
JOURNAL OF ELECTRONIC TESTING: Theory and Applications 19, 537–548, 2003

On-Line Monitor Design of Finite-State Machines

FENG GAO AND JOHN P. HAYES
Advanced Computer Architecture Lab., University of Michigan, Ann Arbor, MI 48109, USA

Received July 21, 2002; Revised November 24, 2002

Editors: C. Metra and M. Sonza Reorda

Abstract. On-line monitoring is a useful technique for ensuring system reliability. By continuously supervising the system's operation, a wide range of problems, such as physical defects, transient faults and design errors, can be detected. A monitor M^*'s behavior can be viewed as an abstraction of the target system M's behavior, and can be represented by a homomorphic mapping from M to M^*. We present a systematic procedure to select homomorphisms for monitor design and measure their costs based on a behavioral fault model. Analysis of the method shows that monitors with very few states and low area can provide high fault coverage. Experimental results are presented which quantify the basic trade-off between area overhead and fault coverage. Simulation results under the industry-standard single stuck-at fault model are also reported.

Keywords: on-line monitoring, homomorphism, finite-state machine

1. Introduction

Increasingly, integrated circuits (ICs) are being used in safety-critical applications, from anti-lock braking in automobiles, to fly-by-wire aircraft, to prosthetic systems in the human body. At the same time, ICs are becoming more complex and compact, resulting in their increased susceptibility to transient or intermittent faults. Such faults affect the behavior of the circuits from time to time and must be detected quickly, suggesting a need for efficient on-line error monitoring. An on-line monitor continuously supervises the operation of a circuit and reports illegal behavior.

On-line monitoring strategies can be applied at various levels of abstraction, ranging from the gate and register-transfer levels to the system level. Run-time errors can be caught if we can check whether the current outputs are valid. In such approaches, the outputs of a circuit are usually encoded with an error-detecting code, and a monitor detects the occurrence of non-code outputs. A straightforward approach [18] is to append check bits to the normal output bits. The resulting designs consist of functional logic, a check-bit generator, and a checker. The functional logic generates the normal outputs, the check-bit generator generates the check bits, and the checker determines if the outputs are valid codewords. Various encoding methods, for instance, group parity codes [20] and Berger codes [14], can be used to encode the outputs. A basic disadvantage of these methods is that a single fault inside the circuit may change the outputs to other codewords and so be undetectable.

We can also embed self-testing structures in the target design. Tests for functional modules can then be applied during the cycles when the modules are inactive. If the module functions are simple and the free cycles for each functional module can be pre-determined, we can schedule the test and normal operations at the same time without clearly defining a boundary between these two modes [8]. On the other hand, if the cycles during which a module is inactive are irregular, extra control signals may be needed to switch between the testing and normal modes [15]. The main disadvantage of this type of self-testing is that it is not good at detecting transient faults.

Signature analysis [12] is another widely studied approach to on-line monitoring of programmable systems. A co-processor, often called a watchdog or diagnostic processor, whose duties include signature computation and checking, is usually employed. Error detection by means of a watchdog is a two-phase process. In the first phase, signatures are computed and assigned to basic blocks of the target program, where a basic block is a set of instructions with no jumps allowed from or into these instructions. During the second phase, the monitor computes run-time signatures and compares them with the precomputed signatures received from the target system. Differences between these two sets of signatures reveal permanent or transient errors. Besides their limited ability to detect data-related errors, such methods also cause performance degradation by increasing the number of instructions in a program.

In this paper, the system to be monitored M is modeled as a finite-state machine (FSM), for which a simple behavior fault model is defined. A monitor M^* is then abstracted systematically from the behavioral description of M using homomorphisms selected under the guidance of appropriate cost measures. The effects of the number of states in the monitor on its area overhead and fault coverage are analyzed. It is shown that one-state monitors, i.e. combinational monitors, have small area overhead. Moreover, for FSMs with high output/state ratios, such monitors also have very high fault coverage under our behavioral fault model. Simulations under the standard single stuck-at fault model are also performed.

The remainder of this paper is organized as follows. After the introduction of some notation and our system model in the next section, we propose an algorithm for FSM behavior abstraction based on homomorphisms in Section 3. Section 4 analyzes how the area overhead and fault coverage of the monitors vary with respect to the number of states in the monitors. Experimental results are presented in Section 5, while some conclusions are discussed in Section 6.

2. System Model

A *finite-state machine* (*FSM*) M is defined as a 6-tuple $\{I, S, \delta, O, \lambda, S_0\}$, where I is the set of inputs, S is the set of states, δ: $I \times S \rightarrow 2^S$ is the state transition function, S is the set of outputs, λ: $I \times S \rightarrow 2^O$ is the output function, and $S_0 \subseteq S$ is the set of initial states [11]. An FSM is deterministic if δ and λ are deterministic, that is, no input can map the machine to two or more outputs or next states at the same time. An FSM can be described using a state transition graph (STG). The STG of M is denoted by $G_M(V_M, E_M)$, where V_M is the set of nodes, representing the state set of M and $E_M = \{\langle u, v\rangle, u, v \in V_M\}$ is the set of edges, representing the transition set of M. Each edge $\langle u, v\rangle$ is labeled with a set of input-output pairs, each of which denotes an input that triggers the state transition and the corresponding output. The number of edges that start from or end at a state $u \in V_M$ is called the degree of u. The maximum degree of all the states is called the degree of the STG. For convenience, the degree of the STG is also referred to as the degree of the corresponding FSM. Fig. 1(a) shows a state transition graph representing a seven-state FSM. The degrees of state A, D, and F are two, while that of B is four, which is the maximum degree of all these states. Therefore, the degree of the STG is four.

Let $M = \{I, S, \delta, O, \lambda, S_0\}$ be the specification of an FSM, and $M' = \{I, S, \delta', O, \lambda', S_0\}$ be its implementation. M' has a *next-state fault* on receiving input i in state s if $\delta(s, i) \neq \delta'(s, i)$, where $s \in S$ and $i \in I$.

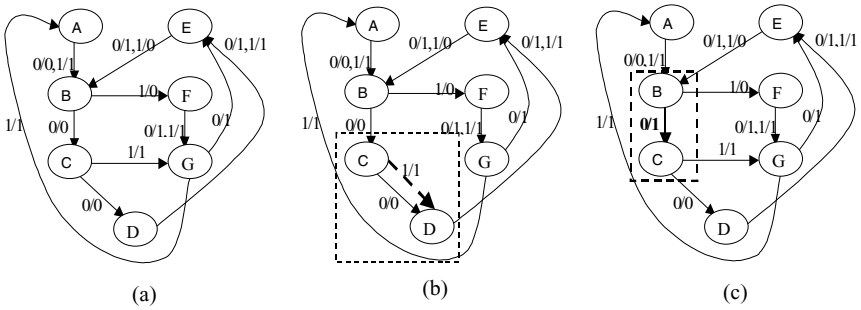


Fig. 1. (a) An example FSM M, (b) a next-state fault of M, and (c) an output fault of M.

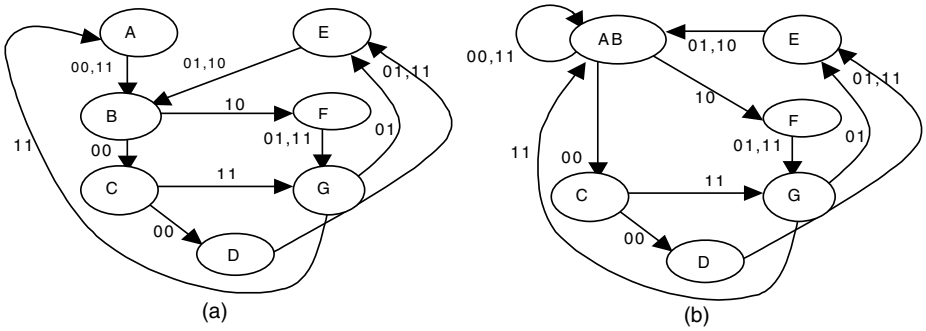


Fig. 2. (a) Base monitor M_0^*; (b) monitor M_1^* formed by applying a state homomorphism to M_0^*.

M' has an *output fault* on receiving input i in state s if $\lambda(s, i) \neq \lambda'(s, i)$, where $s \in S$ and $i \in I$. We refer to next-state faults and output faults as *behavior faults*. Two versions of M with behavior faults are given in Fig. 1(b) and (c). The dotted rectangles highlight a next-state fault and an output fault of M. For simplicity, we assume that an FSM has at most one next-state or one output fault at any time. We call this the *single behavior fault* (SBF) model.

With the concepts defined above, we now introduce our overall system model. Suppose $M = \{I, S, \delta, O, \lambda, S_0\}$ is the FSM to be monitored. We first define a *base monitor* $M_0^* = \{I \times O, S, \delta_0^*, \{0, 1\}, \lambda_0^*, S_0\}$, which is able to detect all the SBFs of M, where (1) $\delta_0^*(ab, s) = \delta(a, s)$, for $a \in I, b \in O$, and (2) $s \in S$ and $\lambda_0^*(ab, s) = 0$ if $\lambda(a, s) = b$; otherwise, the output is 1 indicating an SBF. For example, the base monitor of the FSM M in Fig. 1(a) is shown in Fig. 2(a). For brevity, only those transitions with output 0 are shown, while those with output 1 are omitted. Therefore, the indicated transitions correspond to all valid transitions in M.

We next perform behavior abstraction on the base monitor using homomorphisms, which provides the theoretical foundation for our approach. A many-to-one mapping $H = (\phi_I, \phi_S, \phi_O)$ is an *FSM homomorphism* from $M_1 = \{I_1, S_1, \delta_1, O_1, \lambda_1, S_{01}\}$ to $M_2 = \{I_2, S_2, \delta_2, O_2, \lambda_2, S_{02}\}$ if and only if $\phi_I(i_1) \in I_2$ for $i_1 \in I_1, \phi_S(s_1) \in S_2$ for $s_1 \in S_1, \phi_O(o_1) \in O_2$ for $o_1 \in O_1, \phi_S(\delta_1(i_1, s_1)) = \delta_2(\phi_I(i_1), \phi_S(s_1))$ and $\phi_O(\lambda_1(i_1, s_1) = \lambda_2(\phi_I(i_1), \phi_S(s_1))$ for $i_1 \in I_1, o_1 \in O_1$. M_2 is the homomorphic image of M_1. Fig. 2(b) shows a homomorphic image of the FSM M_0^* in Fig. 2(a). This particular homomorphism maps states A and B of M_0^* into one state in its image, while all other states, inputs and outputs are mapped to themselves. Intuitively, a homomorphism reduces the number of states and preserves the mappings of the next-state and output functions; hence it is a precise form of behavior abstraction.

The overall monitoring scheme is illustrated in Fig. 3(a). The monitor M^* operates in lockstep with M, it can mimic the state transitions of M and report SBFs with just a one-cycle delay. If the monitor is the base monitor M_0^*, all possible SBFs can be detected. If lower fault coverage is allowed, we can perform homomorphisms on M_0^* to obtain a smaller, and therefore more economical, monitor M^*. As shown in Fig. 3(b), M^* is composed of two parts: interface logic which implements the homomorphisms from M to M^*, and a compact FSM M_c which approximates the state transitions of M.

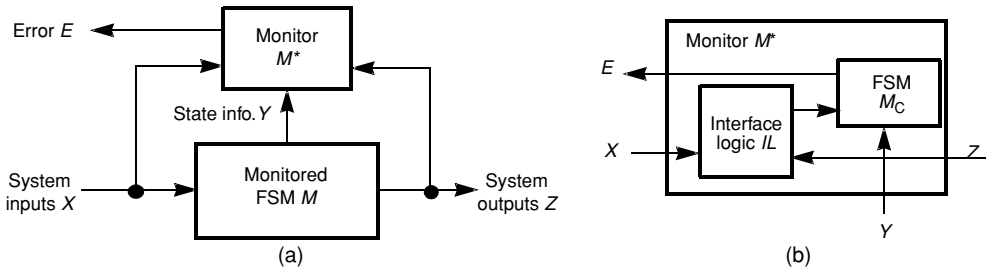


Fig. 3. On-line monitoring system: (a) overall structure; (b) monitor design.

3. Abstraction Algorithm

To reduce the monitor area, we apply homomorphisms to M_0^* in two stages called the state and input stages. In the state homomorphism stage, we gradually reduce the number of states in the monitor. The fault coverage hence decreases due to the limited knowledge available concerning target system states. While in the input homomorphism stage, we only perform a simple transformation, which does not hurt the fault coverage. We do not consider output homomorphisms, since the base monitor has only a single output. Note that the inputs of M_0^* incorporate both the inputs and outputs of the original machine M.

The state homomorphism stage involves constructing a series of homomorphisms of the form: $M_0^* \rightarrow M_1^* \rightarrow M_2^* \rightarrow \cdots \rightarrow M_n^* = M^*$. Each homomorphism H: $M_i^* \rightarrow M_{i+1}^* (0 < i < n)$ maps two states of M_i^* to one state of M_{i+1}^*. Such mappings are referred to as *simple state homomorphisms*. While H reduces the number of states in the monitors, the number of detectable SBFs also decreases because of the state merging. Homomorphisms that lead to as few undetectable SBFs as possible are desirable, because our goal is to obtain monitors with small area, capable of detecting as many SBFs as possible. Furthermore, the number of available simple state homomorphisms is proportional to the square of the number of states in the monitor. Therefore, to guide the selection of simple state homomorphisms, we define a homomorphism cost function $C(H)$ that measures the homomorphism's impact on fault coverage.

For a homomorphism H: $M_i^* \rightarrow M_{i+1}^*$, $C(H)$ *is* the number of SBFs that become undetectable if we replace M_i^* with M_{i+1}^* as the monitor. In fact, those faults can readily be enumerated. To facilitate the calculation, we classify them into three types, which are defined next via an example, along with a calculation method. We denote the inputs that trigger transitions from state s_1 to state s_2 by *tr_input*(s_1, s_2), and the states in M_0^* that are mapped to s by *pre-image*(s). Let $|T|$ be the cardinality of set T. Using the base monitor and its homomorphic image in Fig. 2 as an example, the base monitor M_1^* in Fig. 2(b) is obtained by applying a simple state homomorphism H to M_0^*, which maps states A and B of M_0^* into one state AB of M_1^*. The three SBF types introduced by H are as follows.

- *Next-state-change faults*. If the next state of a transition changes erroneously from B to A, this fault becomes undetectable when we replace M_0^* with M_1^* as the monitor. For example, the transition from E to A on receiving input 01 or 10 is detectable in M_0^*. This fault is, however, undetectable using M_1^*. The transition from G to B on input 11 also belongs to this type. The associated cost $C_1(H)$ is defined as:

$$C_1(H) = |\textit{pre-image}(E)| * |\textit{pre-image}(A)| * 2 + |\textit{pre-image}(G)| * |\textit{pre-image}(B)| * 1$$

  The constant coefficients 2 and 1 are the cardinalities of *tr_input*$(E, B) = \{01, 10\}$ and *tr_input*$(G, A) = \{11\}$. In general, if states s_1 and s_2 are to be merged and some inputs trigger the transition from a state s to s_1 or s_2, each input corresponds to a next-state-change fault that will escape detection by the image monitor. The general formula to compute the cost $C_1(H)$ is given in the first row of Fig. 4.

Type	Description	Cost function
Next-state-change fault	Next-state of a transition changes to another state having the same image	$\sum_{tr\_input(s,\, s_1)}^{s \in S} \lvert pre\text{-}image(s)\rvert \times \lvert pre\text{-}image(s_2)\rvert \times \lvert tr\_input(s,s_1) - tr\_input(s,s_2)\rvert + \sum_{tr\_input(s,\, s_2)}^{s \in S} \lvert pre\text{-}image(s)\rvert \times \lvert pre\text{-}image(s_1)\rvert \times \lvert tr\_input(s,s_2) - tr\_input(s,s_1)\rvert$
Current-state-change fault	Current state of a transition changes to another state having the same image	$\sum_{tr\_input(s,\, s_2) \neq \emptyset}^{s \in S} \lvert pre\text{-}image(s_1)\rvert \times \lvert pre\text{-}image(s)\rvert \times \lvert tr\_input(s,s_2) - tr\_input(s,s_1)\rvert + \sum_{tr\_input(s,\, s_1) \neq \emptyset}^{s \in S} \lvert pre\text{-}image(s_2)\rvert \times \lvert pre\text{-}image(s)\rvert \times \lvert tr\_input(s,s_1) - tr\_input(s,s_2)\rvert$
Current-next-state-change fault	Current state and next-state interchange	$\sum_{i,j \in \{1,2\}} \lvert tr\_input(s_i, s_j) - tr\_input(s_j, s_i)\rvert \times \lvert pre\text{-}image(s_1)\rvert \times \lvert pre\text{-}image(s_2)\rvert$

Fig. 4. The three types of SBFs and the corresponding homomorphism costs.

- *Current-state-change faults*. State transitions with current state A or B can also become undetectable if their destinations are different, forming the second fault type. For instance, next-state fault $A \xrightarrow{00} C$ is detected by M_0^*. After mapping A and B to one state, M_1^* is not able to determine if the transition is initiated from A or B, causing undetectable next-state fault $A \xrightarrow{00} C$. Erroneous transitions falling into this type also include $A \xrightarrow{00} F$ and $B \xrightarrow{00,11} B$. Similarly, we can construct a cost function $C_2(H)$ for this fault type as follows:

$$\begin{aligned} C_2(H) = {} & |\textit{pre-image}(A)| * |\textit{pre-image}(C)| \\ & + |\textit{pre-image}(A)| * |\textit{pre-image}(F)| \\ & + |\textit{pre-image}(B)| * |\textit{pre-image}(B)| * 2 \end{aligned}$$

  The general formula for $C_2(H)$ is shown in the second row of Fig. 4. We can read the formula as follows. If states s_1 and s_2 are to be merged and some inputs triggers the transition from s_1 or s_2 to a state s, each of the inputs introduces to a current-state-change fault that will escape detection by the image monitor.
- *Current-next-state-exchange faults*. The existence of a state transition from A to B causes undetectable next-state faults, which change both the current and next states of the transition. For example, erroneous transition $B \xrightarrow{00,11} A$, which has different current and next states from valid transition $A \xrightarrow{00,11} B$, becomes undetectable in M_1^*. The cost contributed by this type of fault is

$$C_3(H) = |\textit{pre-image}(B)| * |\textit{pre-image}(A)| * 2$$

  In general, suppose that states s_1 and s_2 are to be merged into a single state s, and input i causes a transition from s_1 to s_2, the faulty transition from s_2 to s_1 on input i corresponds to a current-next-state-exchange fault that cannot be detected by the image monitor. The general formula of $C_3(H)$ is shown in the third row of Fig. 4.

The total cost $C(H)$ of a simple state homomorphism H that maps state s_1 and s_2 into one image is the sum $C_1(H) + C_2(H) + C_3(H)$ of the costs specified above. The formulas in Fig. 4 indicate the complexity of computing $C(H)$. Given a state pair, the only computation is a group of set differences, whose complexity is $O(d)$, where d is the degree of the monitoring FSM. There are in total $|S|^2$ state pairs, resulting in a final complexity of $O(d|S|^2)$.

We turn next to the selection of the homomorphisms used to construct M^*. In general, a homomorphism H can introduce non-determinism. For example, M_1^* in Fig. 2(b) is non-deterministic because applying input 00 to state AB triggers transitions to itself and to C. Such non-determinism is easily eliminated, but it increases the number of states. Thus we want to select homomorphisms that have both low cost C and deterministic images. To aid this heuristic selection process, we introduce the state dependency graph. A *state dependency graph* (*SDG*) $G = (V, E)$ of an FSM $M^* = \{I, S, \delta, O, \lambda, S_0\}$ is a directed graph, where each node n_i is labeled by two states $(s_{i,1}, s_{i,2})$ representing a homomorphism $\Phi^i = (\phi_{i,I}, \phi_{i,S}, \phi_{i,O})$, where $\phi_{i,S}(s_{i,1}) = \phi_{i,S}(s_{i,2}) = s$, and other states, inputs and outputs are mapped to themselves. There is an edge (n_i, n_j) from n_i to n_j in G if there is an input t such that $\lambda(s_{i,1}, t) = s_{j,1}$ and $\lambda(s_{i,2}, t) = s_{j,2}$, or else $\lambda(s_{i,1}, t) = s_{j,2}$ and $\lambda(s_{i,2}, t) = s_{j,1}$. Intuitively, the edge (n_i, n_j) indicates that if we apply homomorphism Φ^i, we must also apply homomorphism Φ^j to eliminate non-determinism due to Φ^i. In the FSM in Fig. 2(a), the next states of states A and B on input 00 are B and C, respectively. Therefore, there is an edge from node (A, B) to (B, C), as shown in Fig. 5. For a state pair $(s_{i,1}, s_{i,2})$, we have to compare the inputs for every state pair that $s_{i,1}$ and $s_{i,2}$ may transit to, resulting in d^2 comparisons in the worst case. The worst-case complexity of constructing the SDG is hence $O(d^2|S|^2)$ since we have to make the comparison for every state pair. The SDG is similar to the implication graph used for FSM minimization [10]. The major difference is that in an SDG, we only consider whether two states have the same next states on all inputs, while in an

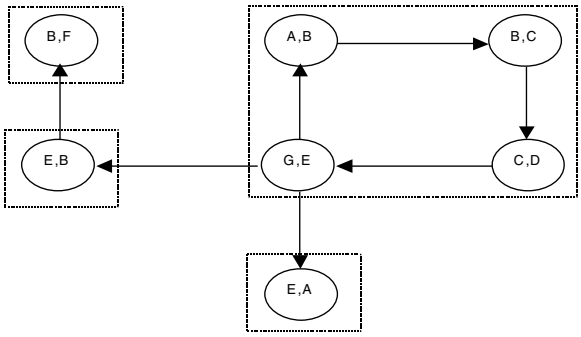


Fig. 5. Part of the SDG for M_0^* in Fig. 2(a).

```
Heuristic_SH (FSM M0, Cost C){
  Construct base monitor M0* ;
  Compute the cost of each pair of states;
  Construct SDG for M0* ;
  while (C > 0) {
    Find valid homomorphism Φ with minimum cost T;
    if (T > C) break;
    C = C - T;
    M(i+1) = Φ( Mi );
    i = i + 1;
    Construct SDG M(i+1);
  }
```

Fig. 6. Heuristic algorithm for state homomorphism selection.

implication graph, we take both next states and outputs into consideration.

A strongly connected component (SCC) in a directed graph is a set of nodes in which there are paths between any two nodes. The dotted lines in Fig. 5 indicate some SCCs in the SDG. SCCs that do not depend on other SCCs represent sets of mappings that should be applied together to avoid non-deterministic images. The homomorphism cost of an SCC is the sum of the costs of the simple state homomorphisms represented by the nodes in the SCC. We apply homomorphisms represented by nodes in SCCs with the lowest cost during the abstraction process until no further homomorphisms can be applied without violating fault coverage constraints. The algorithm is shown in Fig. 6. The complexity of the algorithm is dominated by SDG construction, which occurs in every iteration. Since each iteration reduces the number of states by at least one, the maximum number of iterations is $|S|$. Therefore, the worst case complexity of the overall algorithm is $O(d^2|S|^3)$.

After the state homomorphism stage, homomorphisms that map the inputs to a smaller input space can be applied. Fig. 7 shows a monitor M^* obtained after the state homomorphism stage on base monitor M_0^* in Fig. 3 has been completed. Note that this monitor's

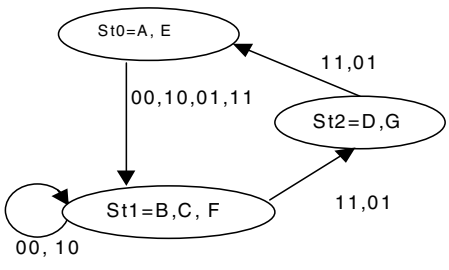


Fig. 7. Sample monitor M^*.

behavior is incompletely specified. For instance, the transitions of state *st*2 on inputs 00 and 01 are not defined. The reason for this is that valid inputs of the target machine M are usually only a non-trivial subset of the whole input space. We can think of this as saying there is a dummy state serving as the next state of the unspecified transitions, and the output of those transitions is 1, indicating an error in M.

The input space is naturally partitioned into two groups by the state transitions initiated from state *st*1 by inputs 00, 10 and inputs 11, 01. Similarly, state *st*2 also divides the input space into two groups based on its transitions. The inputs that can be mapped to the same image are those always in the same group for all partitions. For example, inputs 01 and 11 of M^* are in the same group because they always trigger transitions with the same next states and outputs for all current states. On the other hand, inputs 00 and 01 cannot be mapped to the same image input because they result in the same next state when the current state is *st*0, but result in different next states when the current state is *st*1. Therefore, instead of considering inputs pair by pair, we form a cross product of all the partitions made by the states. All inputs in the same partition of the cross products can then be mapped to the same image.

In the above algorithm, state abstraction starts directly from the base monitor. One can ask whether it is beneficial to extend the two-stage approach (state homomorphism selection followed by input homomorphism selection) to a three-stage one, where we first apply some input homomorphisms. For example, given the base monitor $M_0^* = \{I \times O, S, \delta_0^*, \{0, 1\}, \lambda_0^*, S_0\}$ of the target system $M = \{I, S, \delta, O, \lambda, S_0\}$, we can first map the input space of M_0^* to a smaller one. We can use the input homomorphism $H: I \times O \rightarrow I \times O'$, where O' is obtained by a group parity checking code. That is, we divide the signals of O into groups, compute an even (or odd) parity check bit for each group, and form O' with these parity check bits. Thus, if 1101 is an output of M, and is viewed as two groups 11 and 01, then parity bits for these two groups are 0 and 1, respectively. Hence 01 is a parity code for output 1101. This process is analogous to first generating an FSM whose outputs are the group parity check bits of M's outputs, and then constructing a base monitor M_0' for the new FSM. The previous two-stage approach can then be applied to M_0'. Note that if we construct O' by generating one even parity check bit for each bit of M's outputs, we obtain $M_0' = M_0^*$. Therefore, the two-stage approach can be viewed as a special case of the

three-stage one, where an identity input homomorphism is applied. The area of M_0' is almost certainly smaller than that of M_0^* if the input homomorphism is nontrivial, because it has fewer input signals. Starting with a smaller monitor for state abstraction, we may obtain a smaller final monitor that meets some bounds on fault coverage. However, the number of available homomorphisms with deterministic images may also decrease, leaving the potential benefits of this technique an open question. Our experiments will show that it is possible to obtain monitors with high fault coverage and low area overhead following this approach.

4. Area/Fault Coverage Trade-Offs

The monitor design approaches proposed above trades off fault coverage for area overhead. The area reduction primarily results from the reduction of the number of states in the monitor. We now investigate how the number of states in monitor M^* affects its area overhead and fault coverage. In Section 5, we will verify this analysis experimentally.

As shown in Fig. 3(b), M^* consists of the interface logic *IL* and the compressed finite-state machine M_c. Let n and l be the number of input and output bits of *IL*, respectively, and S the number of states in M_c. We use $L = 2^l$ to denote the number of outputs. As discussed in Section 3, the transitions initiated from one state divide the input space into disjoint sets. Since a state may change to any other state, the state transitions initiated from a state divide the input space into $(S + 1)/2$ sets. The final partition of the input space has $L = ((S + 1)/2)^S$ sets in the worst case. In fact, the compressed finite-state machine M_c typically has only a few states and inputs. Therefore, the interface logic takes the major portion of the monitor area.

Let F be an arbitrary multiple-output Boolean function representing the interface logic, and having n input bits and l output bits. The area of *IL* is approximated by the number of literals in the following equations [5].

$$F = a_1 p_1 + a_2 p_2 + \cdots + a_t p_t \qquad (1)$$

$$p_1 = \tilde{x}_{11}\tilde{x}_{12}\ldots\tilde{x}_{1u_1} \qquad (2)$$

$$p_2 = \tilde{x}_{21}\tilde{x}_{22}\ldots\tilde{x}_{2u_2} \qquad (3)$$

$$\ldots\ldots$$

$$p_t = \tilde{x}_{t1}\tilde{x}_{t2}\ldots\tilde{x}_{tu_t} \qquad (t+1)$$

Here, $\tilde{x}_i$ is a literal representing an input variable or its complement. p_i is prime implicant of the multiple-output function F and $a_i \in A = \{i \mid 0 \le i < L = 2^l\}$.

The probability that $\tilde{x}_{i1}\tilde{x}_{i2}\ldots\tilde{x}_{i(n-k)}$ forms a prime implicant is $L^{-2^k}(1 - L^{-2^k})^{n-k}$, where a k-cube is assigned a fixed value while all its $n-k$ neighboring k-cubes are not. There are altogether $\binom{n}{k}2^{n-k}$ possible k-cubes and L possible assignments to a k-cube. Consequently, the average number of k-cube prime implicants in F is

$$\binom{n}{k}2^{n-k}L^{-2^k}\left(1 - L^{-2^k}\right)^{n-k}(L-1)$$

The above expression contains $L-1$ rather than L so that the prime implicants with $a_i = 0$ can be removed. Therefore, the average number of prime implicants in F is

$$\sum_{k=0}^{n-1}\binom{n}{k}2^{n-k}L^{-2^k}\left(1 - L^{-2^k}\right)^{n-k}(L-1)$$

Since a k-cube has $n-k$ literals, the average number of literals in Eqs. (2), (3), ..., $(t+1)$ is

$$\sum_{k=0}^{n-1}\binom{n}{k}2^{n-k}L^{-2^k}\left(1 - L^{-2^k}\right)^{n-k}(L-1)(n-k)$$

In Eq. (1), p_i occurs multiple times if the number of 1's in a_i is more than one. Since the expected number of 1's in a_i is $l/2$, the expected total number of literals for F is

$$K(n, L) = \sum_{k=0}^{n-1}\binom{n}{k}2^{n-k}L^{-2^k}\left(1 - L^{-2^k}\right)^{n-k} \times (L-1)(n-k+l-1)$$

The value of $K(n, L)$ for several nL combinations is plotted in Fig. 8. This figure shows that the expected

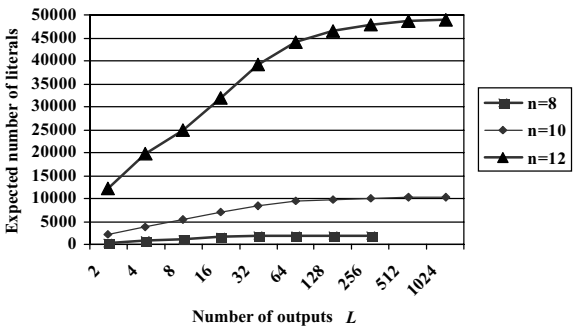


Fig. 8. The expected number of literals $K(n, L)$ of *IL* as a function of its number of outputs L when *IL* has $n = 8$, 10 and 12 inputs.

Target FSM M	No. of input bits of M (u)	No. of output bits of M (w)	No. of states of M (V)	Area of M	Duplication overhead	No. of output check bits	No. of input bits of M^* (n)	No. of states of M^* (S)	Area of M^*	Area overhead	SBF Fault Coverage
ex1	9	19	20	150.	172%	7	16	4	95	63%	96.1%
						8	17	1	77	51%	95.6%
						19	28	2	227	150%	99.9%
						19	8	1	109	59%	96.8%
ex6	5	8	8	89	147%	3	8	2	109	122%	96.8%
						8	13	1	38	52%	95.0%
mc	3	5	4	28.	185%	5	8	2	52	180%	93.8%
						5	8	1	18	66%	80.0%
tbk	6	3	32	215	138%	3	9	4	158	73%	86.7%
						3	9	1	24	11%	15.6%
tma	7	6	20	117	126%	6	12	2	118	101%	93.6%
						6	12	1	45	38%	88.3%
s1488	8	19	48	342	132%	19	27	15	1000	300%	99.9%
						19	27	1	128	37%	99.9%

Fig. 9. Experimental results for some MCNC benchmark circuits.

number of literals $K(n, L)$ increases significantly with the number of outputs L. Furthermore, the number of the outputs increase exponentially with the number of states in M_c. Therefore, the expected number of literals, or equivalently, the monitor area decreases significantly as the number of states in M_c decreases.

We now consider the monitor's fault coverage. Usually, the behavior of a system is not completely specified, leaving more optimization opportunities in the design process. The don't cares in the specification of the target FSM M affect the complexity of both M and its monitor. Therefore, it may be beneficial if we consider the overall area of M and its monitor when the don't cares are exploited. This "design-for-monitoring" problem is beyond the scope of the present paper, however. Since our monitors are designed to detect faults in the final FSMs whose don't cares have been eliminated during the design process, we focus on completely specified FSMs. For a completely specified FSM with u input bits, V states and w output bits, the total number of single next-state faults is $2^u \times V \times (V-1)$ since under the SBF model, any incorrect next state for a given current state and input is a fault. Similarly, the number of single output faults is $2^u \times V \times (2^w - 1)$. Therefore, the total number of SBFs is $2^u \times V \times (2^w + V - 2)$.

When the monitor has a single state, we cannot distinguish the faulty next state from the correct one. In addition, if an input triggers different outputs for different current states, we cannot identify a faulty output if the output is valid for some current states. Therefore, the total number of undetectable faults is $2^u \times V \times (2V-2)$ given that $2^w \geq V$ in the worst case. Consequently, the fault coverage is

$$1 - \frac{2^u \times V \times (2V-2)}{2^u \times V \times (V + 2^w - 2)} = \frac{2^w - V}{V + 2^w - 2} \approx \frac{1 - V/2^w}{1 + V/2^w}$$

If a target FSM has high output/state ratio $2^w/V$, which means that $V/2^w$ is approximately 0, we can usually obtain single-state monitors with high fault coverage. For example, the MCNC benchmark circuit *S1484* in Fig. 9 has 48 states and 19 output bits. On the other hand, single-state monitors typically take much less area than their multiple-state counterparts. Therefore, for FSMs with high output/state ratios, we can often reduce the area significantly by reducing the number of monitor states to one with only minor fault coverage loss.

5. Experimental Results

A set of experiments was performed on the MCNC FSM benchmark circuits [2] to evaluate our monitor design method and the foregoing analysis. Fig. 9 shows the results on several representative benchmarks. The FSMs to be monitored and their properties (number of input bits, number of output bits, number of states and area) are shown in columns 2 to 5, followed by the area overhead of simple duplication. The next four columns present the number of output check bits, inputs, states, and area of the corresponding monitors, respectively. The area overhead and fault coverage are shown in the

FSMs	No. of input bits	No. of output bits	No. of states in target FSMs	Run time (s)
b03	6	4	2057	6700
b05	3	36	70	0.2
b07	2	8	87	0.3
b10	13	6	840	402

Fig. 10. Program run times for some larger FSMs.

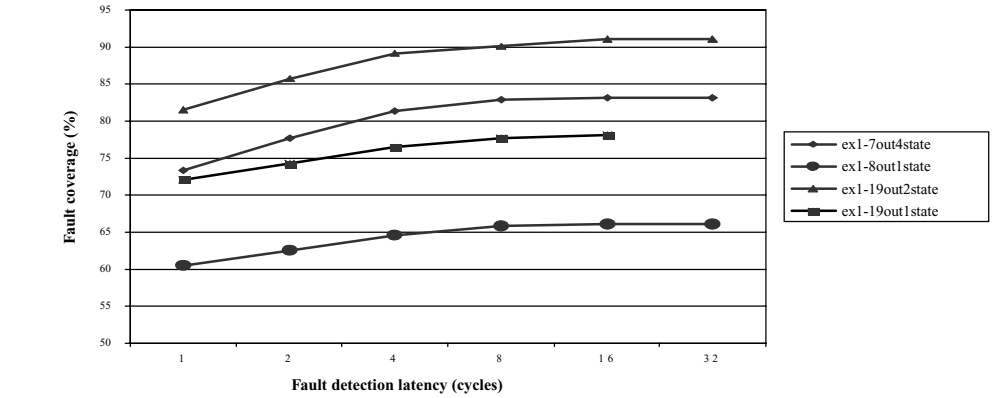
last two columns, which are our main metrics of monitor effectiveness. In the duplication case, we assume that the outputs from the two copies of the target FSMs are compared with dual-rail checker trees [19]. The FSMs to be monitored, their monitors, and the dual-rail checkers were all synthesized using the Synopsys synthesis tools [17], which also reported the areas of the FSMs and their monitors. Area is shown in Fig. 9 as a multiple of the area of a standard two-input NAND.

The results demonstrate that monitors with a few states and high fault coverage can be obtained. For example, the first monitor for *ex1* takes 63% of the area of *ex1*, while its fault coverage is above 96%. However, the monitors with only one state, i.e., combinational monitoring circuits, almost always lead to monitors with smaller area overhead. For example, when the number of states of the monitors for *ex1*, *ex6* and *tma* increases from 1 to 2, the area overheads are almost tripled. This trend holds for all other examples, too. On the other hand, the fault coverage of the one-state monitors for *ex1* and *s1488* is still around 99%. Even for *ex6* and *tma*, their one-state monitors have fault coverage around 90%. However, the fault coverage of circuit *tbk*'s combinational monitor is as low as 15.6%. As analyzed in Section 4, the fault coverage of the combinational monitors depends on the output/state ratios of the target machines. The ratios for all circuits except *tbk* are high, leading to one-state monitors with high fault coverage, which confirms our earlier analysis. The results also show that the overhead of most of the online monitors is smaller than that of duplication. In particular, the overhead of the combinational monitors is much smaller than simple duplication.

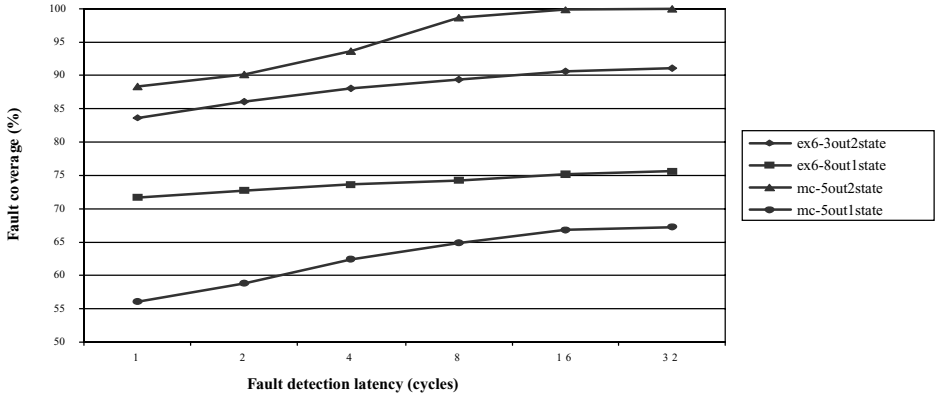
We also applied our method on some larger ITC'99 benchmark circuits [7] to investigate the scalability of the proposed approach. We examined the benchmarks from *b01* to *b10*, among which *b01*, *b02*, and *b06* are excluded because they have fewer than 10 registers. Since these circuits are given in blif format, we extracted their FSM representations using the "*stg_extract -e -c*" routine of SIS [16]. Circuits *b08* and *b09* have tens of thousands of states, and require too much memory for our current implementation. SIS also failed to generate an FSM for *b04* because of its huge number of states. For the remaining circuits, we ran our program on a SUN Blade-1000 with 512 MB of RAM and the run times needed to obtain the combinational monitors are reported in Fig. 10. We can see that the run times for FSMs with tens of states are less than a second. Even for FSMs with more than 2,000 states, the run times are less than two hours. These data give us confidence that processing even larger FSMs like *b08* or *b09* is feasible given a more memory-efficient implementation of our method.

In order to investigate the monitors's fault detection capability under the standard single stuck-at fault (SAF) model, we did another set of experiments. Since the monitor has only one output signal, we can detect an SAF inside the monitor if it forces the output to 1; otherwise, the monitor will not report the error. This problem can easily be solved by using dual-rail logic, for instance, differential cascode voltage switch (DCVS) [6] logic. DVCS has an inherent self-testing property which can provide good coverage for stuck-at faults [13]. Although it is also claimed to have advantages over traditional static CMOS in terms of layout area, its application is limited to small modules due to transistor sizing and crossbar current problems. Therefore, it is more appropriate to use it in the monitor rather than the whole circuit. In our experiments, we only inject SAFs into the target system and keep the monitor fault-free. Because the monitors are designed to detect transient faults, we assume that a fault will disappear after a certain time. Since the duration of a transient fault is usually only a fraction of a cycle [1], we limit the fault duration to one cycle.

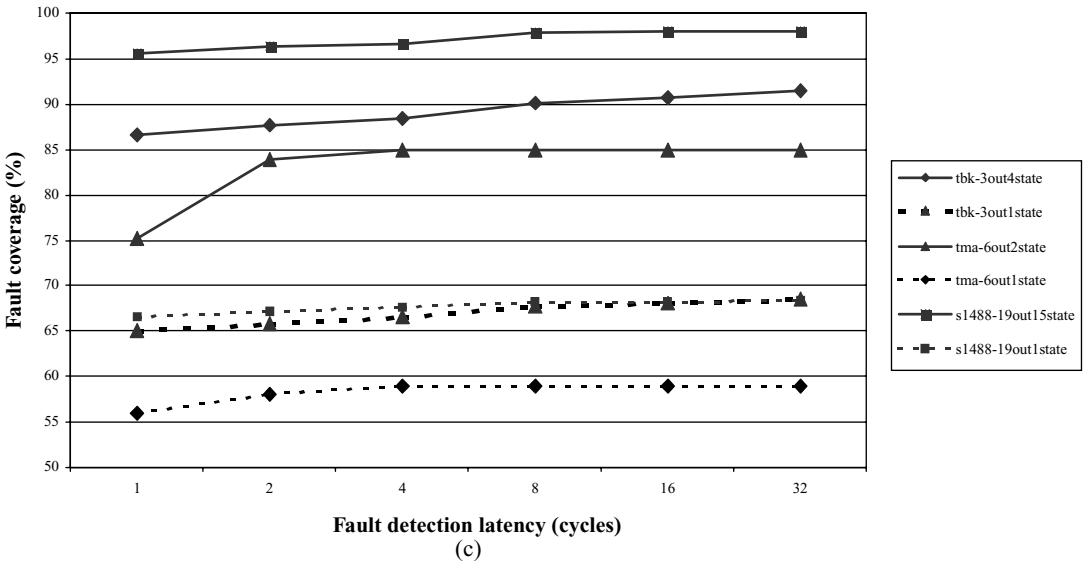
In this set of experiments, we studied the fault coverage of the monitors as a function of the fault detection latency. Intuitively, we regard those SAFs that actually lead to wrong outputs or might lead to wrong outputs later as detectable faults. Therefore, we treat an SAF as detectable if it satisfies one of the following conditions:



(a)



(b)



(c)

Fig. 11. SAF fault coverage as a function of fault detection latency for the monitors of (a) *ex1* (b) *ex6* and *mc*, and (c) *tbk*, *tma*, and *s1488*.

(1) it produces a wrong output within a fixed time after the fault occurrence, which is set to 32 cycles, or (2) it leads to a wrong state in the 32nd cycle but does not cause a wrong output. We enumerated all the SAFs in the circuits and injected every fault 100 times. After a fault is injected into the circuit, we randomly applied input patterns until the fault was detected, or the circuit started to generate both correct outputs and correct next states, or 32 cycles elapsed. We collected the number of faults detected by the monitor at a given latency ranging from one to 32 cycles and divided it by the total number of detectable faults; this is defined as the fault coverage of the monitor under the given latency assumptions.

The SAF fault coverage as a function of the fault detection latency for all the monitors is plotted in Fig. 11. This figure shows that the monitors detect most of the faults in the first cycle after fault injection, and far fewer in the following cycles. Furthermore, the fault coverage increases gradually in the first eight cycles and flattens thereafter for almost all the monitors. Taking the monitor *mc-5out1state*[1] in Fig. 11(b), for example, we can see that 55% of the SAFs are detected at the first cycle, and another 10% are detected in the following seven cycles. Only a few percent of the SAFs are detected after the eighth cycle. The fault coverage here shows trends similar to the fault coverage under the SBF model. Nevertheless, the fault coverage of SAFs varies more than that of SBFs. The fault coverage of the combinational monitors after eight cycles is 60 to 80%. In fact, detectable SAFs form only a small part (10 to 35%) of all injected SAFs.

6. Conclusion

We have presented a systematic method to design on-line monitors for finite-state machines. We first define the single behavior fault (SBF) model and construct a base monitor that detects all SBFs. A heuristic method based on homomorphisms is then systematically applied to reduce the monitor area. The state dependency graph is introduced to guide homomorphism selection. We analyzed the effects of the number of states on the monitor's area overhead and fault coverage. Our results show that monitors with a few states often have both low area overhead and high fault coverage. In FSMs with high output/state ratios, a decrease in the number of monitor states can significantly reduce the area overhead, while incurring marginal fault coverage loss. In fact, it is often beneficial to compress the number of states in the monitor to one for these FSMs. Our experimental results are consistent with the analysis. Simulations under the standard single stuck-at fault (SAF) model show similar trends. Furthermore, the simulations also show that most transient faults are detected quickly—usually within a few clock cycles.

Acknowledgment

This research was supported by the National Science Foundation under grant no. CCR-0073406.

Note

1. This is the version of *mc* with 5 output check bits and a 1-state monitor as defined in Fig. 9.

References

1. H. Cha and J.H. Patel, "A Logic-Level Model for Alpha-Particle Hits in CMOS Circuits," in Proc. *Int. Conf. Computer Design*, 1993, pp. 538–542.
2. Collaborative Benchmarking Lab., North Carolina State Univ., http://www.cbl.ncsu.edu/benchmarks.
3. K. De, C. Natarajan, D. Nair, and P. Banerjee, "RSYN: A System for Automated Synthesis of Reliable Multilevel Circuits," *IEEE Trans. VLSI*, vol. 2, no. 2, pp. 186–195, 1994.
4. D. Gizopoulos, A. Paschalis, and Y. Zorian, "An Effective BIST Scheme for Data Paths," in *Proc. International Test Conf.*, 1996, pp. 76–85.
5. G.D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*, Boston, MA: Kluwer Academic Publishers, 1996.
6. L.G. Heller, W.R. Griffin, J.W. Davis, and N.G. Thoma, "Cascode Voltage Switch Logic: A Differential CMOS Logic Family," *Dig. Tech. Papers*, *ISSCC*, pp. 16–17, 1984.
7. ITC'99 Benchmark Documentation. http://www.cerc.utexas.edu/itc99-benchmarks/bendoc1.html.
8. R. Karri and N. Mukherjee, "Versatile BIST: An Integrated Approach to On-Line/Off-Line BIST," in *Proc. International Test Conf.*, 1998, pp. 910–917.
9. H.B. Kim, D.S. Ha, T. Takahashi, and T.J. Yamaguchi, "A New Approach to Built-In Self-Testable Datapath Synthesis Based on Integer Linear Programming," *IEEE Trans. VLSI*, vol. 8, pp. 594–605, Oct. 2000.
10. Z. Kohavi, *Switching and Finite Automata Theory*, 2nd ed., New York: McGraw-Hill, 1978.
11. H.R. Lewis and C.H. Papadimitriou, *Elements of the Theory of Computation*, Englewood Cliffs, NJ: Prentice-Hall, 1981.
12. A. Mahmood and E.J. McCluskey, "Concurrent Error Detection Using Watchdog Processors—A Survey," *IEEE Trans. Computers*, vol. 37, no. 2, pp. 160–173, 1988.
13. R.K. Montoye, "Testing Scheme for Differential Cascode Voltage Switch Circuits," *IBM Tech. Disc. Bull.*, vol. 27, pp. 6148–6152, 1985.

14. A. Morozov, V.V. Saposhnikov, V.V. Saposhnikov, and M. Gossel, "New Self-Checking Circuits by Use of Berger Codes," in *Proc. 6th IEEE On-Line Testing Workshop*, 2000, pp. 141–146.
15. S. Ravi, G. Lakshminarayana, and N.K. Jha, "Tao-BIST: A Framework for Testability Analysis and Optimization for Built-In Self-Test of RTL Circuits," *IEEE Trans. CAD*, vol. 19, pp. 894–906, 2000.
16. E. Sentovich, K. Singh, C. Moon, H. Savoj, R. Brayton, and A. Sangiovanni-Vincentelli, "Sequential Circuit Design Using Synthesis and Optimization," in *Proceedings of ICCAD*, 1992, pp. 328–333.
17. Synopsys, Synopsys Design Analyzer datasheet, http://www.synopsys.com/products/logic/design_compiler.html.
18. N.A. Touba and E.J. McCluskey, "Logic Synthesis of Multilevel Circuits with Concurrent Error Detection," *IEEE Trans. CAD*, vol. 16, no. 7, pp. 783–789, 1997.
19. J. Wakerly, *Error Detecting Codes, Self-Checking Circuits and Applications*. Amsterdam: North-Holland, 1982.
20. C. Zeng, N. Saxena, and E.J. McCluskey, "Finite State Machine Synthesis with Concurrent Error Detection," in *Proc. International Test Conf.*, 1999, pp. 672–679.

Feng Gao received his B.S. degree in mathematics at Shandong University in 1997 and his M.E. degree at the Institute of Computing Technology, Chinese Academy of Science, Beijing in 2000. He is currently a Ph.D. candidate in the Department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor. His research interests are in the area of low power circuit design, power grid optimization, testing, and DFT.

John P. Hayes is the Claude E. Shannon Professor of Engineering Science at the University of Michigan, Ann Arbor, where he teaches and does research in the areas of computer-aided design, verification and testing; VLSI circuits; embedded system architecture; and quantum computing. He received the B.E. degree from the National University of Ireland, Dublin, and the M.S. and Ph.D. degrees from the University of Illinois, Urbana-Champaign. At Illinois he participated in the design of the ILLIAC III computer. In 1970 he joined the Shell Benelux Computing Center in The Hague, where he worked on mathematical programming. From 1972 to 1982 Dr. Hayes was a faculty member of the Departments of Electrical Engineering-Systems and Computer Science of the University of Southern California. He joined the University of Michigan in 1982 and was the founding director of Michigan's Advanced Computer Architecture Laboratory (ACAL). Dr. Hayes is the author of numerous technical papers, several patents, and five books, including *Hierarchical Modeling for VLSI Circuit Testing*, (Kluwer, 1990, coauthored with D. Bhattacharya) and *Computer Architecture and Organization*, (3rd edition, McGraw-Hill, 1998). He has served as editor of various technical journals, including the *IEEE Transactions on Parallel and Distributed Systems* and the *Journal of Electronic Testing*. Dr. Hayes is a Fellow of IEEE and ACM, and a member of Sigma Xi.