows:

$$A_{i+1} = \left[\frac{A_i}{2^{g-1}}\right] + 1 \qquad (2)$$

and we have:

$$S_{min}(EVC) = \sum_{i=1}^{\left[\frac{N}{g-1}\right]+1} A_i \qquad (3)$$

3.2. Estimations of the EXC complexity

As have been mentioned above, EXC consists of two equal converging multilevel trees. One of these trees implements a "1-out-of-2g" function, while the second implements a function "(2g-1)-out-of-2g".

The complexity of the EXC can be computed as:

$$S(EXC) = 2\,(B_1 + \ldots + B_k + \ldots + B_K), \qquad (4)$$

where B_k is the number of CLBs at each k-level of the multilevel tree and $K = [\log_{2g} M]+1$ and

B_k can be computed using the following recursive expression:

$$B_1 = \left[\frac{M}{2g}\right]; \; B_k = \left[\frac{B_{k-1} + (M - 2gB_{k-1})}{2g}\right],$$

$k = 2, \ldots K.$ (5)

The sequence $B_1, \ldots, B_K$, which defines the EXC complexity, converges rapidly to 1. All members of this sequence turn to 1 at step $j = \left[\frac{\log_2 M}{g-1}\right]$ and stay equal to 1, for all $i > j$. This fact allows assessing an effectiveness of the proposed estimations. It is obvious, that for any N and M, $\left[\frac{N}{g-1}\right] \geq \left[\frac{\log_2 M}{g-1}\right]$. In the case of equality the gap between lower and upper bounds is minimal, which means that the accuracy of the proposed estimations is maximal. In this situation the real EXC complexity can be estimated by it's upper bound. The accuracy of the estimations decreases as the difference between the two sides of the inequality increases.

These bounds were used for estimation of complexity of SOM checkers for a number of benchmarks. Results of the computation are presented in Table 2, Section 5 and enable comparison of the estimated complexities with the real ones.

4. ESTIMATIONS OF FAULT LATENCIES

In this section, we present estimations of time between the occurrence of a fault and its manifestation (appearance of the error in a microinstruction due to that fault). This time is called the fault latency. We will describe now a method for estimation of latencies for FSM-based controllers. This method is applied to the proposed architecture and is illustrated by the example of FSM defined by Table 1.

Let inputs be independent random binary variables with probabilities $\Pr(x_i=1) = p_l$, $\Pr(x_i=0) = \Pr(\bar{x}_i=1) = q_l = 1 - p_l$, $l=1,\ldots L$.

Then the behavior of a fault free FSM can be described by the Markov chain with R states, (where R is the number of internal states of the FSM) with transition probabilities matrix $\left(p_{ms}^{(1)}\right)$ [5], (where the upper index $^{(1)}$ shows that we deal with the fault-free case). For our example we have:

$$\left(p_{ms}^{(1)}\right) = \begin{pmatrix} p_1q_2q_3 & p_1p_2 & q_1 & p_1q_2p_3 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ p_1p_4 & 0 & q_1p_4 & q_4 & 0 \\ q_2 & 0 & 0 & 0 & p_2 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (6)$$

To investigate the behavior of the FSM with a fault (faults), we will construct a second Markov chain with the following properties:

(a) the number of the states in the new chain is equal to R + 1. We define the additional (R + 1)-th state as an *absorbing* state;

(b) the chain moves to (R + 1)-th state when the fault is manifested by a distortion of the output microinstruction.

(c) if faults are absent, the behavior of the second chain within the first R states does not differ from the behavior of the first chain.

Thus, if a fault has occurred, the number of steps before entering the absorbing state is the latency value for that fault.

Therefore, the first step of our task will be to compute the transient probability matrix $\left(p_{ms}^{(2)}\right)$ for the second chain. If the stack-at-one fault for variable x_l occurs ($x_l/1$), in Table 1 each literal x_l should be replaced by "1". It will lead to the loss of orthogonal of product terms. For our example the following two matrices correspond to input variable x_1:

$$\left(p_{ms}^{(2)}\Big|_{x_1=1}\right) = \begin{pmatrix} q_2q_3 & p_2 & 0 & q_2p_3 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ p_4 & 0 & 0 & q_4 & 0 & 0 \\ q_2 & 0 & 0 & 0 & p_2 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad (7)$$

$$\left(p_{ms}^{(2)}\Big|_{x_1=0}\right) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & q_4 & 0 & p_4 \\ q_2 & 0 & 0 & 0 & p_2 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \quad (8)$$

Then we have for the unconditional transition probabilities matrix:

$$\left(p_{ms}^{(2)}\right) = p_1\left(p_{ms}^{(2)}\Big|_{x_1=1}\right) + q_1\left(p_{ms}^{(2)}\Big|_{x_1=0}\right) =$$

$$\begin{pmatrix} p_1q_2q_3 & p_1p_2 & 0 & p_1q_2p_3 & 0 & q_1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ p_1p_4 & 0 & 0 & q_4 & 0 & q_1p_4 \\ q_2 & 0 & 0 & 0 & p_2 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \quad (9)$$

Denote the probability distribution for internal states by the moment when the fault occurred as:

$$\left(p_0^{(2)}\right) = \left(p_1(0), \ldots, p_5(0), \ 0\right). \quad (10)$$

Then, the probability that the chain has entered the absorbing state by the k-th step, is:

$$F(k) = p_{R+1}(k) = \left(p_0^{(2)}\right)\left(p_{sm}^{(2)}\right)^k (0, \ldots, 0, 1)^T \quad (11)$$

Function F(k) is the distribution function for the latency of the selected fault.

Figure 3 presents latency curve 1-F1(k) for the above example, where the function F1(k) is computed for the particular fault (x_l /1), and where $p_l = q_l = 1/2$ for all l.

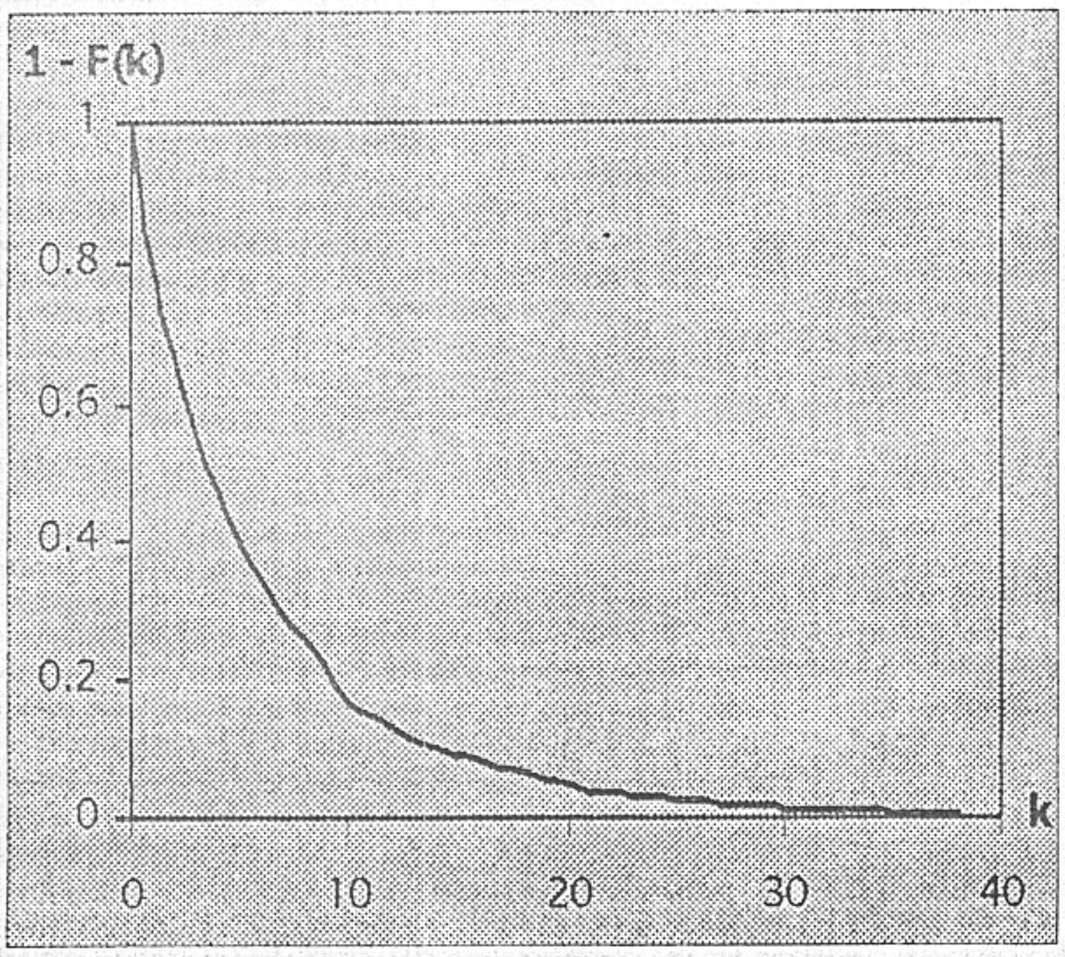


Figure 3. Latency curve for the fault x_l/1.

The average latency duration of the fault can be computed as in [6]:

$$\bar{k} = \left(p_0^{(1)}\right)\left(I - \left(q_{ms}^{(2)}\right)\right)^{-1}(1, \ldots, 1)^T, \quad (12)$$

where I is unitary $R \times R$ matrix, $\left(q_{ms}^{(2)}\right)$ is the submatrix of the matrix $\left(p_{ms}^{(2)}\right)$, obtained by eliminating the extreme right column and the lowest row.

Let $\{f_i\}_{i=1}^{Q}$ be a set of faults with probabilities $\Pr(f_i) = s_i$. Then the statistical characteristics of the whole set can be obtained as follows:

$$F(k) = \sum_{i=1}^{Q} s_i F_i(k), \ \bar{k} = \sum_{i=1}^{Q} s_i \bar{k}_i, \quad (13)$$

where $F(k)$ and $\bar{k}_i$ are the distribution function and the average value of the latency for the i-th fault.

Using the same method, a latency curve can be obtained for each of the "stuck" faults. The average latency curve for single stuck faults, calculated by equation (13) for the above example, is given in Figure 4.

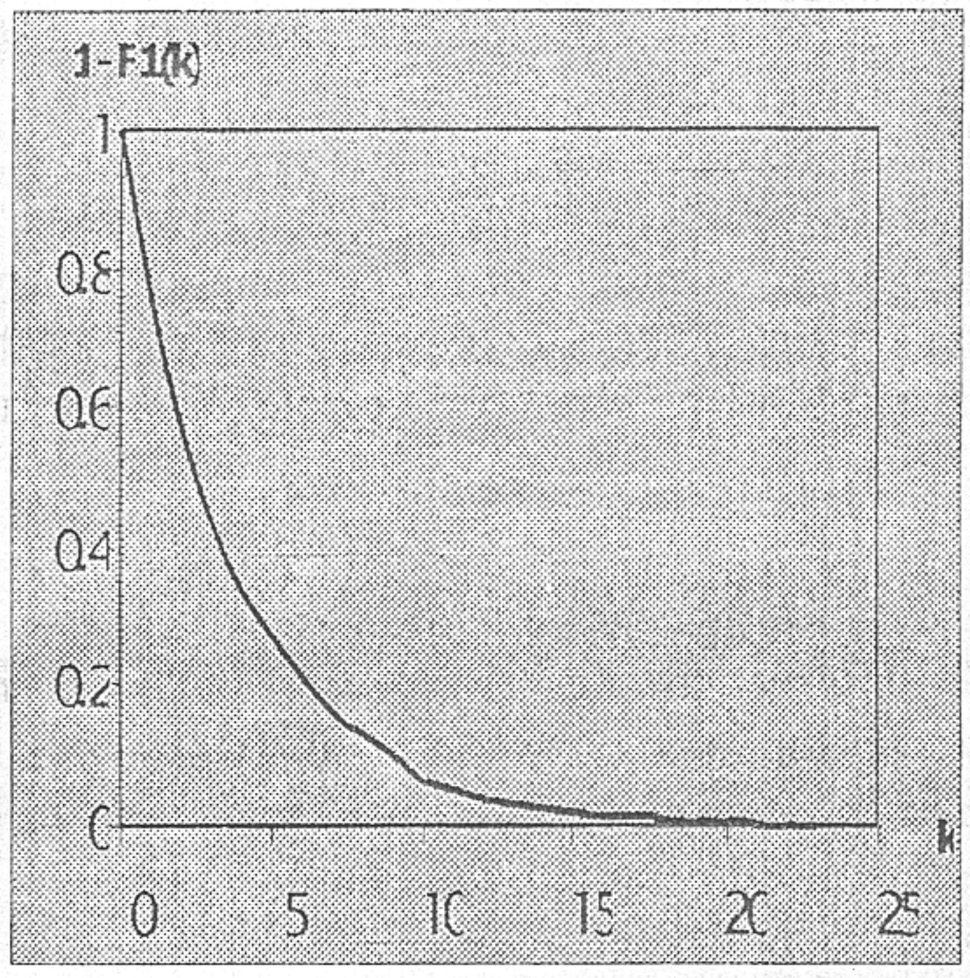


Figure 4. Average fault latency curve for the FSM defined by Table 1.

Concerning the results, the following comments can be made:

- The probability, that the fault will not be detected during a considerable number of steps, tends to 0. We therefore believe that the proposed probabilistic analysis clearly identifies the TSC property of the discussed FSM (at least with respect to the considered faults) as illustrated in Figure 4;
- We note that a large difference can be observed between the two curves shown in Figs 3 and 4. This difference can be explained by the fact that variable x_1 appears quite often in Table 1. So, the corresponding stack fault can manifest itself rather quickly. Faults related to variables that rarely appear in the FSM transition table have larger latency, since it is more difficult to provoke them.

The average fault latency values for several FSM benchmarks computed by (7) and (8) are presented in Table 2.

5. BENCHMARKS RESULTS

Seven FSM controllers were used as benchmarks in our research. The FSMs were developed by undergraduate Computer Engineering students as their final projects. Each FSM describes functioning of a special purpose microprocessor. We use the Xilinx-4000 series architecture [4] for calculation of overheads for these benchmarks.

Results for benchmarks are presented in Tab. 2. In this table:

- L – number of input lines,
- N – number of output lines,
- H – number of product terms (transitions),
- R – number of states of the FSM,
- M – number of possible codewords (micro-instructions).
- S^{FSM} and S^{C} are complexities (numbers of CLBs) of the FSM and the SOM-checker, correspondingly.
- S^{C}_{min} and S^{C}_{max} – minimal and maximal SOM-checker's complexity calculated using formulae from Section 3.

$$\Omega = S^{C}/S^{FSM}*100\%;$$

- k_{av} – average fault latency.

One can see from this table that the proposed approach for design of totally self-checking microcontrollers results in overheads which is not exceed 65% and the small average latencies for single stuck-at faults.

6. CONCLUSION

We have proposed herein a novel technique for the synthesis of self-checking FPGA-based microcontrollers. By utilizing several intrinsic features of the corresponding FSMs, the proposed architecture allows implementation of FSMs by a single-rail scheme without any additional encoding of output words. This results in considerable reduction of the required overhead. The Markovian analysis was used for investigation of fault latencies. The approach was confirmed by number of FSM benchmarks. Benchmarks results indicate that the proposed approach for the design of self-checking microcontrollers is efficient from the points of view of required overheads and average fault latencies.

REFERENCES

[1]. Levin, M. Karpovsky. On-line Self-Checking of Microprogram Control Units. 4th *International On-line Testing Conference*, Capri, 1998, Compendium of papers, pp. 153-159.

[2]. L. Burress and P. K. Lala. On-line Testable Logic Design for FPGA Implementation. *Proc. of 1997 International Test Conference*, pp. 471-478.

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

[4]. Xilinx, The Programmable Logic, *Data Book*, 1996.

[5]. G. Hatchel, E. Macii, A.Pardo and F.Somenzi, Marcovian Analysis of Large Finite State Machines. *IEEE Transaction on Computer-Added Design of Integrated Circuits and Systems*, vol.15, **12**, Dec. 1996

[6]. J. Kemeny, J. Snell, Finite Markov chains. D. Van Nostrand Company, 1967.

Table 2. Results for FSM benchmarks implemented by Xilinx-4000 series architecture

Name	L	N	R	H	M	S^{FSM}	S^{C}	S^{C}_{min}	S^{C}_{max}	Ω	$\kappa_{\alpha\omega}$
big	18	28	17	185	17	124	71	22	153	57%	747
bs	19	13	17	185	17	125	43	20	68	34%	247
acdl	16	27	22	214	23	158	89	28	207	56%	456
cow	49	24	24	261	18	262	84	23	144	32%	366
v1_6	14	18	17	169	17	74	45	21	102	60%	237
v1_10	15	18	18	264	18	102	49	22	108	48%	300
v11_20	14	29	18	367	17	110	71	22	153	65%	360