Approach to partially self-checking finite state machine design

G. Lj. Djordjevic, T. R. Stankovic, M. K. Stojcev

Abstract - This paper presents a cost-effective technique of partially self-checking finite state machine (FSM) design. The proposed technique is similar to duplication with comparison, wherein duplicated combinational logic (CL) block of the FSM and comparator act as a function checker that detects any erroneous response of the original CL block. However, instead of realizing checker with full error-detection capability, we select a subset of checker's inputs to implement partial, but simplified function checker. Effectiveness of the technique is evaluated on a set of MCNC 91 benchmark sequential circuits.

I. Introduction

One of the main side-effects of shrinking device sizes, decreasing supply voltages, and high-speed operation of current VLSI IC technologies, is increasing sensitivity to various internal and external noise sources. In this context, protection against temporary faults will be necessary even for applications for which reliability is not a main concern. In order to deal with these problems, the adoption of self-checking approaches as powerful means for concurrent error detection of both temporary and permanent faults can be a viable solution [1-3].

From the reliability point of view, the control unit, typically realized in a form of FSM, is usually the most critical part of the system. Classical approaches for designing self-checking FSMs are based on either duplication, or application of specific error detecting codes. In most cases, these approaches include high hardware cost and high design effort. Several approaches have been taken in the past to design self-checking sequential circuits with lower area overhead in respect to duplication. In [4], a monitoring machine is used in order to monitor the states of the original machine. This technique is effective for delay fault model. However, for the stuck-at fault model, the monitoring machine results in high area overhead. Work presented in [5] uses a control flow checking technique for detection of illegal state transitions, that is based on path signatures. The proposal has low hardware overhead, but this technique has uncertain error detection latency and low fault coverage.

In this paper, we propose a method for designing partially self-checking FSM based on on-line monitoring of FSM operation. We propose and analyze two self-checking scheme. The first monitors FSM state transitions, only, while the second additionally takes into account FSM outputs.

G. Lj. Djordjevic, T. R. Stankovic, M. K. Stojcev are with the Department of Electronics, Faculty of Electronic Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia & Montenegro, E-mail: gdjordj@elfak.ni.ac.yu

II. System Model

An FSM is defined as 6-tuple $M = (X, Y, S, f, g, s_0)$, where X, Y, and S are finite sets of inputs, outputs, and states, respectively, and s_0 is the initial (reset) state of S. The unique mapping: $f : X \times S \rightarrow S$, and $g : X \times S \rightarrow Y$ defines the so-called *state transition function* and *output function*, respectively. An FSM with $n = |S|$ states can be described by a *state transition graph* (STG) defined by a vertex (state) set $S = \{s_1, ..., s_n\}$ and related directed edge set T representing set of transitions from one state to another: $T = \{(s_i, s_j) \mid \exists x \in X,\ f(x, s_i) = s_j\}$. Edges of the STG are labeled with corresponding inputs and output values. An example of STG is shown in Fig. 1 (a).

Let us assume that each state of the FSM, having n states, is encoded with c number of bits – what means that the machine is implemented using c number of flip-flops. The flip-flops may generate 2^c states, out of which only n correspond to *valid* states of the FSM. The remaining $2^c - n$ codes are kept unused. Unused state codes are referred as *invalid*, or *unreachable* states of the FSM. During normal, i.e. fault-free operation, the FSM will never be in any of these states. Similarly, we distinguish between valid and invalid transitions. All edges in the STG defined by a state transition function f of the FSM are said to be *valid*. On the other hand, all other possible transitions, either between valid states or between valid and invalid states, are referred as *invalid transitions*.

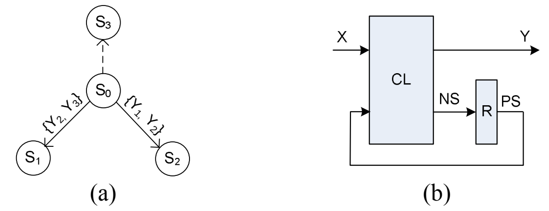


Fig. 1. Finite state machine: (a) state transition graph; (b) structural model.

General structure of the FSM is pictured in Fig. 1(b). It is comprised of a combinational logic, CL, and state register, R. The constituent CL accepts primary inputs, *X*, and present state, *PS*, and generates output signal, *Y*, and next state code, *NS*. In general, concerning FSM model given in Fig. 1(b), faults can appear both in CL and R logic. Our efforts, in this paper, are oriented towards designing of a checker block that can detect transient faults in CL, only.

II. Partially self-checking FSM

Duplication with comparison (DWC) is the simplest form of hardware redundancy that can be used for concurrent error detection (Fig. 2(a)). Both CL copies implement identical output and next state functions, *f* and *g*, respectively. If outputs disagree, the comparator indicates an error. From one hand, it is evident that DWC involves more then 100% hardware redundancy. On the other hand, the DWC technique is simple for implementation and ensures full coverage of all single faults in CL that produce an error. However, in many cost-sensitive applications, due to high hardware overhead, DWC is an unacceptable design solution.

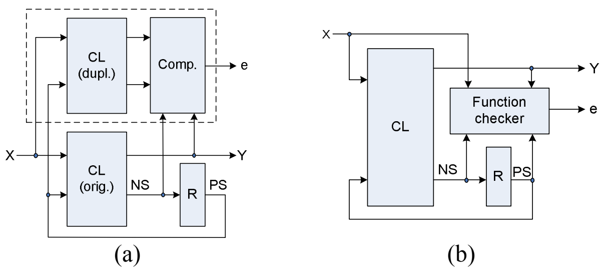


Fig. 2. Duplication with comparison: (a) basic scheme; (b) alternative representation.

An approach to design a more practical concurrent error detection solution, that we adopt in this paper, is to trade-off DWC's high error detection capability for reduced hardware complexity. To this end, we introduce an alternative representation of the DWC scheme, given in Fig. 2b. In this arrangement, two of DWC constituents from Fig. 2(a), the duplicated CL and the comparator, are implemented as a single module, referred as a *function checker* [6]. The function checker is driven by input signals, *X* and *PS*, as well as, output signals, *Y* and *NS*, of the original CL. A single output, *e*, that indicates whether a mapping $(X,PS)\rightarrow(Y,NS)$ is valid or not, is generated at the output of the function checker. Having in mind that the function checker can make a distinction between error-free and any erroneous response of the original CL, we will call this circuit as *total checker,* TC. TC has the same fault coverage as the original DWC scheme, but with probably smaller implementation cost, due to combined synthesis of the duplicated CL and comparator. With this approach, we can obtain a limited hardware reduction in respect to DWC scheme.
Our intention now is oriented towards designing checkers that have hardware complexity less then 100% (with respect to the DWC) at the cost of fault coverage less then 100%, too. A checker with this property we will call *partial checker (PC).*

PC complexity can be simplified by decreasing the number of PC's inputs. Instead to select, on a bit-by-bit basis, the "best" subset of inputs for the checker, we will consider partial checkers that are feed by a subset of CL input/output signal groups, X, PS, NS, and Y. In a set of four signal-groups, there are 16 subsets. After eliminating the empty subset and four subsets that contain input groups, X and PS, only, it remains 11 different meaningful partial checker configurations at our disposal. Next, we will analyze, in more details, two of them: a) Checker that monitors NS and PS signal groups (Fig. 3(a)); and b) checker that in addition to NS and PS signal groups accepts FSM primary output signal group Y, also (Fig. 3(b)). We will call the checker sketched in Fig. 1(a) as a *State-Transition Checker* (STC), and that one given in Fig. 3(b) as *State-Transition-Output Checker* (STOC).

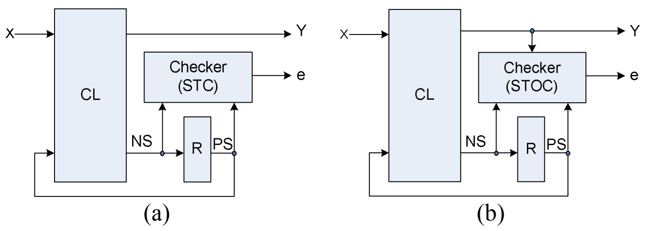
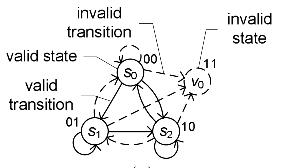


Fig. 3. Partial self-checking: (a) State-transition checker; (b) State transition-output checker.

A. State-transition checker

In order to describe, in a more explicit way, the STC's operation, we modify the corresponding STG into a state graph. The state graph is formed by adding invalid states and invalid transitions into the original STG. For example, the state graph shown in Fig. 4(a) is relative to the original STG presented in Fig. 1(a). Invalid states and invalid transitions are sketched by dashed lines. Since the STC does not accept neither inputs, X, nor outputs, Y, transition labels are omitted, too. STC truth table is given in Fig. 4 (b). The STC signals an erroneous condition under the following condition: PS corresponds to a valid state; but the pair of present-next state codes (PS, NS) is not a valid transition. In contrary, if PS is a valid state and (PS, NS) is valid transition, then the STC output is set to logic 0, indicating error-free FSM operation. The STC does not consider the cases when the present state is invalid, since such a situation can occur iff the previous transition was invalid, and thus detected during the previous clock cycle. This incompleteness in a specification represents degree of freedom that can be used during logic optimization process.



(a)

STC	
Input condition	e
$PS \in S \wedge (PS, NS) \in T$	0
$PS \in S \wedge (PS, NS) \notin T$	1
$PS \in V$	-

(b)

Fig. 4. State-transition checker: (a) model; (b) definition.

In order to estimate STC error detecting capability we will make the following two assumptions: (1) during normal operation, all FSM states are equi-probable, and (2) all erroneous bit vectors that occur at the CL’s outputs, due to a fault, are equi-probable. Let define now an outdegree (d_i) of state s_i as a number of valid transitions exiting this state. Assuming c-bit encoding, the number of invalid transitions from state s_i is 2^c - d_i. The number of possible erroneous transitions as a consequence of fault, at any state, is 2^c - 1. Among them, d_i - 1 transitions are valid, while the remaining 2^c - d_i are invalid. Note that, since the STC is able to make distinction between valid and invalid transitions, any erroneous but valid transition will be considered as fault-free.

Illustration only, assume now that the FSM from Fig 4(a) is set in state S_0. Suppose further that, as response to a current input change, the FSM, due to a fault, erroneously goes to state S_2, instead to state S_1. By analyzing Fig. 4(a) we conclude that a transition (S_0, S_2) is valid one, so the STC will not signals an error. Since the outdegree d_0=2, a probability that STC will detect an erroneous transition from state S_0 is 50%. In general, an *error coverage in state* s_i can be defined as a ratio of invalid transitions exiting that state and the total number of erroneous transitions: cv^{STC} = (2^{c+k} - d_i) / (2^{c+k} - 1). Since all FSM states are equi-probable, STC's error coverage, cv^{STC}, is defined as a mean value of state error coverage:

$$cv^{STC} = \frac{1}{n}\sum_{i=1}^{n} cv_i = \frac{2^c - \hat{d}}{2^c - 1} \qquad (3)$$

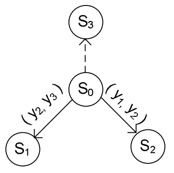
where $\hat{d}$ is average outdegree of FSM's states.

By analyzing Eq. (3) we can draw the following conclusions: a) STC's error detection capability is higher for sparse STG, i.e. for FSM with smaller average state outdegree, $\hat{d}$. b) STC's error coverage increases by increasing the number of encoding bits, c. However, as we increase c, we have to pay a cost for CL hardware complexity augmentation.

B. State-transition-output checker

State-transition-output checker (STOC) is partial FSM checker that extends functionality of STC by additionally taking into account FSM primary outputs (Fig. 3(b)). STOC operation can be described by means of a state graph where each valid transition is labeled with a set of all possible different valid primary output values. Fig. 5(a) shows a part of the state graph with four states and several valid and invalid transitions. The set {Y1, Y2} associated with a transition (S_0, S_2) means that Y1 and Y2 are only correct output values when FSM moves from state S_0 to S_2. The truth table of the STOC is presented in Fig. 5(b). L(PS,NS) denotes a set of all distinct output values associated with the transition (PS, NS).

Logic 1 at the STOC output signals an error in cases when PS is a valid state, but either (PS, NS) is not a valid state transition, or when Y is not a valid output value with respect to the observed transition. The STOC will indicate an error-free FSM operation, when PS is valid state, and when both the pair (PS, NS) corresponds to valid state transition, and Y is a valid output value in respect to that transition. Although the number of erroneous conditions that STOC can capture is larger that then of the STC, it still misses to detect all possible FSM errors. This is direct consequence of the fact that STOC does not takes into account FSM primary inputs. Therefore, there is always a probability that in a case of fault, the resulting erroneous next-state and output value match some valid condition [(PS, NS), Y] that correspond to a given PS and **some** arbitrary input X.



(a)

STOC	
Input condition	e
$PS \in S \wedge (PS, NS) \in T \wedge Y \in L(PS, NS)$	0
$PS \in S \wedge [(PS, NS) \notin T \vee Y \notin L(PS, NS)]$	1
$PS \in V$	-

(b)

Fig. 5. State-transition output checker: (a) model; (b) definition.

Suppose now that the FSM in Fig. 5(a) is in state S_0, and an erroneous value appears at CL’s outputs due to a fault. First, consider a case when the generated next-state (say, S_2) is correct while the output (say, Y3) is incorrect. Since Y3 is not contained in the corresponding set of valid output values, L(S_0, S_2), it will be recognized by the STOC as an error. However, the STOC will not signals an error if the faulty FSM produces Y2 instead of Y1, i.e. if the incorrect output value belongs to the set of valid output values associated with the current state transition. In addition, consider now a case when the next state is erroneous. Suppose that the faulty FSM moves from S_0 to S_1 instead to S_2. Note that such an error could not be detected by the STC since S_1 is a valid state. Whether the STOC, under such circumstances, will react or not depends on the generated output value. For example, if an output value of Y1 is generated, an error will be detected, since for transition (S_0, S_1), the output value Y1 is not valid. Contrary, if a generated output value is Y2, the error cannot be detected, since this value is valid in respect to transition (S_0, S_1).

Starting with identical simplified assumptions that we have already applied for STC, we can now estimate the error coverage of STOC as follows. For a given present state S_i, we define quantity w_i as the number of valid different (c+k)-bit combinations generated at the CL outputs (NS, Y). Note that w_i can be calculated by summing the cardinality of valid output values sets over all S_i's outgoing transitions. For example, for STG in Fig. 4, we have w_0=4. For given c-bit state and k-bit output encoding, the number of possible erroneous (c+k)-bit combinations at the CL's outputs, for any present-state, is 2^{c+k}-1. Among them, w_i-1combinations are valid, while the remaining 2^{c+k}- w_i are invalid. Since STOC is able to differentiate between valid and invalid CL's output combinations, only, its *error coverage in state* S_i is: $cv^{STOC}/cv^{STC} = (2^{c+k} - w_i) / (2^{c+k} - 1)$. STOC's *error coverage* is defined as a mean value of state error coverage:

$$cv^{STOC} = \frac{1}{n}\sum_{i=1}^{n} cv_i^{STOC} = \frac{2^{c+k} - \hat{w}}{2^{c+k} - 1} \tag{4}$$

where $\hat{w}$ is an average over w_i, i=1, …, n.

It follows from (4) that the STOC has a higher error coverage for FSM with a smaller number of alternative output values per present-next state pair. Parameter $\hat{w}$ can be expressed as $\hat{w} = \lambda \cdot d$ where $1 \le \lambda \le 2^k$. The case $\lambda = 1$ corresponds to STG with single output value per each transition. The case $\lambda = 2^k$ is appropriate for STG when each transition is labeled with a set of all possible *k*-bit combinations. Note that $\lambda = 1$ corresponds to Moore, while $\lambda \ge 1$ to Mealy FSM. It is evident from Eq. (4) that the STOC scheme is more effective for Moore FSM.

Finally, we will compare error coverage of STC and STOC. Therefore, define now an improvement factor of STOC over STC as: $\alpha = cv^{STOC}/cv^{STC}$. By substituting expressions for cv^{STC} and cv^{STOC} and assuming that $2^c - 1 \approx 2^c$ and $2^{c+k} - 1 \approx 2^{c+k}$ it is easy to show that:

$$\alpha = \frac{2^{c+k} - \lambda \hat{d}}{2^{c+k} - 2^k \hat{d}} \tag{5}$$

Since $1 \le \lambda \le 2^k$, we can conclude that STC error coverage is never greater then that one of STOC. From one hand, in the extreme case (when in all FSM states, all 2^k *k*-bit combinations at the primary output are valid, $\lambda=2^k$), both partial checkers posses identical error detection capabilities. From the other hand, the improvement factor is largest for λ=1, i.e. for Moore FSM.

III. Experimental Results

A system for automatic synthesis and implementation of partially self-checking FSMs has been developed in order to evaluate the performances of the proposed approach. The system uses SIS 1.2 [7] for logic synthesis and technology mapping together with several specifically developed tools for partial checker generation and fault-coverage analysis. Generated checkers are optimized for area using the SIS script *script.rugged* and then mapped to gates in a given technology library (*lib2.genlib* in this case) in order to obtain an area-minimal implementation in a specific technology. Fault-coverage simulation uses gate-level netlists in order to simulate the effects of single stuck-at faults, and calculates the fault-coverage value.

Table I reports the data for several example FSM taken form the MCNC 91 sequential benchmark suite. For each FSM we have synthesized four partially self-checking circuits and compare them with the circuit based on DWC scheme. The first two circuits are based on STC scheme: one using c_{min}, and one c_{min}+1 bits for state encoding, where c_{min} represents the minimal number of bits needed to encode FSM states. The second two circuits are based on STOC scheme. Results includes the area expressed as percentage of the area needed for DWC scheme (%A) and fault coverage expressed in percentage of errors detected during simulation (%C).

Results are highly FSM dependent. In general, the STOC scheme is more efficient in terms of fault coverage, but with higher area cost with respect to the STC scheme. In both schemes, fault coverage improves with adding one extra bit for state encoding. However, since increase of the number of state bits affects both the CL and checker, the additional area cost is rather large.

TABLE I
Area and fault coverage analysis

FSM	STC				STOC			
	$c=c_{min}$		$c=c_{min}+1$		$c=c_{min}$		$c=c_{min}+1$	
	%A	%C	%A	%C	%A	%C	%A	%C
CSE	24.6	20	64.7	70	55.5	74.2	69.7	90
BBSSE	29.2	37.7	60.3	72	67.6	78.7	99	89
KEYB	37.2	57	81.4	74	41.8	61	95.8	84
DK14	13.3	20	56.5	38	58.5	61	102	81

IV. Conclusions

A method for the synthesis of low cost, partially self-checking FSMs is proposed. The method is based on reduced duplication with comparison scheme used for monitoring operation of FSM combinational logic. Two variants of the proposed scheme are presented and analyzed. Area and fault coverage analysis give promising results. Ongoing research is directed towards optimization of the technique through limiting the number of observed FSM states and selecting appropriate FSM state assignment which will take into account the CL error statistics.

References

[1] P.K. Lala, *Self-Checking and Fault-Tolerant Digital Design, Morgan Kaufmann Publishers*, San Francisco, 2001.

[2] N. K. Jha and S.-J. Wang, “Design and Synthesis of Self-checking VLSI Circuits”, *IEEE Trans. On CAD of Integrated Circuits and Systems,* vol. 12, No. 6, pp. 879-887, June 1993.

[3] C. Bolchini, R. Montandon, F. Salice, D. Sciuto, "Design of VHDL-based totally self-checking finite-state machine and data-path descriptions", *IEEE trans. on VLSI Systems*, Vol. 8, no. 1, Feb. 2000, pp. 98-103.

[4] R. A. Parekhji, G. Venkatesh and S.D. Sherlekar, "Concurrent Error Detection using Monitoring Machines," *IEEE design & Test of Computers,* Vol. 12, pp.24-32, Fall 1995.

[5] R. Leveugle, and G. Saucier “Optimized Synthesis of Concurrently Checked Controller,” *IEEE Trans. Computers,* Vol. 39, pp. 419-425, Apr.1990.

[6] G. Lj. Djordjevic, M. K. Stojcev and T. R. Stankovic, “Approach to partially self-checking combinational circuits design”, *Microelectronics Journal*, Vol. 35 (12) , Dec. 2004, pp. 945-952.

[7] E. M. Sentovich et al., "SIS: A System for Sequential Circuit Synthesis", Tech. Report UCB/ERL M92/41, Electr. Research Lab, Univ. of California, Berkeley, May 1992.