Sequential Circuits Applicable for Detecting Different Types of Faults[1]

I. Levin[a], V. Sinelnikov[a], M. Karpovsky[b] , S. Ostanin[a]
[a]*Tel-Aviv University, Ramat Aviv, 69978, Israel*
[b]*Boston University, 8 Saint Mary's Street, Boston, MA 02215, USA*
ilia1@post.tau.ac.il, sinel@hait.ac.il, markkar@bu.edu, sostanin@post.tau.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, which will be detected as error by the checker after a number of clock cycles. In the case of a transient fault, the SSC is able to survive and to return to normal operation after a number of clock cycles.

A novel universal architecture of self-checking SSCs enabling to overcome the above contradiction is proposed. This architecture can be adopted both for reduction of the fault latency of a permanent fault and for increasing the SSC survivability with respect to a transient fault. A method for SSC synthesis for the proposed architecture is presented. This method is oriented to FPGA implementation.

1. Introduction

Two main approaches can be identified in the design of self-checking SSCs. The first one is based on applying special techniques to observe state transitions or a control flow. These techniques range from state assignment by codewords of error-detection codes to control flow monitoring by signature analysis [Leveugle 90, Noubir 96] or special monitoring machines [Parekhji 95]. Usually, these techniques lead to considerable overhead.

The second approach is based on checking the SSC's outputs without direct checking of the memory [Ozguner 77, Diaz 79]. When these techniques are applied, a fault that leads to a non-code state vector is detected in the next clock cycle. This property can be achieved by introducing an additional overhead.

The present paper investigates the behavior of the SSC without memory checking. We deal with implementations where both the next state and output equations are unate [Lala 00] 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 using these implementations results in a considerable reduction in overhead [Matrosova 00]. Furthermore, and that is substantial in the present work, such SSCs can function properly with the presence of a fault and even recover from the fault [Levin 01]. This property is called self-healing.

Let a fault occurs within an SSC. Known self-checking SSCs architectures provide either immediate or next-clock fault detection after the appearance of the test vector of this fault [Lala 00]. Such a requirement is reasonable for permanent faults because such faults have to be detected as soon as possible. However, 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 error detection, both the fault and its manifestations may disappear, and the SSC may become fault-free again. Should we use any one of the known approaches, we would mark an SSC as erroneous prematurely, although it could survive successfully. In other words, the known approaches for synthesis of self-checking SSCs are mostly oriented to permanent faults, while usually declaring that both intermitted/transient faults can be detected as well.

The fault latency is the number of clocks required for detecting a fault after it occurs. Reduction of the fault latency is one of the well-known challenges of self-checking design. The above-mentioned self-healing property correlates with the period of time when a fault is present but has not yet been detected, i.e. with the latency. Indeed, if any fault is detected either immediately, or on the next clock pulse after its manifestation, the latency looks minimal, but the SSC does not have enough time to recover. Reduction of the latency leads to a decrease in the self-healing ability. Since reduction of the fault latency relates to permanent faults, while the self-healing ability characterizes transient faults, the above contradiction can

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

be considered as a contradiction between the SSC's reactions to the two different types of faults.

In the present paper we propose a technique to overcome the above contradiction. We develop an architecture for implementing self-checking SSCs, which is applicable to both types of faults: permanent and transient. Moreover, it can be adapted to both types of 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 [Matrosova 00]; b) the Match Detector architecture that does not require any coding of output vectors [Levin 99].

We investigate the architecture from the point of view of fault latency and the self-healing ability for both kinds of faults, assuming that the resulting on-line checking SSC will be implemented by an FPGA.

This paper is organized as follows. Section 2 gives basic definitions. Description of the self-healing SSC is given in Section 3. Universal Match-Detector (UMD) architecture is proposed in Section 4. Implementing PTC and ExSSC are given in Section 5. Benchmarks results are given in Section 6. Conclusions are presented in Section 7.

2. Basic Definitions

We describe an SSC according to the Mealy model.

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

We will use the following notations for the SSC: $ff_1,...,ff_{N_y}$ - D-flip-flops, $x=\{x_1,...,x_{N_x}\}$ - input variables of SSC, $y=\{y_1,...,y_{N_y}\}$ - current state variables, $Y=\{Y_1,...,Y_{N_y}\}$ - next state variables, $Z=\{Z_1,...,Z_{N_z}\}$ - output variables.

An SSC that is monotonic in state variables belongs to the class of partially monotonic SSC [Matrosova 00]. Such an SSC is also called state monotonic [Levin 01]. A state-monotonic SSC can be presented in the form of a sum-of-products, which is unate in state variables [Mago 73]. Table 1 shows an example of a state monotonic SSC. $\mathbf{P}=\{p_1,...,p_{N_p}\}$ is a set of product terms.

Table 1. Example of a state monotonic SSC

P	$x_1x_2x_3x_4$	$y_1y_2y_3$	$Y_1Y_2Y_3$	$Z_1Z_2Z_3Z_4$	**O**
p_1	11--	1--	001	1010	o_2
p_2	101-	1--	001	0010	o_3
p_3	100-	1--	100	0000	o_1
p_4	0---	1--	010	0101	o_4
p_5	1--1	-1-	100	0101	o_4
p_6	1--0	-1-	001	1010	o_2
p_7	0---	-1-	001	1010	o_2
p_8	-1--	--1	100	0101	o_4
p_9	-0--	--1	100	0000	o_1

The basis of target implementation of the SSC is LUT-based FPGA comprising Configurable Logic Blocks (CLBs). The fault model used in this paper is general model of single cell faults. We assume that at most one CLB can produce a faulty output. The circuit primary inputs are considered to be fault-free.

3. Self-healing SSC

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.

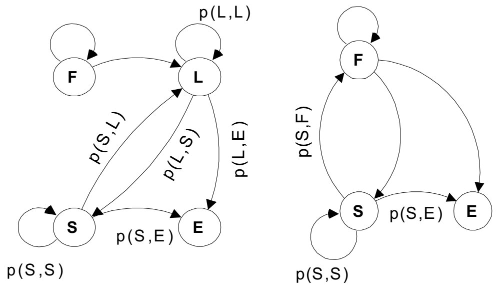
An MTG describing the SSC behavior in a presence of a permanent fault is shown in Figure 1a.

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



a) **b)**

Figure 1. Mode Transition Graph for: a) a Permanent Fault; b) a Transient Fault

output vector has been produced. The circuit is able to move to this mode either from the silent mode or from the latent mode.

The number of clocks, required for detecting a permanent fault (transition to **E** mode) after the fault occurs (transition from **F** mode to **L** mode) is called the fault latency.

In the case of transient fault, the dynamics of the above-mentioned modes are shown in Figure 1b.

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.

The number of clocks, required for the disappearance of the consequences of the transient fault (transition from **S** mode to **F** mode) after the fault occurs (transition from **F** mode to **S** mode), is called the self-healing time.

4. Universal Match Detector Architecture

The proposed universal self-checking architecture is based on the Match Detector (MD) architecture presented in [Levin 99]. The main advantage of the MD architecture is that it does not require a redundant portion, which is always necessary when using a standard solution of self-checking FSMs. The MD architecture does not require encoding of any output vectors and consequently provides solutions having a relatively low overhead. The MD architecture is based on using a Match Detector within the checker. The checker that includes the Match Detector is called an MD-checker.

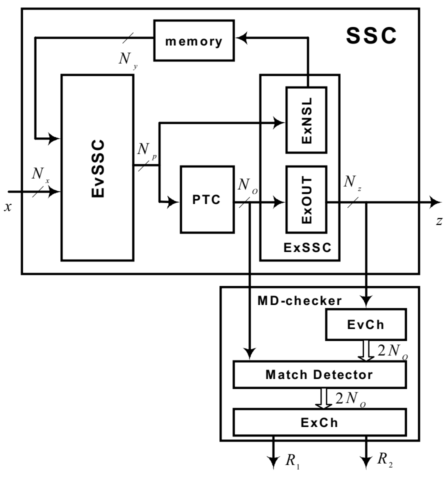


Figure 2. The Schematic Diagram of UMD Architecture

The Universal Match Detector (UMD) architecture proposed in this paper is a natural generalization of the architecture described in [Levin 99]. A schematic diagram of the UMD architecture is shown in Figure 2.

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 $Y = \{Y_1, \ldots, Y_{N_y}\}$. 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.

The 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. 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.

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 both equal 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.

5. Implementing PTC and ExSSC

In this section, we discuss the influence of different types of implementing blocks PTC, ExOUT and ExNSL on both the self-healing property and the latency of the UMD architecture. We show that changing the basis of implementation between the *OR* function and the 1-*out-of-n* 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 properties of the MD architecture will detect the fault immediately, so that self-healing cannot be achieved. Thus, the ExOUT block has to be implemented by using 1-*out-of-n* functions, regardless of 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), both the PTC and the ExNSL are to be implemented by *OR* elements. In turn, in the case of permanent fault orientation (minimizing the fault latency) both of blocks are to be implemented 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 its two 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.

The implementation of the SSC shown in Table 1 is illustrated in Figure 3. Elements of ExNSL and PTC blocks implement either the *OR* functions ($\vee$) or the 1-*out-of-n* functions ($\oplus$) according to the selected solution.

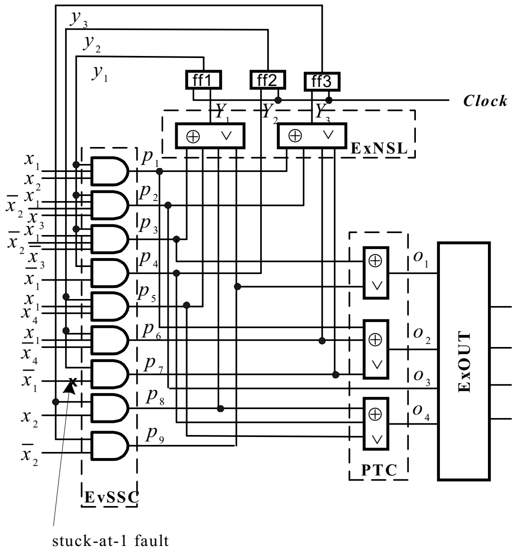


Figure 3. Exemplary SSC implemented according to UMD architecture

Let a *stuck-at*-1 fault occur on the input $\bar{x}_1$ of the element realizing the product term p_7. As was estimated in Section 6, 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 (less than half). 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%.

6. Experimental Results

To arrive at values of fault latencies and self-healing for both of the proposed implementations of the UMD architecture, experiments were performed with benchmarks circuits in which both permanent and transient fault injections were made. 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 injecting the 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. Column T_1 corresponds to the fault latency for a permanent fault in the *OR*-based case, and columns T_2 - for the 1-*out-of-n* based case. $\Omega = 100 * (T_1 - T_2)/T_1$ is the percentage of the latency reduction for the 1-*out-of-n* based case in comparison with latency for the *OR*-based case. T_{det} and T_{heal} are the latency and the time for self-healing of transient faults respectively. Percentages for sequences on which the circuit survived are shown in column ψ .

Experimental results show that the 1-*out-of-n* - based implementation provides a reduction of the permanent faults latency of about 41% as compared with the *OR*-based implementation. On the other hand, in the case of the *OR*-based implementation, the SSC is able to survive with a probability of about 21%.

7. Conclusions

In this paper, we present a novel architecture for self-checking Synchronous Sequential Circuits (SSC) that is based on a Universal Match Detector (UMD). We point out the phenomenon of self-healing of such circuits. Circuits with the self-healing ability are partially monotonic in their state variables. Such circuits are called state monotonic SSCs. On the one hand, the state monotony enables the circuit to be self-healing with respect to 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 to both types of 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. In the case of the predominance of permanent faults these blocks are implemented by using an 1-*out-of-n* function, while in the case where transient faults predominate the same blocks are implemented by using an *OR* function. This difference between the two versions of the same architecture does not affect the overhead. Moreover, the implementation can be changed from one type to the other by reconfiguring the blocks in the UMD architecture.

Table 2. Experimental results

Example	Permanent Faults			Transient Faults					
	OR-based	1-*out-of-n*-based		*OR*-based			1-*out-of-n*-based		
	T_1	T_2	Ω	T_{det}	T_{heal}	Ψ	T_{det}	T_{heal}	Ψ
ex1	6589	4876	25.998	0.171	1.331	10.608	0.194	1.0	4.205
ex6	2124	1577	25.753	0.186	1.366	12.931	0.226	1.0	0.379
s386	11243	4529	59.717	0.207	1.42	14.14	0.226	0	0
s820	5324	4324	18.783	0.204	1.215	5.16	0.221	0	0
sse	12340	9807	20.527	0.194	1.41	16.94	0.226	1.024	1.230
beecount	4253	581	86.339	0.048	1.614	48.72	0.23	1.0	2.11
bbtas	698	323	53.725	1.694	3.151	24.96	0.235	0	0
bbara	9798	5854	40.253	1.644	5.284	42.68	0.248	0	0
dk15	1720	994	42.209	0.181	1.466	14.32	0.22	0	0
Average	6009.9	3651.7	41.48	0.50	2.03	21.16	0.23	0.45	0.88

8. References

[Diaz 79] Diaz, M., and P. Azema, "Unified design of self-checking and fail-safe combinational circuits and sequential machines", IEEE Trans. on Comp., C-28, 1979, pp. 276-281.

[Lala 00] Lala, P., “Self-checking and Fault-Tolerant Digital Design”, Morgan Kaufmann Publishers, San-Francisco / San-Diego / New-York/ Boston/ London/ Sydney/ Tokyo, 2000.

[Leveugle 90] Leveugle, R., and G. Saucier, “Optimized Synthesis of Concurrently Checked Controllers”, IEEE Trans. on Comp., Vol. 39, No. 4, 1990, pp. 419-425.

[Levin 01] Levin I., Matrosova A., Ostanin S., “Survivable Self-checking Sequential Circuits”. Proceedings of IEEE Intl. Symp. on Defect and Fault Tolerance in VLSI Systems (DFT01), October 2001, San Francisco, USA, pp. 395-402.

[Levin 99] Levin, I., 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.

[Mago 73] Mago, G., “Monotone Functions in Sequential Circuits”, IEEE Trans. on Comp., vol. C-22, 1973, pp. 928-933.

[Matrosova 00] Matrosova A., S. Ostanin., “Self-checking FSM Design with Observing only FSM Outputs”, Proc. The 6-th Intl. On-Line Testing Workshop, July 2000, pp.153-154.

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

[Ozguner 77] Ozguner, F., "Design of totally-self-checking asynchronous and synchronous sequential machines", Proc. Int. Symp. Fault-Tolerant Computing, 1977, pp. 124-129.

[Parekhji 95] Parekhji, R.A., G. Venkatesh, S. D. Sherlekar, “Concurrent Error Detection Using Monitoring Machines”, IEEE Design & Test of Comp., Vol. 12, No. 3, 1995, pp. 24-31.