VLSI Design
1994, Vol. 1, No. 4, pp. 335–343
Reprints available directly from the publisher
Photocopying permitted by license only

An Approach for Self-Checking Realization of Interacting Finite State Machines

FADI BUSABA
Department of Electrical Engineering, North Carolina A&T State University, Greensboro

PARAG K. LALA
Department of Electrical Engineering, North Carolina A&T State University, Greensboro

This paper presents a technique for designing interacting finite state machines which will be totally self-checking for any single stuck-at fault. In the proposed technique m-out-of-n codes are used for both primary output and state assignments. In addition, the next state logic (NSL) for each submachine and the output logic (OL) are realized such that any single stuck-at fault results in either single bit error or unidirectional multibit error at the output. The proposed technique does not have any restriction on the way the NSL and the OL are implemented.

Key Words: *State-assignment; Error-detection; Self-checking; Logic synthesis; Unidirectional errors*

Testing of state machines remains one of the most difficult tasks in VLSI test generation/testability area. Several approaches based on scan philosophy [1] e.g. LSSD, Scan-path have been proposed to overcome the problem. Despite their advantages scan-based circuits cannot be tested at normal speed and also require long test times.

An approach for designing fully testable state machines without using scannable memory elements is discussed in [2]. This method adds edges to the initial STG (State Transition Graph) specification to raise the number of states to 2^n where n is the number of the latches in the machine. In addition, the synthesis procedures are hard to apply and may increase the area without any significant improvement in testability. This is because the state machine under test must be placed in a valid state so that with appropriate inputs it can excite the fault and propagate its effect to the external outputs. These steps are slow because for every possible fault the appropriate state and input have to be found, and it might take 2^n cycles to place the circuit in the desired state.

Also, the above mentioned approaches for testing state machines are unable to cope with transient faults. Transient faults are emerging as the dominant failure mode in VLSI circuits [3]. Current testing strategies are incapable of detecting this type of fault since these testing techniques, whether applied externally or implemented as BIST scheme, have been designed to detect only permanent faults. The characteristics of a transient fault require a test strategy which continuously monitors the operation of a circuit so that whenever an invalid output is present, the fault will be automatically detected. Such a test strategy is known as self-checking or on-line testing.

The problem of designing totally self-checking synchronous state machines has been examined by Diaz [4] and by Ozguner [5]. If a Moore machine has m different inputs, p different outputs and n states, Diaz proposed using m-out-of-2m code for input encoding, n/2-out-of-n code for state encoding and p-out-of-2p code for output encoding. Ozguner presented a method for designing totally self-checking Mealy synchronous state machines. In principle, this method is similar to that of Diaz; it uses 1-out-of-n code for state codes and 1-out-of-m code for the input encoding.

The decomposition of a large Finite State Machine (FSM), or a lumped FSM, into smaller interacting FSMs increases performance and reduces area overhead. This is because currently available schemes for state assignment and for logic optimization are ex-

ponentially related to the size of the input, and therefore inefficient for large FSMs. In general, these schemes guarantee better results for small FSMs than for large ones. In addition, if smaller interacting FSMs are used instead of a lumped machine, there will be less logic between latches, thus significantly increasing the speed of operation. As in [6], three types of state machine decomposition are considered in this paper: parallel decomposition (Figure 1a), cascade decomposition (Figure 1b) and arbitrary decomposition (Figure 1c). In Figure 1, 'I' is the primary input, 'O' is the primary output, M1 and M2 are the two submachines, INT1 is the intermediate lines from M1 to M2, INT2 is the intermediate lines from M2 to M1, and OL is the combinational logic for generating the primary outputs.

Parallel decomposition does not allow any interaction between the component submachines. The submachines M1 and M2 are supplied with the same input sequence, but operates independently. The next state lines of the submachines as well as the external inputs are supplied to the combinational circuit OL to generate the output. Cascade decomposition allows unidirectional interaction between the component submachines. Both submachines are driven by the same input sequence, but they do not operate independently; the next state lines of M1 are direct inputs to M2. The information flow from M1 to M2 enables M2 to generate the appropriate output. In this case, it is necessary to guarantee that a fault in M1 will be propagated to M2 and to the primary output. Submachine M1 is usually called the *driving* machine, and submachine M2 is called the *driven* machine. Finally, arbitrary decomposition allows bidirectional interaction between the component submachines. We also assume here that the submachines interconnect through their next state lines, and there is no logic in the interconnection paths. The state lines of both submachines and the primary input lines drive the OL.

Decomposition of state machines was first described in [7], where preserved partitions were used to find cascade decomposition. An approach for finding general decomposition using factoring algorithm has been proposed in Ref. [6]. A recent approach for decomposition ensures that the sum of the number of product terms in the one-hot encoded submachines is minimum [8].

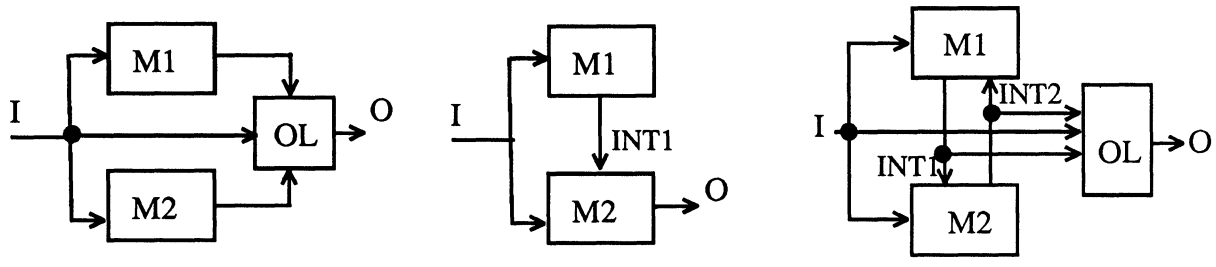
As mentioned previously, currently available schemes for self-checking state machines consider only lumped machines [4] [5]. Even if these schemes are used to make all interacting submachines self-checking, it cannot be guaranteed that the composite machine itself will be self-checking. This is due to the limited controllability/observability of the interconnection lines between the different submachines. Therefore, different schemes have to be applied for designing interacting finite state machines such that they are self-checking. In this paper, we propose a novel technique for designing interacting finite state machines from the state transition graphs of the constituent submachines so that the composite machine will be fault-secure and self-testing for all single stuck-at faults i.e., totally self-checking.

PRELIMINARIES

Def. 1: A vector X **covers** vector Y if X has a 1 in every bit position where Y has a 1. Vector Y is then **covered** by X.

Def. 2: Two vectors O_1 and O_2 are **partially bidirectional** if there exist at least two output bits which are 10 in O_1 and 01 in O_2 or vice versa. For example, O_1 = 1100 and O_2 = 0101 are partially bidirectional whereas O_1 = 1000 and O_2 = 1111 are not.

Def. 3: Two input cubes are called ***m*-directional** if they only differ in position *m* and their corresponding outputs are partially bidirectional.



(a) Parallel (b) Cascade (c) Arbitrary

FIGURE 1 State machine decomposition.

Def. 4: A fault f creates a **unidirectional error** if the correct and the faulty outputs are not partially bi-directional.

Def. 5: A circuit is **fault-secure** for a given set of faults, if for any fault in the set the circuit never produces an incorrect code word at the output for the input code space.

Def. 6: A circuit is **self-testing,** if for every fault from a given set of faults, the circuit produces a non-code word at the output for at least one input code word.

Def. 7: A circuit is totally **self-checking,** if it is both fault-secure and self-testing. The following definition is taken from [10].

Def. 8: A primitive gate in a network is **prime** if none of its inputs can be removed without causing the resulting circuit to be functionally different. A gate is **irredundant** if its removal causes the resulting circuit to be functionally different. A gate level circuit is said to be **prime** if all the gates are prime and **irredundant** if all the gates are irredundant.

Def. 9: A state machine is represented by its **state transition graph** (STG) G(V, E, W(E)) where V is the set of vertices corresponding to the set of states, E is the set of edges that join the vertices corresponding to the transition from one state to another, and W(E) is the set of labels attached to each edge corresponding to the inputs that lead to the transition and the associated outputs. A state machine has an Output Logic (OL) block which generates the primary outputs, the Next State Logic (NSL) block which drives the next state lines, and memory elements. In Mealy machines, the primary inputs as well as the present state lines feed the OL and the NSL. When n latches are used, each state in V is represented by an n-bit vector. An n-bit vector at the output of the latches represents a **valid state** if that state belongs to V. All other n-bit vectors that represent states outside V are **invalid states.** A single stuck-at fault can occur at OL, NSL, memory elements or at the primary outputs. We assume that the primary inputs are fault free. The output of the latches are referred to **state lines** throughout this paper.

State machines can have **combinationally redundant faults** (CRFs) and **sequentially redundant faults** (SRFs) [9]. It was shown in Ref. [9] that these kind of redundancies could be eliminated by logic synthesis restriction. CRFs are due to the presence of lines/wires in the logic circuit that do not contribute to the primary output or the next state functions. SRFs in a *state machine* can be classified into three categories [9]:

1) **Equivalent-SRF:** The fault causes the interchange/creation of equivalent states in the STG.
2) **Invalid-SRF:** the fault does not corrupt any fan-out edge of a valid state in the STG.
3) **Isomorph-SRF:** the fault results in a faulty machine that is isomorphic; i.e., the faulty machine is equivalent to the fault-free machine but with a different encoding.

If parallel decomposition is used, the redundant faults in the resulting interacting submachines will be the same as that in the composite machine since the submachines work independently.

On the other hand, if a cascade decomposition is used, redundant faults in the composite machine can be classified into four categories [10]:

1) A fault in M1 that cannot propagate to the intermediate lines INT1.
2) A fault in M1 that propagates to INT1 but not to primary output O.
3) A fault in M2 that does not propagate to O, but will have if INT1 were completely controllable.
4) A fault in M2 that does not propagate to O even if INT1 were completely controllable.

Clearly, if none of the submachines has redundant faults, then type 1 and 4 of redundant faults will not appear in a cascade decomposition.

Sequential redundant faults in an arbitrary decomposition can be classified into six categories:

1) A fault in M2 that cannot propagate to the intermediate lines INT2.
2) A fault in M1 that does not propagate to O, but will have if INT2 is completely controllable.
3) A fault in M1 that does not propagate to O even if INT2 were completely controllable.
4) A fault in M1 that cannot propagate to the intermediate lines INT1.
5) A fault in M2 that does not propagate to O, but will have if INT1 is completely controllable.
6) A fault in M2 that does not propagate to O even if INT1 were completely controllable.

Redundant faults of type 1, 3, 4 and 6 would not appear if none of the submachine has redundant faults in itself. Redundancies similar to type 2 in cascade decomposition are not present in the arbitrary decomposition described in this paper. Because, if a fault in M1 (M2) propagates to INT1 (INT2), the first erroneous output will be an invalid output (Lemmas 5 and 6 in Section 3). Therefore, if the fault propagates to the INT1 (INT2), the fault will propagate to O provided the OL is prime and irredundant.

IMPLEMENTATION OF NSL AND OL

We use m-out-of-n codes for state assignments for all submachines. The NSL of all submachines and the OL will be driven by arbitrarily encoded primary inputs and m-out-of-n encoded state lines. The NSL and OL have to be implemented such that any single stuck-at fault can only result in either single bit error or unidirectional multibit errors at the output. The implementation of the NSL of each submachine and the OL does not impose any restrictions on the type of gates used or on the minimization procedures (boolean or algebraic). The primary inputs are assumed fault-free.

We will first present lemmas that identify the existence of bidirectional errors [11].

Lemma 1: If there exist two x-bidirectional (Def. 3) input cubes, then a fault at input x may create a bidirectional error at the output.

Lemma 2: If no two input cubes are x-bidirectional, then a single stuck-at fault at an input line can only produce a unidirectional error.

Lemma 3: A fault f creates unidirectional error at the outputs for any input pattern if and only if the number of inversions from the fault site to each of the affected outputs is either even or odd but not both.

The following lemma suggests how bidirectional errors can be eliminated [11].

Lemma 4: If a combinational circuit is designed such that all faults at the inputs which create bidirectional error at the output are removed, then any fault in the circuit, internal or at the inputs, can only result in either single bit error or unidirectional multibit error at the output irrespective of the way the circuit is implemented.

If the inputs to a circuit are m-out-of-n code, then the minimum distance between two codewords is 2. Thus, there does not exist any x-bidirectional input cubes. Consequently, no single stuck-at fault at an input line can produce bidirectional error (Lemma 2); therefore, any single stuck-at fault in the circuit can only cause unidirectional error (Lemma 4). It should be noted that for m-out-of-n input encoding, a single stuck-at-0 (stuck-at-1) at an input line can only cause a transition from 1 to 0 (0 to 1) at the outputs if only the true values of the variables are considered.

Lemma 5: If the inputs to a circuit are m-out-of-n code, then any unidirectional multiple stuck-at faults at the input lines can cause either single bit error or unidirectional multibit error at the output.

Proof: Suppose that the fault free input codeword is C. Since m-out-of-n codes are used for input encoding, a unidirectional multiple stuck-at fault at the input will result in an invalid input codeword, C_f, C covers C_f in case of unidirectional multiple stuck-at-0 faults, and C is covered by C_f in case of unidirectional multiple stuck-at-1 faults. If C covers C_f, then C_f cannot cover any of the input codewords. The output in this case will be all 0's; thus, this multiple stuck-at fault will create unidirectional error. On the other hand, if C_f covers C, C_f might cover other input codewords. Thus, the output will be the bitwise Boolean OR of the outputs corresponding to the input codewords covered by C_f. In this case, only 0 to 1 error can occur at the output lines. Therefore, the error caused by the multiple stuck-at fault is unidirectional. Q.E.D.

Lemma 6: Any arbitrary logic circuit with two sets of inputs, one set ($I_1, I_2, \ldots, I_n$) being m-out-of-n encoded and the other set ($J_1, J_2, \ldots, J_y$) being arbitrarily encoded, can be implemented such that all single stuck-at fault except those at input lines $J_1, J_2, \ldots, J_y$ result in single bit or unidirectional multibit errors at the output irrespective of the minimization scheme or the type of gates used.

Proof: The circuit can be represented as in Figure 2 where $O_2, O_2, \ldots, O_m$ are the outputs and $I_1, I_2, \ldots, I_n, J_1, J_2, \ldots, J_y$ are the inputs (the dashed box represents the original circuit). The output expressions can be also written as functions of inputs $J_{1a}, J_{1b}, \ldots, J_{ya}, J_{yb}, I_1, I_2, \ldots, I_n$ (the solid box represents the new circuit). For two input cubes, if the distance between their arbitrarily input encoded set is 0, the minimum distance between these cubes is 2 because m-out-of-n codes are used for the second input set. Therefore, if two inputs cubes in the original circuit are x-bidirectional (Def. 3), input x has to be from the arbitrarily input encoded set. Without any loss of generality, suppose that two input cubes are J_x-bidirectional in the original circuit (J_x will be

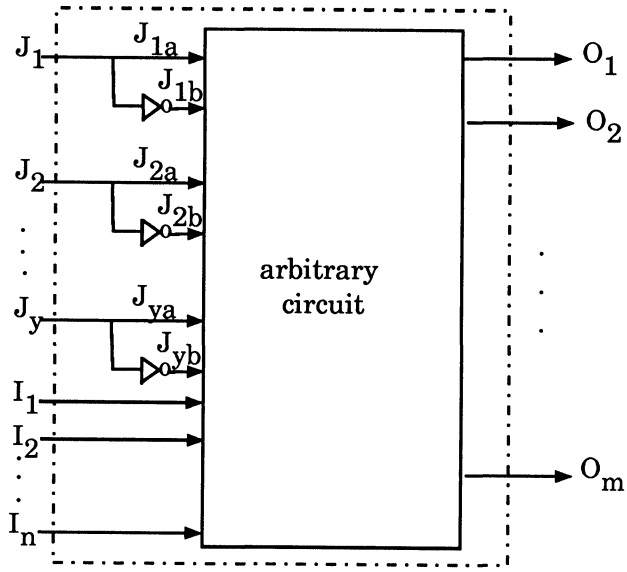


FIGURE 2 Logic circuit realization.

0 in one of the cubes and 1 in the other). In the new circuit, the distance between the same two cubes will be 2 because $J_{xa}J_{xb}$ will 01 in one cube and 10 in the other. In this case, bidirectional input cubes are eliminated, and consequently any single stuck-at fault either internally or at input lines (I_1, I_1, . . . , I_n, J_{1a}, J_{1b}, . . . , J_{ya} and J_{yb}) produces either single bit error or unidirectional multibit error at the outputs (Lemma 4). Q.E.D.

Thus, if the input lines are assumed fault-free, then the above strategy for implementing logic circuits will guarantee that any possible stuck-at fault in the circuit will result in either single bit error or unidirectional multibit error at the output without having any restrictions on the type of gates used or on the minimization procedures.

Lemma 7: A circuit that is both fault-secure and irredundant is also self-testing, and therefore totally self-checking.

Proof: For an input pattern t, let $C(t)$ be the fault-free output of the circuit, and let $C_f(t)$ be the output of the circuit with fault f present. Because the circuit is irredundant, there exists input pattern t such that $C(t)$ does not equal to $C_f(t)$ for every f in the fault set. The circuit is fault secure, so for any such test t, the fact that $C(t)$ does not equal to $C_f(t)$ implies that $C_f(t)$ is not a codeword. Therefore, the circuit is self-testing, and thus totally self-checking. Q.E.D.

TOTALLY SELF-CHECKING FSM

Theorem 1: Consider a state machine with I inputs, S states and N different symbolic outputs.

1. If m-out-of-n code is used for state assignment such that $\binom{n}{m} \geq S$, and
2. m-out-of-n-code for output encoding such that $\binom{n}{m} \geq N$, and
3. Next State Logic (NSL) and Output Logic (OL) are implemented as suggested in Lemma 6, then this state machine will be totally self-checking.

Before we prove the theorem, we show that if the machine is in an invalid state, the first erroneous output is an invalid codeword.

Lemma 8: If a state machine is realized as described in Theorem 1, the OL will be fault-secure for any unidirectional multiple stuck-at faults at the state lines. In other words, in case of an invalid state, the first erroneous output is invalid.

Proof: an invalid state can result from either a fault in the NSL, or a stuck-at fault at the input or output of a memory element. In both cases, the faulty state and correct state are not partially bidirectional since any stuck-at fault in the NSL will result in either single bit error or unidirectional multibit error (Lemma 6). Therefore, an invalid state can be viewed as multiple unidirectional stuck-at faults at the inputs or outputs of the state flip flops (D type flip flop is assumed), which will take the machine from a correct state to an invalid state. By Lemma 5, this multiple stuck-at fault can only result in unidirectional error at the output i.e., either an invalid output codeword or the correct valid codeword will be produced. Therefore, the first erroneous output is an invalid codeword. Q.E.D.

Proof of Theorem 1: We have to prove that machine is fault secure for all single stuck-at fault. For a single fault f, some input patterns may mask the fault while others may propagate its effect to the output. In the proofs of the fault-secure property in this paper, we will only consider the situation when the fault is not masked. We consider three cases:

Case 1: A single stuck at fault in the OL will cause either single bit error or unidirectional multibit error at the output lines (Lemma 6). Since m-out-of-n code is used for output encoding, the output is an invalid codeword.

Case 2: A single stuck-at fault in the NSL will result in only single bit error or unidirectional multibit error at the state lines (Lemma 6). Since m-out-of-n code is used for state assignment, this fault will produce an invalid state. By Lemma 8, the resulting output codeword is invalid.

Case 3: A single stuck at fault at either an input or an output of a memory element will force the circuit

into an invalid state, and thus produce an invalid output (Lemma 8).

Therefore, the machine will never produce a faulty but valid code word in case of a single stuck-at fault; i.e., the circuit is fault secure for any single stuck-at fault.

In order to prove that a fault-secure machine is totally self-checking we have to prove that the circuit is irredundant for all types of faults (Lemma 7). Combinationally redundant faults (CRFs) can be eliminated by designing prime and irredundant (Def. 9) networks for OL and NSL by using techniques proposed in Ref. [12]. Sequential redundant faults (SRFs), however, require restricted logic synthesis methods. For a combinationally irredundant fault to be sequentially redundant, the faulty circuit has to be isomorphic to the fault-free circuit [9]. It was proven in Ref. [9] that if NSL and OL were implemented such that there is even or odd number of inversions between a node (other than the primary inputs) and the primary outputs, then no fault can convert the state machine into a faulty one which is isomorphic with the fault-free state machine. Since Lemma 6 guarantees that only unidirectional errors are present in case of a single fault, then by Lemma 3, the number of inversions from any node to the outputs has to be odd or even, but not both. Thus, the proposed implementation of state machines will eliminate SRFs. Therefore, the circuit is irredundant and hence totally self-checking. Q.E.D.

TOTALLY SELF-CHECKING INTERACTING FSMs

In order to design totally self-checking interacting FSMs, the interacting machines have to be fault-secure and self-testing for any single stuck-at fault. The following theorem outlines the steps for designing self-checking interacting machines:

Theorem 2: Given any type of decomposition with I inputs and O outputs; S1 and S2 are the number of states for submachine M1 and M2 respectively, if

1. m-out-of-n code is used for state assignments such that $\binom{n}{m} \geq S1$, and $\binom{n}{m} \geq S2$, and
2. m-out-of-n code for output encoding such that $\binom{n}{m} \geq N$, and
3. Next State Logic (NSL) for each submachine and the Output Logic (OL) are implemented as suggested in Lemma 6, and
4. the schemes discussed in [10] are used to eliminate redundant faults in the interacting machines then this composite state machine will be totally self-checking for all single stuck-at faults. We will consider subcircuits resulting from each type of decomposition separately.

Parallel Decomposition

In parallel decomposition, Fig. 1a, each submachine operates independently. Since each submachine is realized as in Theorem 1, the composite machine is totally self-checking. Note that the OL is common for both submachines, and if at any time either of the states in M1 and M2 is invalid, the output will be an invalid codeword (Lemma 8).

Cascade Decomposition

We will prove here that if the interacting submachines are designed as proposed in Theorem 2, they will be fault-secure and irredundant for all single stuck-at faults. We will first prove the fault-secure property for the following cases:

Case 1: A single stuck-at fault in the NSL of submachine M1 will create a single bit error or unidirectional multibit error at the state lines (Lemma 6). Since m-out-of-n code is used for state assignments, this fault will place M1 in an invalid state. The state lines of M1 are inputs to M2. Invalid input lines for M2 will place the machine in an invalid state and/or will produce invalid output since the state lines of M1 are m-out-of-n codes (Lemma 8). If the fault produce invalid output, then the circuit is fault-secure. However, if the fault place M2 in an invalid state, the output will be also invalid (Lemma 8).

Case 2: A fault at the input or output of a memory element in M1 will force M1 to an invalid state, and as in case 1, M2 will generate invalid output.

Case 3: A fault in the OL will generate either single bit error or unidirectional multibit error at the output lines (Lemma 6). Therefore, the output is a non codeword.

Case 4: A single stuck-at fault in the NSL of submachine M2 will also create unidirectional errors at the state lines (Lemma 6) which will place M2 in an invalid state, thus creating invalid output (Lemma 7).

Case 5: A fault at the input or output of a memory element in M2 will force M2 to an invalid state, thus creating an invalid output (Lemma 8).

We have proved above that the circuit (submachines implemented using cascade decomposition) is

fault-secure; therefore, to guarantee that the circuit is self-checking, we have to prove that none of the redundancies possible in a cascade decomposition is present (Lemma 7). Four categories of redundancies are present in cascade decomposition which were discussed in Section 2. Type 1 and type 4 which are associated with single submachines M1 and M2 are eliminated because each submachine is self-checking by itself (Theorem 1). Type 2 will not be present in this case because INT1 corresponds to the state lines; therefore, if a fault in M1 propagates to INT1 (state lines), the state lines will generate an invalid codeword (Lemma 8). Type 3 can be eliminated by methods discussed in [10]. CRFs can be eliminated by methods presented in [12]. Therefore, a state machine using cascade decomposition is fault-secure and irredundant, hence totally self-checking. Q.E.D.

Arbitrary Decomposition

In arbitrary decomposition, each submachine has information about the other submachine through its state lines. The state lines of both submachines and the primary inputs feed the OL. By Lemma 8, an invalid state at either of the submachines will cause invalid output. Similar to the proofs of Theorems 1 and 2, a fault in the NSL of either submachine will cause either single bit error or unidirectional multibit error. Therefore, this fault will place the corresponding submachine in an invalid state. By Lemma 8, an invalid state will produce an invalid output. A fault at the input or output of a memory element will result in an invalid state thus resulting in an invalid output. Finally, a fault in the OL will also result in either a single bit error or unidirectional multibit error at the output. Thus, the decomposed circuit is fault-secure.

To prove that the circuit is irredundant, all redundant faults associated with arbitrary decomposition have to be eliminated. CRFs can be eliminated from NSL of each submachine and from OL, by schemes presented in [12]. Six categories of redundancies are present in arbitrary decomposition as described in Section 2. Type 1, type 3, type 4 and type 6 which are associated with submachines M1 and M2 are eliminated since each submachine by itself has no redundant faults (Theorem 1). Type 2 and type 5 can be eliminated by methods discussed in [10]. Therefore, the resulting cascade decomposition is fault-secure and irredundant, hence totally self-checking. Q.E.D.

RESULTS

We applied the proposed synthesis techniques to a set of interacting state machines formed by connecting different MCNC benchmark state machines in cascade and in arbitrary decomposition. The NSL and OL of the decomposed submachines are implemented using multi-level logic (factored form). The number of literals obtained from such realization has been compared to the results obtained by applying techniques that target two-level (NOVA [14]) and multi-level (MUSTANG [13]) implementations; as far as we are aware, no other state assignment and output encoding schemes are available that can be directly applied to interacting FSMs. The number of literals in multi-level realization is approximately twice the number of transistors used in CMOS implementation because each literal will be an input to two transistors (one n-type and one p-type). Therefore, the results reported here (the number of literals in factored form) give a fair estimate of the area requirement for VLSI implementation.

Table I shows the statistics of 10 interacting state machines which are constructed by connecting different MCNC benchmark circuits. Six of these machines are connected in cascade decomposition and the re-

TABLE I
Statistics of the benchmark examples
#inp is the number of primary inputs to the interacting machine
#out is the number of primary outputs
#diff_o is the number of different output patterns
#state is the number of states

Examples	Type of decomposition	#inp	#out	#diff_o	#state M1	#state M2
seq1	cascade(bbtas,bbara)	2	2	3	6	10
seq2	cascade(dk15,bbtas)	3	2	4	4	6
seq3	cascade(dk27,beecount)	1	4	4	7	7
seq4	cascade(tbk,cse)	6	7	14	32	16
seq5	cascade(ex6,dk14)	5	5	12	8	7
seq6	cascade(train11,ex4)	2	9	11	11	14
seq7	arbitrary(bbara,dk14)	4	7	15	10	7
seq8	arbitrary(opus,dk15)	5	11	19	10	4
seq9	arbitrary(mark1,mc)	5	21	17	15	4
seq10	arbitrary(ex6,train11)	5	9	14	8	11

maining four are connected in arbitrary decomposition. Cascade(M1,M2) means that the two machines, M1 and M2, are connected in cascade decomposition where M1 is the driving machine and M2 is the driven machine. The state lines of M1 are now part of the inputs to M2, and the outputs of M2 are considered as primary outputs. For example, **seq1** is an interacting FSM constructed by cascading bbtas and bbara, where bbtas is the driving machine and bbara is the driven machine. Arbitrary(M1,M2) means that the two machines are connected in arbitrary decomposition. The outputs of the interacting machine is the concatenation of the primary outputs of the two submachines. For example, **seq7** has been obtained by connecting bbara and dk14, and the outputs of **seq7** are the concatenation of the bbara outputs and dk14 outputs.

Table II gives the number of literals in factored form for each submachine, and for the composite machine obtained by applying our technique, MUSTANG [13] and NOVA [14], where the sum represents the number of literals for the composite machine. The TSC column is produced by applying Theorem 2 and by performing unconstrained boolean minimization using MIS [15]. The results in Table II do not include the area of the latches. Table III gives the number of literals when the latches are included as well; each latch is considered to have 3 literals [16].

TABLE II
Literal Counts Without Counting Latches

	TSC			MUSTANG			NOVA		
Examples	M1	M2	sum	M1	M2	sum	M1	M2	sum
seq1	25	66	91	24	86	110	21	95	116
seq2	32	36	68	28	34	62	28	30	58
seq3	28	15	43	28	18	46	31	20	51
seq4	287	222	509	276	184	460	247	189	436
seq5	64	112	176	61	113	174	67	103	170
seq6	19	39	58	24	71	95	25	82	107
seq7	64	99	163	83	100	183	92	91	183
seq8	61	65	126	82	58	140	88	58	146
seq9	45	21	66	104	24	128	103	26	129
seq10	86	20	106	82	23	101	90	24	114
total			1406			1499			1510

TABLE III
Literal Counts Including Latches

	TSC			MUSTANG			NOVA		
Examples	M1	M2	sum	M1	M2	sum	M1	M2	sum
seq1	43	87	130	33	95	128	30	104	134
seq2	44	54	98	32	43	75	32	39	71
seq3	49	24	73	37	22	59	40	24	64
seq4	335	249	584	288	196	484	259	201	460
seq5	88	133	221	70	122	192	76	112	188
seq6	31	54	85	28	80	108	29	91	120
seq7	85	120	205	92	109	201	101	100	201
seq8	73	77	150	86	64	150	92	64	156
seq9	78	27	105	116	27	143	115	29	144
seq10	110	32	142	91	22	123	99	33	132
total			1793			1663			1670

CONCLUSION

This paper presented a technique for making interacting FSMs totally self-checking for single stuck-at faults. These techniques use m-out-of-n code for state assignments and for output encoding. The NSL of each submachine and the OL are designed such that a fault can only cause unidirectional errors at the output. The overhead of these techniques when applied to benchmarks circuits are considerably low, less than 8%. MUSTANG and NOVA try to minimize the 'cost' of a state machine without considering the testability problem. If scan-based techniques were used, there will be an area overhead because of scan-in circuitry, scan-out circuitry and scan flip flops.

Acknowledgment

This work was supported by NASA-Johnson Space Center under grant NAG9-504. The authors would like to thank one of the reviewers for providing the proof of Lemma 7.

References

[1] V.D. Agrawal, S.K. Jain and D.M. Singer, "Automation in design for testability." *Proceeding of the Custom Integrated Circuit Conference,* pp. 159–163, May 1984.

[2] S. Devadas, H-K.T. Ma, A.R. Newton, and A. Sangiovanni-Vincentelli, "A synthesis and optimization procedure for fully and easily testable state machines," *IEEE Trans. Computer-Aided Design,* Vol. 8, pp. 1100–1107, Oct. 1989.

[3] Y. Savaria, N.C. Rumib, J.F. Hayes and V.K. Agrawal, "Soft-error filtering: A solution to the reliability problem of future VLSI digital circuits," *IEEE Proc.,* Vol. 74, pp. 669–683, May 1986.

[4] M. Diaz, "Design of totally self-checking and fail safe state machines," *Proc. Int. Symp. Fault-Tolerant Comput.,* Urbana, IL, pp. 9–24, June 1974.

[5] F. Ozguner, "Design of totally-self-checking asynchronous state machines," *Proc. Symp. Fault-Tolerant Comput.,* pp. 124–129, June 1977.

[6] S. Devadas, and A.R. Newton, "Decomposition and Factorization of sequential finite state machines," *IEEE Trans. Computer-Aided Design,* Vol. 8, pp. 1206–1217, Nov. 1989.

[7] J. Hartmanis, "Symbolic analysis of a decomposition of information processing," *Infarm. Contr.,* pp. 154–178, June 1960.

[8] P. Ashar, S. Devadas, and A. R. Newton, "Optimum and heuristic algorithms for an approach to finite state machine decomposition," *IEEE Trans. Computer-Aided Design,* Vol. 10, pp. 296–310, March 1991.

[9] S. Devadas, H.T. Ma, A.R. Newton, and A. Sangiovanni-Vincentelli, "Irredundant state machines via optimal logic

synthesis," *IEEE Trans. Computer-Aided Design,* Vol. 9, pp. 8–17, Jan. 1990.

[10] P. Ashar, S. Devadas, and A.R. Newton, "Irredundant interacting state machines via optimal logic synthesis," *IEEE Trans. Computer-Aided Design,* Vol. 10, pp. 311–325, March 1991.

[11] F. Busaba and P.K. Lala, "Input and output encoding techniques for on-line error detection in combinational logic circuits," to be presented at the 11th IEEE Test Symposium, Atlantic City, NJ, April 1993.

[12] K. Bartlett, R.K. Brayton, G.D. Hachtel, R.M. Jacoby, C.R. Morrison, R.L. Rudell, A. Sangiovanni-Vincentelli, and A.R. Wang, "Multilevel logic minimization using implicit don't cares," *IEEE Trans. Computer-Aided Design,* Vol. 7, pp. 723–740, June 1988.

[13] S. Devadas, H-K.T. Ma, A.R. Newton, and A. Sangiovanni-Vincentelli, "MUSTANG: State assignment of finite state machines targeting multilevel logic implementations," *IEEE Trans. on Computer-Aided Design,* Vol. 7, pp. 1290–1300, Dec. 1988.

[14] T. Villa and A. Sangiovanni-Vincentelli, "NOVA: State assignment of finite state machines for optimal two-level logic implementations," *Proc. Design Automation Conf.,* pp. 327–332, June 1989.

[15] R.K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A.R. Wang, "MIS: A multiple-level logic optimization program," *IEEE Trans. on Computer-Aided Design,* Vol. 7, pp. 1062–1081, Nov. 1987.

[16] X. Du, G. Hachtel, B. Lin and A.R. Newton, "MUSE: A multilevel symbolic encoding algorithm for state assignment," *IEEE Trans. on Computer-Aided Design,* Vol. 10, pp. 28–38, January 1991.

Biographies

FADI BUSABA is an Assistant Professor at North Carolina A&T State University. His research interests include fault-tolerant computing, self-checking and synthesis for testability. He holds a BE from American University of Beirut, an MS in Electrical Engineering from North Carolina Agricultural and Technical State University, and a PhD from North Carolina State University at Raleigh. He received MCNC Fellowship and is an IEEE member.

PARAG K. LALA is a Professor in the Department of Electrical Engineering at North Carolina Agricultural and Technical State University. His research interests include test generation/testability, fault-tolerant computing, digital system design, and self-checking design. He authored *Fault-Tolerant and Fault-Testable Hardware Design* and *Digital System Design Using PLDs,* both published by Prentice-Hall, Inc. He holds an MSc in Electrical Engineering from King's College, London, and a PhD from the City University of London.