Optimized Synthesis Techniques for Testable Sequential Circuits

Bernhard Eschermann and Hans-Joachim Wunderlich, *Associate Member, IEEE*

Abstract—**Innovative synthesis for testability strategies aim at considering testability while synthesizing a circuit, whereas conventional design for testability methods modify the design after the circuit structure is synthesized. We describe a synthesis approach that maps a behavioral FSM description into a testable gate-level structure. The term "testable" in this context, besides implying the existence of tests, also means that the application of test patterns is facilitated. Depending on the test strategy, the state registers of the FSM are modified e.g. as scan path or self-test registers. The additional functionality of these state registers is utilized in system mode by interpreting them as "smart" state registers, capable of producing certain state transitions on their own. To make the best use of such registers, we propose a novel state encoding strategy based on an analytic formulation of the coding constraint satisfaction problem as a quadratic assignment problem. An additional minimization potential can be exploited by appropriately choosing the pattern generator for self-testable designs. Experimental results indicate that, compared with conventional design for testability approaches, significant savings are possible this way.**

I. Introduction

CONVENTIONAL design for testability methods commonly require circuit modifications *after* the functional design is finished. Supplementary hardware used only for testing purposes has to be added (e.g. [15], [25], [35]). Testability can also be taken into account *during* the synthesis of the circuit ("synthesis for testability"). The test strategy is determined in advance, based on considerations such as test generation complexity, required test equipment, test application effort, tolerable hardware overhead, and fault coverage. The advantages of this approach are twofold. First, the circuit is designed for a specific test strategy; logic design decisions can be targeted toward obtaining circuits which are easily testable "by construction" (cf. [1], [14], [16], and [17]). Second, test hardware, which has to be implemented anyway, can be utilized in system mode instead of being superfluous after the test is finished, thus reducing the amount of logic needed to implement the system functionality.

Manuscript received April 3, 1990; revised September 24, 1990. This paper was recommended by Associate Editor K. Keutzer.

The authors were with the Institut für Rechnerentwurf und Fehlertoleranz, Fakultät für Informatik, Universität Karlsruhe, Zirkel 2, 7500 Karlsruhe, Germany. They are now with the Department of Electrical Engineering and Computer Science, University of Siegen, 5900 Siegen, Germany.

IEEE Log Number 9102368.

This paper describes an approach for transforming a behavioral finite state machine (FSM) description into a structural description supporting a given test strategy. We focus on the synthesis of FSM's with state registers modified to enhance testability, e.g. scan path or self-test registers, called smart state registers in the sequel. For an external test a scan path often stays necessary even if the circuit is sequentially irredundant (cf. [14] for a definition), since the worst-case length of an input sequence to detect a fault otherwise increases exponentially with the number of storage elements [29]. This is particularly critical if the FSM is embedded in a larger circuit such that its inputs and outputs are not directly accessible. The application of test patterns is further facilitated by including pattern generators for implementing a self-test.

The main idea is that in test mode the above-mentioned storage elements cycle through a sequence of states. If the encodings of the present and the next state of the FSM are consecutive elements in this cycle, the state transition does not have to be implemented by additional logic, and the next state can be generated by using the test mode. By utilizing the test logic for implementing a part of the system functionality, hardware overheads of testing can be reduced.

In the sequel we first review some design for testability strategies for FSM's and summarize their properties in an abstract model. In Section III we give a synthesis procedure that utilizes these properties to reduce the amount of combinational logic needed to implement FSM's. Section IV analyzes the solution in terms of testability and area consumption. The main results are summarized in Section V.

II. Basic Principles

A. Finite State Machines

The *behavior* of a synchronous sequential circuit can be modeled with an FSM description (e.g. a state transition diagram), its *structure* by an interconnection of combinational logic and storage elements (see Fig. 1). In integrated circuits these storage elements are generally realized with D-type latches or flip-flops. We assume a general Mealy machine

$$\mathfrak{M} = (I, S, O, f_s, f_o)$$

with the input set I, the state set S, the output set O, and the next state and output functions

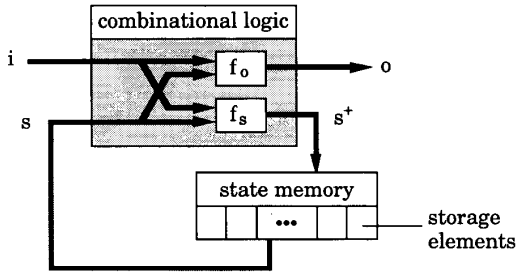


Fig. 1. Finite state machine model.

$$f_s: I \times S \rightarrow S, f_s(i, s) = s^+$$

$$f_o: I \times S \rightarrow O, f_o(i, s) = o.$$

The FSM model is particularly useful for circuits with irregular combinational logic and a relatively small number of storage elements.

B. Test Strategies and Test Equipment

One of the most widely used design for testability methodologies is to incorporate a scan path, in which all the flip-flops are connected as a serial shift register in test mode. Test patterns thus can be shifted into the flip-flops; test responses can be shifted out. Examples of this test methodology can be found in [15] (LSSD) and [35] (edge-triggered scan design, ETSD).

Self-testable circuits contain test pattern generators as well as response analyzers on chip. Depending on the test strategy, the pattern generator may be a linear feedback shift register (LFSR) [25], a cellular automaton [22], or a nonlinear feedback shift register [10]. In all these cases the memory elements work as D flip-flops in system mode, whereas in test mode a complex specialized behavior is desired. Although in some cases the same register may be used for pattern generation and response analysis [24], in the circuits typically modeled as FSM's this is more difficult [9]. An architecture for FSM's in which the system flip-flops can be configured as pattern generator and the responses are compacted in a separate multiple-input LFSR (MISR) was proposed in [4] and [33] (see Fig. 2).[1] The MISR may be saved if the observability of state variables is secured by other means (cf. e.g. [20]).

C. Target Structure of the Synthesis Process

Both scan designs and self-testable designs demand more area than designs without test aids. The additional area is required to modify the storage elements. At least two modes of operation are needed. In system mode the storage elements merely load the outputs of the combinational logic, whereas in test mode some provision is made to apply test patterns or to capture test responses. Since a fair amount of hardware is spent to enable the storage elements to perform these additional tasks, it would be desirable to utilize this increased functionality during normal system operation as well.

[1]Pattern generation and response analysis for the primary inputs and outputs are not shown.

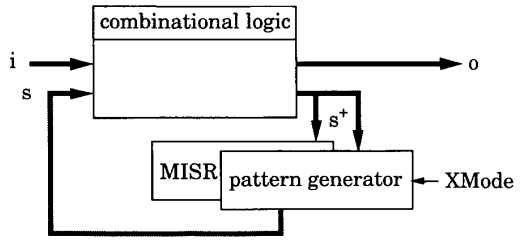


Fig. 2. Architecture of self-testable FSM's.

Pattern generators for self-testable designs in autonomous mode cycle through a fixed sequence of states to stimulate the circuit. This property can also be used in system mode if the encodings of the present and the next state are consecutive elements in this cycle. Whenever the next state code is produced by the pattern generation register, which has to be implemented for testing purposes anyway, it is not necessary to generate it in the next state logic. Replacing the next state entries with don't cares for all such transitions greatly increases the potential for logic optimization of the combinational logic. Fig. 3 illustrates a possible realization of this idea [16]. An additional output signal "Mode" determines whether the state machine flip-flops behave like ordinary D flip-flops or work as a pattern generator. In the second case the state register is "smart" enough to generate the next state on its own, so the next state signals generated by the combinational logic can be set to arbitrary values.

A similar result can be obtained for scan paths by feeding back the contents of the last storage element into the first element of the chain. Depending on whether or not the feedback is inverted, the resulting state memory becomes a loadable Johnson counter or ring counter. Such a "counter" is not mandatory—it is possible to choose any other feedback structure as well—but these simple feedback structures minimize the necessary overhead. There are three modes: a test mode (scan mode), a normal system mode, in which the storage elements work as D flip-flops, and a feedback mode, in which the next state is produced by operating the storage elements as a counter. The feedback mode and the test mode are actually equivalent for all flip-flops in the scan chain (see Fig. 4). A special case, PLA-based FSM's implemented with loadable Johnson counters as state memories, was previously investigated and found to be quite effective [2]. However, the synthesis procedure presented there is not general enough for the majority of test registers.

Both approaches can be generalized in that there is a "smart" state register (in the sequel abbreviated SSR) that is capable of generating a number of state transitions on its own. This characteristic can be utilized to reduce the combinational logic of the FSM. The price to be paid is that an additional output signal must be produced by the combinational logic to control the state register mode. Ex-

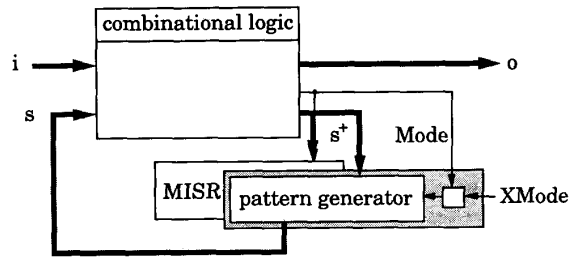


Fig. 3. Self-testable FSM with "smart" state memory.

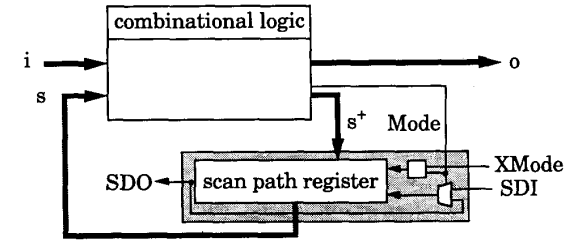


Fig. 4. FSM with scan path and "smart" state memory.

perimental evidence indicates that this expense is small compared with the possible savings.

The main problem is to find a state assignment that will reduce the combinational logic required to implement the modified FSM. Conventional state assignment algorithms are not suitable for this purpose because the pattern generation capability of the state memory cannot be taken into account until after the state assignment. On the other hand, it is not necessarily advantageous to maximize the number of state transitions generated by the SSR, since it may be impossible to further minimize the combinational network for the remaining state transitions. Both aspects, minimization by replacing next state entries with don't cares and minimization by conventional logic optimization techniques, must be regarded concurrently during state assignment.

D. State Assignment Methods

Many recent state assignment algorithms for PLA-based FSM's[2] are based on the work of DeMicheli *et al.* [11], [12]. After a "symbolic minimization" of the FSM, in which a symbolic cover $\mathcal{C}$ with certain *coding constraints* is created, the task of the state assignment algorithm is to satisfy as many of these coding constraints as possible.

- An *adjacency constraint* requires a set of states to be encoded in a Boolean subspace not containing any other states [11].
- A *covering constraint* between a pair of states requires that each state variable which is "1" in the state to be covered has a correspond to a "1" in the other state [12].

The process is illustrated with the example in Fig. 5, in which the state groups $Z_1 \cup Z_2$ and $Z_1 \cup Z_3$ correspond

[2]Multilevel logic is treated in subsection III-D.

state	input	n.state	output
Z_1	0–	Z_2	10
Z_1	10	Z_3	11
Z_1	11	Z_2	00
Z_2	00	Z_2	10
Z_2	01	Z_1	11
Z_2	10	Z_3	00
Z_2	11	Z_3	01
Z_3	0–	Z_1	00
Z_3	10	Z_3	11
Z_3	11	Z_1	00

state	code	input	n.state	code	output
$Z_1 \cup Z_2$	–1	0–	$Z_2 \subseteq Z_1$	01	10
Z_2	01	01	$Z_1 \setminus (Z_2 \cup Z_3)$	––	––
Z_2	01	–1	$Z_3 \subseteq Z_1$	10	01
$Z_1 \cup Z_3$	1–	10	Z_3	10	11
Z_1	11	11	Z_2	01	00
Z_2	01	10	Z_3	10	00
Z_3	10	0–	Z_1	11	00
Z_3	10	11	Z_1	11	00

Fig. 5. Symbolic minimization and satisfaction of coding constraints.

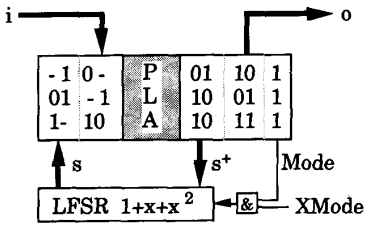


Fig. 6. PLA realization of the modified self-test structure.

to the adjacency constraints and $Z_2 \subseteq Z_1$ and $Z_3 \subseteq Z_1$ to the covering constraints. Since the first and the third line in the minimized table of Fig. 5 completely cover the second line, the outputs in that line can be left unspecified. The state assignment Z_1: 11, Z_2: 01, Z_3: 10 satisfies all these constraints and the number of PLA product terms is reduced to 7.

For four of the symbolic implicants no reduction is possible, because they do not give rise to any constraints and, therefore, are not considered in the encoding process. They can, however, be completely saved if the ideas of subsection II-C are applied. A self-test register based on an LFSR with a feedback polynomial $p(x) = 1 + x + x^2$ can produce the state transitions 00 → 00, 11 → 01, 01 → 10, and 10 → 11 on its own. With Mode = 0 causing the SSR to switch to LFSR mode, only three product terms are needed for the combinational logic of the FSM (Fig. 6).

III. Synthesis Procedure

A. Problem Formulation

To incorporate the ideas of subsections II-C and II-D into a synthesis procedure for PLA-based FSM's, three different mechanisms of reducing the number of product terms should be taken into account: adjacency relations, covering relations, and transitions realizable with the SSR. Our approach is based on the analytical formulation of a cost function for the state assignment problem, in which the SSR transitions can be easily incorporated. Let s be the number of states of the FSM, and n the number of bits used for the encoding of these states. We use the minimal number of state bits, $n_0 = \lceil \log_2 s \rceil$, since choosing a number $n > n_0$ only rarely decreases the PLA area [23], [32], particularly if the area of the additional flip-flops is also considered.

Definition 1: The *adjacency matrix*, A, of a minimized

symbolic cover $\mathcal{C}$ is an $s \times s$ matrix of nonnegative integer entries a_{ij}. For $i \neq j$ the value a_{ij} corresponds to the number of adjacency constraints of $\mathcal{C}$ to which states i and j both belong; the diagonal entries a_{ii} are set to 0.

Definition 2: The *distance matrix*, D, is a $2^n \times 2^n$ matrix with Boolean entries $d_{ij} \in \{0, 1\}$, where $d_{ij} = 0$ if the Hamming distance of codes i and j is less than or equal to 1, and $d_{ij} = 1$ otherwise.

Similar adjacency and Hamming distance values have been used in other state assignment algorithms (see [3] for one of the first references). In a good assignment, state pairs appearing together in many symbolic implicants are assigned to codes with small Hamming distances, preferably to adjacent codes. Let

$$\phi: \{s_1 \cdots s_s\} \rightarrow \{(q_1^k \cdots q_n^k) | k \in \{1 \cdots s\}\}$$

be an injective mapping, which assigns a unique code $(q_1^k \cdots q_n^k)$ to each state s_k. Then a partial cost of

$$\alpha(i, j, \phi) := \tfrac{1}{2} \cdot a_{ij} \cdot d_{\phi(i)\phi(j)}$$

is incurred by the assignment of a pair of nonadjacent codes $\phi(i)$ and $\phi(j)$ ($d_{\phi(i)\phi(j)} = d_{\phi(j)\phi(i)} > 0$) to states appearing together in symbolic implicants of $\mathcal{C}$ ($a_{ij} = a_{ji} > 0$). The factor $\frac{1}{2}$ takes the symmetry of the matrices A and D into account. Matrix A collects the information about pairs of *states* from the minimized symbolic cover; matrix D contains the information about pairs of *codes* necessary for the encoding process. To include the covering conditions, two additional matrices are necessary.

Definition 3: The *state covering matrix*,[3] S, of a minimized symbolic cover $\mathcal{C}$ is an $s \times s$ matrix of nonnegative integer entries s_{ij}, where s_{ij} corresponds to the number of implicants of $\mathcal{C}$ that require covering state j with state i.

Definition 4: The *code covering matrix*, C, of an r-bit code is a $2^n \times 2^n$ matrix with Boolean entries $c_{ij} \in \{0, 1\}$, where $c_{ij} = 0$ if code i covers code j, and $c_{ij} = 1$ otherwise.

A cost function similar to $\alpha(i, j, \phi)$ can be formulated for the covering constraints. A partial cost of

$$\chi(i, j, \phi) := s_{ij} \cdot c_{\phi(i)\phi(j)}$$

is incurred if state i should cover state j ($s_{ij} > 0$) and code $\phi(i)$ does not cover code $\phi(j)$ ($c_{\phi(i)\phi(j)} > 0$). To describe the effect of the SSR transitions, two more matrices are introduced.

Definition 5: The *transition matrix*, T, of a minimized symbolic cover $\mathcal{C}$ is an $s \times s$ matrix of nonnegative integer entries t_{ij}, where t_{ij} is the number of implicants with a present state i and next state j, which do not belong to an adjacency or covering constraint of $\mathcal{C}$.

Definition 6: The *pattern matrix*, P, of an r-bit pattern generator is a $2^n \times 2^n$ matrix with entries $p_{ij} \in \{0, 1\}$, where $p_{ij} = 0$ if code j is the successor of code i in the sequence generated by the autonomous SSR, and $p_{ij} = 1$ otherwise.

[3]Note that this matrix is a simple extension of the matrix termed adjacency matrix in [12]. In spite of the naming, it does not correspond to the matrix introduced in Definition 1 of this paper.

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \qquad D = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$S = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$T = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 2 & 0 & 0 \end{bmatrix} \qquad P = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

Fig. 7. Example matrices for the FSM of Fig. 5.

If transitions not minimizable with the help of adjacency or covering constraints ($t_{ij} > 0$) cannot be realized with the help of SSR transitions ($p_{\phi(i)\phi(j)} > 0$), this corresponds to a cost function value of

$$\tau(i, j, \phi) := t_{ij} \cdot p_{\phi(i)\phi(j)}.$$

Example: For the cover of Fig. 5, the matrices in Fig. 7 are obtained. In matrices A, S, and T the ith row/column corresponds to state Z_i. The order of entries in matrices D, C, and P corresponds to the codes 00, 01, 10, and 11 in that sequence. For matrix P, an LFSR with a feedback polynomial $p(x) = 1 + x + x^2$ was used. Note that A and D are symmetric matrices, whereas S, C, T, and P are not.

Finding an appropriate assignment ϕ such that as many adjacency and covering constraints as possible are satisfied and that the remaining state transitions are preferably produced by the SSR can then be formulated as a combinatorial optimization problem:

$$\begin{aligned} \min(\phi): V(\phi) &= \sum_{i,j} [k_1 \cdot \alpha(i, j, \phi) + k_2 \cdot \chi(i, j, \phi) \\ &\quad + k_3 \cdot \tau(i, j, \phi)] \\ &= V_a(\phi) + V_c(\phi) + V_t(\phi) \end{aligned}$$

with certain weighting factors $k_1, k_2, k_3 \geq 0$. The complete cost function consists of three terms, one for the violation of adjacency constraints, a second for the violation of covering constraints, and a third for the remaining transitions not realizable with the help of the SSR.

Definition 7: The *assignment matrix*, X, of a state assignment ϕ is an $s \times 2^n$ matrix with Boolean entries $x_{ik} \in \{0, 1\}$, where $x_{ik} = 1$ if $k = \phi(i)$ and $x_{ik} = 0$ otherwise.

Using this matrix, the problem can be formulated as a 0–1 integer program in which the product $x_{ik} \cdot x_{jl}$ selects all those cost values belonging to the current assignment:

$$\begin{aligned} \min(X): V(X) &= \sum_{i,j,k,l} [k_1 \cdot \tfrac{1}{2} \cdot a_{ij} \cdot d_{kl} + k_2 \cdot s_{ij} \cdot c_{kl} \\ &\quad + k_3 \cdot t_{ij} \cdot p_{kl}] \cdot x_{ik} \cdot x_{jl} \\ &= \sum_{i,j,k,l} v_{ijkl} \cdot x_{ik} \cdot x_{jl} \end{aligned} \tag{1}$$

$$\text{w.r.t.:} \sum_i x_{ik} \leq 1 \qquad \forall k \tag{2}$$

$$\sum_k x_{ik} = 1 \qquad \forall i \tag{3}$$

$$x_{ik} \in \{0, 1\} \qquad \forall i, k. \tag{4}$$

The $s + 2^n$ linear constraints (2) and (3) ensure that X indeed represents an injective mapping ϕ, *i.e.*, that each state is assigned exactly one code and each code is assigned to at most one state.

Example: For the example of Fig. 6 the assignment matrix, X, is

$$X = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

and the minimal cost value $V(X) = 0$ is obtained. Both adjacency constraints are satisfied ($V_a(X) = 0$), since the state groups $\{Z_1, Z_2\}$ and $\{Z_1, Z_3\}$ are both coded with a Hamming distance of 1. The code for Z_1 covers the code for Z_2 and Z_3 ($V_c(X) = 0$) and all the remaining transitions are realized with the help of SSR transitions ($V_i(X) = 0$).

The problem comprising (1)–(4) is a well-known combinatorial optimization problem, the *quadratic assignment problem* [19]. It was proven to be NP-complete [18], but because of its relevance for many applications much effort was spent in developing feasible solution methods (see [7] for an overview). We use an exact algorithm using implicit enumeration techniques from [5] for FSM's with up to eight states. For larger machines, at least 16 state codes have to be considered and the exact algorithm becomes quite slow. In these cases, we use heuristic algorithms [6], [8]. Since state assignment is an NP-hard problem [34], the use of heuristics is inevitable. However, by representing the state assignment problem as a quadratic assignment problem, a very general problem formulation and solution method are obtained, allowing different types of coding constraints and arbitrary SSR's.

B. Adequacy of the Formulation

In the above framework, only pairwise adjacency relations can be represented. A group of m states to be encoded in a minimal subspace of Boolean r-space is split into $\frac{1}{2} m (m - 1)$ state pairs. The minimization (1) attempts to code all these state pairs with a Hamming distance of 1, which is obviously not possible (for $m > 2$).

Within a single state group, the minimization of $\Sigma \frac{1}{2} a_{ij} \cdot d_{\phi(i)\phi(j)}$ can be reduced to a minimization of $\Sigma d_{\phi(i)\phi(j)}$ over all admissible code pairs $\phi(i)$, $\phi(j)$, because a_{ij} is constant within this state group. Alternatively, $\Sigma (1 - d_{\phi(i)\phi(j)})$ can be maximized; i.e., the maximum number of adjacent codes is sought. Situations with isolated state codes, i.e., state codes not adjacent to any other state code, are obviously not optimal; the number of adjacencies could simply be increased by encoding the isolated state with any unused code adjacent to one of the other state codes. The remaining possibilities for $m = 4$ are illustrated in Fig. 8.

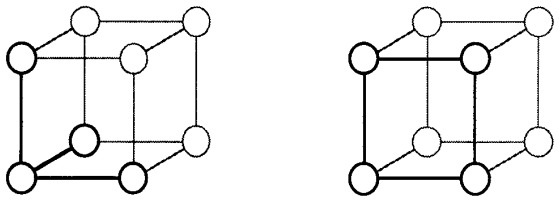


Fig. 8. Encoding possibilities for a state group with $m = 4$ states.

In what follows we prove that maximizing the number of adjacent code pairs guarantees not only that pairs of symbolic implicants can be merged, but also that adjacency groups S with an arbitrary number of states $m = |S|$ are encoded in a minimal subspace of dimension $\lceil \log_2 m \rceil$. As a consequence, for $m = 2^k$ all corresponding transitions can indeed be merged into one. Let

$$m = \sum_{i=0}^{k} c_i 2^i, \qquad c_i \in \{0, 1\},\ c_k = 1 \quad \text{and}$$

$$|\{c_i | c_i = 1\}| = \#c.$$

Theorem 1: Encoding the elements of an adjacency group S in a Boolean subspace of minimal dimension $\lceil \log_2 |S| \rceil$ such that $\#c$ subsets of states with cardinality 2^i (corresponding to the values $c_i = 1$) are encoded in cubes of dimension i, maximizes the number of pairwise adjacencies.

Proof: Theorem 1 is proven by induction on k. Let $A(S)$ be the maximal number of pairwise state adjacencies within a state group S and let $A(S_1, S_2)$ be the maximal number of adjacencies between states of different groups S_1 and S_2.

1) $k = 1$, $m \in \{1, 2, 3\}$, trivial.
2) Assume that Theorem 1 is valid for $m = \Sigma_{i=0}^{k-1} c_i 2^i$.
3) Prove that Theorem 1 is valid for $m' = \Sigma_{i=0}^{k} c_i 2^i$. For $m' = 2^k$, the number of edges in a hypercube of dimension k is $\frac{1}{2} m'k$. Having encoded the states in a hypercube of dimension k, there exists a state variable q_i that partitions S into two disjoint subsets $S_A = \{s \in S | q_i = 0\}$ and $S_B = \{s \in S | q_i = 1\}$. Both subsets (dimension $|S_A| = |S_B| = m'/2 = 2^{k-1}$) are encoded in disjoint hypercubes of dimension $k - 1$. Because of 2) the number of adjacencies within the subsets is maximized by this encoding,

$$A(S_A) = A(S_B) = \frac{1}{2} \frac{m'}{2} (k - 1).$$

The number of adjacencies between the disjoint hypercubes S_A and S_B is bounded by

$$A(S_A, S_B) \leq \min (|S_A|, |S_B|).$$

So we have

$$A(S) = A(S_A) + A(S_B) + A(S_A, S_B)$$
$$\leq 2 \cdot \frac{1}{2}\frac{m'}{2}(k-1) + \frac{m'}{2} = \frac{1}{2}m'k.$$

Therefore, $\frac{1}{2}m'k$ is an upper bound for $A(S)$. By encoding the states in a hypercube of dimension k, this upper bound is reached; i.e., the number of adjencies is maximized. The case $m' > 2^k$ can be treated analogously by using a state variable to partition S into two disjoint subsets S_A, $|S_A| = 2^k$ and S_B, $|S_B| < 2^k$. ■

In experiments it turned out that changing the entries of the distance matrix, D, from Boolean values $d_{ij} \in \{0, 1\}$ to nonnegative integer values d_{ij} equal to the Hamming distance of codes i and j improved the results for two-level state assignment. The reason seems to be that more information is provided by this modified matrix D, although the optimality proof given is no longer valid in this case.

If m is not a power of 2 and the total number, s, of states is smaller than the number 2^n of available state codes, it may still be possible to merge all the corresponding m transitions: Some codes in the Boolean subspace determined by these transitions can be left unused by encoding the other states outside of that subspace. The cost function value obtained for (1) does not reflect this possibility. If a small negative value is assigned to all the a_{ij} previously set to 0, i.e., those state pairs not to be encoded in a Boolean subspace, the cost value decreases if states i and j are assigned to codes with a large Hamming distance; hence nonadjacent states are kept apart from each other. In what follows this modified cost function is denoted $\tilde{V}_a(X)$.

C. Experimental Validation

Deciding whether a Boolean function can be implemented in two-level form with some number, K, of product terms belongs to the class of NP-complete problems [18]. Therefore, we cannot expect to find a cost function for a given state assignment accurately predicting the complexity of the resulting combinational logic, which can be computed in polynomial time. To be useful as a heuristic, the cost function should, however, be able to distinguish between good and bad state assignments with a high probability. Another problem is that exact logic minimization is infeasible for larger Boolean functions. Even if *the* optimal state assignment were known, it might not yield the best implementation obtainable in practice. Hence the only possibility of testing the quality of the cost function is to validate it experimentally.

For several FSM examples were generated a series of random state assignments. For each state assignment we evaluated the cost function $\tilde{V}_a(X)$ and $\tilde{V}_a(X) + V_t(X)$ and minimized the resulting combinational logic. Comparing the cost function value and the actual implementation cost

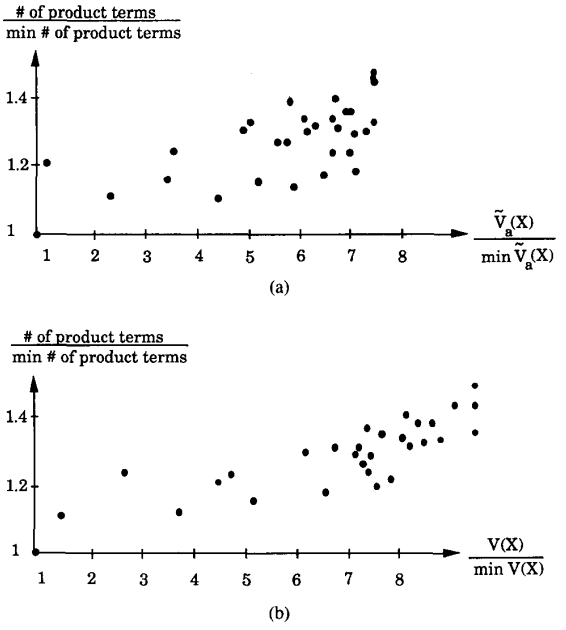


Fig. 9. Predictive value of the cost function.

should indicate how good the cost function estimate is. Results for a typical example are illustrated in Fig. 9. It turns out that the cost function $\tilde{V}_a(X)$ alone gives a mediocre prediction for the ranking of different state assignments with respect to the complexity of the combinational logic (see Fig. 9(a)). The correlation coefficient, r, of a linear regression in Fig. 9(a) is $r = 0.64$; Spearman's rank correlation coefficient $r_S = 0.62$. Unfavorable results particularly have to be expected when only a small number of adjacency constraints can be derived: In this case the minimization potential is mainly due to covering relations and other effects not modeled in the cost function. By utilizing the SSR transitions, the FSM transitions not included in adjacency constraints can also be minimized. This is considered in the composite cost function $V(X) = \tilde{V}_a(X) + V_t(X)$, which gives reasonably accurate estimates for the resulting logic (see Fig. 9(b) for the FSM example of Fig. 9(a)). The correlation coefficient of a linear regression, r, in Fig. 9(b) is $r = 0.81$; Spearman's rank correlation coefficient $r_S = 0.77$.

D. Multilevel Logic

Multilevel logic synthesis programs differ from two-level programs in that expressions can be factored and common subexpressions can be extracted. The multilevel state assignment algorithm developed by Devadas *et al.* [13] seeks to create common cubes. Two approaches can be distinguished: one maximizes the *size* of common cubes (''fan-out oriented''), while the other maximizes the *frequency* with which common cubes occur (''fan-in-oriented''). In either case, a heuristic cost function is minimized. Using the assignment matrix, X, for repre-

senting the resulting state assignment, the cost function can be written as

$$V_m(X) = \sum_{i,j,k,l} (a_{ij} \cdot d_{kl}) \cdot x_{ik} \cdot x_{jl},$$

where d_{kl} represents the Hamming distance of codes k and l. Since this cost function has the same mathematical structure as the cost function for two-level logic, SSR transitions can be taken into account in exactly the same way. The values a_{ij} are not derived by symbolic minimization but by estimating the number of common cubes that can be extracted from the resulting combinational network. The coefficients a_{ij} now represent the importance of encoding states i and j with a small Hamming distance. The adjacency values a_{ij} in our algorithm are generated in a way similar to that in [13], except that we count multiple edges in the state transition graph only once, since the creation of common cubes caused by the state encoding does not depend on the multiplicity of edges.

Thus, a unified framework for both two-level and multilevel state assignment is obtained. Design style specific knowledge enters into the coding constraint generation step, whereas the solution algorithm for the mathematical optimization problem can be kept generic.

E. Implications of the SSR Structure

Until now the characteristics of the SSR were needed only to determine the pattern matrix, P. Therefore, any SSR described by such a matrix can be accommodated: Johnson counters, ring counters, LFSR's, nonlinear feedback shift registers, cellular automata, etc. For self-testable designs the choice is based on the required fault coverage and the hardware overhead. LFSR's with primitive feedback polynomials can be used to obtain a maximum length test sequence. In general one such polynomial is taken from a table listing primitive polynomials over GF(2), e.g. [31]. In order to minimize LFSR area, i.e., to reduce the number of XOR's, minimum weight polynomials are often preferred.

In the synthesis method presented in this paper, the state assignment depends on the feedback polynomial and so does the complexity of the resulting combinational logic. The optimal solution requires an LFSR structure that is best suited to the FSM under consideration. Choosing the feedback polynomial of the LFSR by minimizing the cost function presented earlier over all pertinent polynomials $p(x)$, i.e., all pattern matrices P, thus provides an additional degree of freedom. Although the number of polynomials which have to be considered grows exponentially with the number of flip-flops, it is still small for typical FSM's, such that encoding the states for all polynomials to choose the best solution is feasible. An interesting fact is that it does not matter whether a standard implementation or a modular implementation [28] of the LFSR is chosen as long as a minimum weight polynomial of the form $p(x) = 1 + x^i + x^n$, $1 \le i < n$, is used. The reason is that in this case the two implementations differ only in the sequence of flip-flops (see Fig. 10), which is irrelevant to state assignment and logic minimization. Matrix D remains unchanged, because the permutation of coding columns does not influence the Hamming distance of codes.

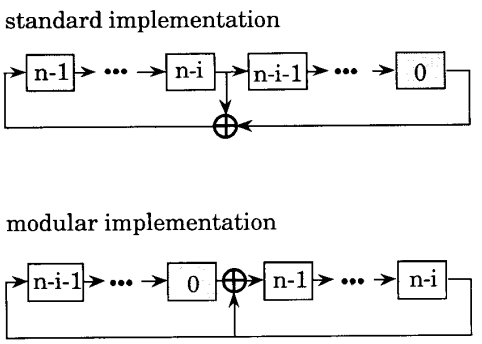


Fig. 10. Standard and modular LFSR with $p(x) = 1 + x^i + x^n$.

Let $p(x) = g_0 + g_1 x + \cdots + g_{n-1}x^{n-1} + g_n x^n$, $g_0 = g_n = 1$, be the LFSR feedback polynomial. The contents of the n LFSR states $q = (q_0 \cdots q_{n-1})^T$ can be described by the recurrence equation $q(t+1) = G \cdot q(t)$ with

$$G = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & 1 \\ g_n & g_{n-1} & \cdots & g_2 & g_1 \end{bmatrix}.$$

The reciprocal polynomial of $g(x)$ is $p^r(x) = 1 + g_{n-1}x + \cdots + g_1 x^{n-1} + x^n$.

Theorem 2: An LFSR with feedback polynomial $p(x)$ produces a code sequence which, when reversed, is equal to the sequence produced by the reciprocal feedback polynomial $p^r(x)$.

Proof: The recurrence equation for the LFSR with the reciprocal feedback polynomial $p^r(x)$ is $r(t+1) = H \cdot r(t)$, where

$$H = \begin{bmatrix} g_{n-1} & g_{n-2} & \cdots & g_1 & g_0 \\ 1 & 0 & \cdots & \cdots & 0 \\ 0 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 0 \end{bmatrix}.$$

It can be easily verified that for $g_0 = g_n = 1$ $G \cdot H = I_n$, the $n \times n$ identity matrix. The recurrence equations for k time steps are

$$q(t+k) = G^k \cdot q(t),$$

$$r(t+k) = H^k \cdot r(t) \Leftrightarrow r(t) = H^k \cdot r(t-k).$$

If we start with the same register contents in both cases, $r(t_0) = q(t_0)$ for some time t_0, using the above equations we get

$$\begin{aligned} q(t_0 + k) &= G^k \cdot q(t_0) = G^k \cdot r(t_0) \\ &= G^k \cdot H^k \cdot r(t_0 - k) \\ &= G^{k-1} \cdot G \cdot H \cdot H^{k-1} \cdot r(t_0 - k) \\ &= G^{k-1} \cdot I_n \cdot H^{k-1} \cdot r(t_0 - k) = \cdots \\ &= I_n \cdot r(t_0 - k) \\ &= r(t_0 - k). \end{aligned}$$

Thus the contents $q(t_0 + k)$ of an LFSR k time steps after t_0 are equal to the contents $r(t_0 - k)$ of the LFSR with reciprocal feedback polynomial k steps back in time. Since G and H are nonsingular matrices for $g_0 = g_n = 1$, the derivation is also valid for $k < 0$. ■

Because of this, the pattern matrix, P^r, obtained for $p^r(x)$ is equal to the transpose of the pattern matrix P for $p(x)$.

Some worthwhile theoretical results can be derived for Johnson counters (Fig. 11). Each pattern has a unique successor and predecessor in Johnson counter mode. The number of cycles is

$$Z(n) = \frac{1}{2n} \cdot \sum_{d|n} \Phi(d) \cdot 2^{n/d} - \frac{1}{2n} \cdot \sum_{2d|n} \Phi(2d) \cdot 2^{n/2d}$$

[21], where Φ denotes the Euler function. Similar to an LFSR, the behavior of a Johnson counter can be described by a recurrence equation $q(t + 1) = J \cdot q(t) \oplus e$, where

$$J = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & & \ddots & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & 0 \\ 0 & \cdots & & \ddots & 1 \\ 1 & 0 & \cdots & \cdots & 0 \end{bmatrix}$$

is a cyclic shift operator and $e = (0 \cdots 0 \quad 1)^t$ performs the complementation of the feedback signal.

Let $HD(i, j)$ be the Hamming distance of two codes i and j and let $w(q)$ denote the weight of a vector q, i.e., the number of 1's in this vector. The Hamming distance of two codes i and j is $HD(i, j) = w(i \oplus j)$, and rotating a vector does not change its weight, $w(J \cdot q) = w(q)$.

Definition 8: The cycle distance, $d_c(i, j)$, of two codes i and j belonging to the same Johnson counter cycle, c is the number of transitions necessary to get from i to j.

Theorem 3: The Hamming distance of all the patterns of a cycle c with the same cycle distance is identical, $d_c(i, j) = d_c(k, l) \Rightarrow HD(i, j) = HD(k, l)$.

Proof: From the recurrence equation we get $q(t + k) = J^k \cdot q(t) \oplus \Sigma_{i=0}^{k-1} J^i \cdot e$. Then we can derive the following result for the Hamming distance of two patterns

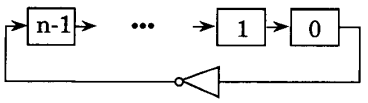


Fig. 11. Johnson counter.

with the cycle distance $d_c = k$:

$$\begin{aligned} &HD(q(t + k), q(t)) \\ &\quad = w(q(t + k) \oplus q(t)) \\ &\quad = w(J^k \cdot q(t) \oplus \sum_{i=0}^{k-1} J^i \cdot e \oplus q(t)) \\ &\quad = w\left((J^k \oplus I_n) \cdot q(t) \oplus \sum_{i=0}^{k-1} J^i \cdot e\right) \\ &\quad = w\left((J^k \oplus I_n) \cdot (J \cdot q(t - 1) \oplus e) \oplus \sum_{i=0}^{k-1} J^i \cdot e\right) \\ &\quad = w\left(J(J^k \oplus I_n) \cdot q(t - 1) \oplus \sum_{i=1}^{k} J^i \cdot e\right) \\ &\quad = w\left(J\left[(J^k \oplus I_n) \cdot q(t - 1) \oplus \sum_{i=0}^{k-1} J^i \cdot e\right]\right) \\ &\quad = w\left((J^k \oplus I_n) \cdot q(t - 1) \oplus \sum_{i=0}^{k-1} J^i \cdot e\right) \\ &\quad = HD(q(t + k - 1), q(t - 1)). \end{aligned}$$

By induction it follows that any two patterns with the same cycle distance $d_c = k$ have the same Hamming distance. ■

Example: For the case $n = 2$, there is only one cycle $00 \rightarrow 10 \rightarrow 11 \rightarrow 01 \rightarrow 00$ containing all the 2-bit patterns, $Z(2) = 1$. The cycle distance of code 00 and 10 is 1, the cycle distance of 00 and 11 is 2. All the codes with a cycle distance of 1 have a Hamming distance of 1, all the codes with a cycle distance of 2 have a Hamming distance of 2.

Corollary: There always exists a Johnson counter cycle in which all consecutive codes are adjacent. It includes the all-zero code $q(t_0) = 0$.

Proof:

$$\begin{aligned} &d_c(q(t + 1), q(t)) \\ &\quad = dc(q(t_0 + 1), q(t_0)) \; [= 1] \Rightarrow \\ &HD(q(t + 1), q(t)) \\ &\quad = HD(q(t_0 + 1), q(t_0)) \\ &\quad = w((J \cdot q(t_0) \oplus e) \oplus q(t_0)) = w(e) = 1. \end{aligned}$$ ■

The satisfaction of adjacency constraints in [2] mainly depends on this corollary. Because of Theorem 3, first determining the Johnson counter transitions and afterwards trying to embed these state chains in the Johnson counter cycles such that as many adjacency constraints as

possible are satisfied is not a very good strategy. The number of satisfiable adjacency constraints between the different states of a Johnson counter cycle is independent of the placement of states within this cycle; it depends only on the SSR transitions chosen and the Johnson counter cycle to which the states are assigned. In our algorithm we therefore choose the SSR transitions concurrently with satisfying the adjacency constraints.

IV. Analysis of the Solution

A. Testability

Consider the modified self-test structure of Fig. 3. The additional control signal Mode is included in the signature analysis process to detect faults in the logic producing this signal. If the Mode signal is produced correctly, there are two more reasons why the circuit might not work as required. Either the Mode line to the pattern generator control logic can be stuck at 0/1 or the control logic itself is faulty. Both faults can be detected by switching the pattern generator to system mode and applying two test patterns that produce signals satisfying the condition

$$\text{Mode} = 1, f_s(i, s) \neq SSR^+(s)$$

and

$$\text{Mode} = 0, f_s(i, s) \neq SSR^+(s).$$

For outputs with these properties a faulty next state is reached. The existence of test patterns causing such outputs is guaranteed by Theorem 4. To be able to distinguish the faulty state, $s^f(t+1)$, from the correct state, $s(t+1)$, there has to be some input pattern, i, for which the output, $o(t+1)$, or the next state, $s(t+2)$, is different. A sufficient condition is that $s(t+1)$ and $s^f(t+1)$ not be equivalent. But that is not strictly necessary: If s and s^f are equivalent and there exists an input i such that $f_s(i, s)$ and $f_s(i, s^f)$ have different state codes (i.e., the two states are not 1-equivalent), the fault also manifests itself in a wrong signature.

Theorem 4: The additional hardware of the modified self-test structure is testable for all single stuck-at faults if the FSM does not contain 1-equivalent states.

Proof: Assume that for one of the Mode values no input combination with $f_s(i, s) \neq \text{SSR}^+(s)$ exists. Then the next state of the FSM is always identical to the next state of the SSR in this Mode. If this happended for Mode = 1 (D-flip-flop mode), the FSM under consideration could be implemented by the pattern generation register without additional hardware. Then the D-flip-flop mode is redundant and would be eliminated in the synthesis process. If the problem occurs for Mode = 0 (SSR mode), being able to switch to SSR mode does not help in minimizing the combinational logic of the FSM. Mode signal and additional control logic are redundant and are eliminated in the synthesis process. ■

Test patterns with the required properties are contained in the normal pattern generator sequence with a high probability, so that no additional effort is needed to generate them besides switching to system mode for one clock cycle each.

Consider the modified scan path structure of Fig. 4. The control signal Mode is not directly observable and should be stored in an additional flip-flop in the scan chain. Thus, the combinational logic producing this signal can be checked. Apart from faults in the additional control logic, which can be treated as in the last section, the Johnson counter feedback and the multiplexer for SDI/$\overline{\text{SDO}}$ might not work properly. It is easy to test these additional gates for stuck-at faults. The output of the multiplexer is observable via the first flip-flop of the scan chain. The inputs are controllable by shifting in appropriate signals into the scan chain, because if it were not possible to set the inputs to values such that an input or output stuck-at fault became visible, the corresponding signal could be replaced by the pertinent constant, which in turn would reduce the amount of combinational logic needed. In other words: The additional gates, if not redundant, can be tested together with the rest of the combinational logic of the FSM.

B. Experimental Results

The complete logic synthesis process for scan path testable and self-testable FSM's is summarized in Fig. 12. Starting from a behavioral description, the coding constraints are generated either by symbolic minimization (for two-level combinational logic) or by an approach similar to [13] (for multilevel logic). They are analyzed and translated into the matrices A, D, S, C, and T. Matrix P is computed for a first default SSR belonging to the predetermined test strategy (scan path, self-test). These matrices are then used by the quadratic assignment algorithm. The resulting cost function value can be compared for different SSR's to determine the SSR with the lowest cost. Logic minimization (either two-level or multilevel) is then performed and a layout for the self-testable FSM is generated.

We performed various experiments by running FSM benchmark examples from the MCNC Workshop on Logic Synthesis [27] through a preliminary implementation of the algorithms presented. First, the machines were encoded and optimized disregarding the pattern generation capability of the state memory. We used the programs NOVA[4] (two-level logic [32]) and MUSTANG[5] (multilevel logic [13]) from the Octools distribution of the University of California at Berkeley [30]. We then used our approach for self-testable FSM's with LFSR's as pattern generators (KOALA[6]).

[4]More exactly, the encoding option "ihybrid" was used to obtain a fair comparison, because our algorithm performs disjoint minimization instead of symbolic minimization until now.

[5]Both, the fan-in-oriented algorithm and the fan-out-oriented algorithm were run. The best result was taken.

[6]Karlsruhe's optimized assignment for testable automata. The results in this table were obtained with the parameters $k_1 = k_3 = 1$ in cost function (1).

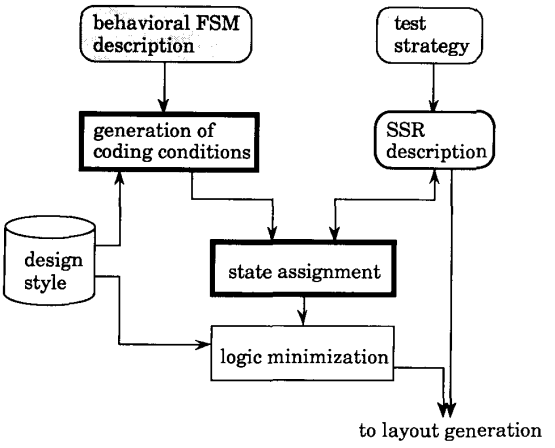


Fig. 12. Overview over the synthesis program for testable FSM's.

TABLE I
SUMMARY OF BENCHMARK RESULTS

Characteristics of Example		Two-Level Results (# Product Terms)			Multilevel Results (# Literals)	
Name	$i/o/s$	NOVA	KOALA	SSR Trans.	MUSTANG	KOALA
scf	27/56/121	146	136	30	822	773
planet	7/19/48	91	83	33	578	569
tbk	6/3/32	149	59	4	547	496
styr	9/10/30	94	93	6	594	512
dk16	2/3/27	59	57	4	270	241
donfile	2/1/24	29	28	5	160	74
ex1	9/19/20	48	44	7	280	253
s1	8/6/20	80	81	11	351	236
s1a	8/6/20	76	65	23	248	171
ex2	2/2/19	29	27	1	149	132
kirkman	12/6/16	64	54	2	176	146
bbsse	7/7/16	30	27	7	121	134
mark1	5/16/15	20	17	4	108	94
dk512	1/3/15	18	17	5	70	48
ex4	6/9/14	19	16	11	77	70
modulo12	1/1/12	13	9	8	35	29

The column $i/o/s$ gives the number of input variables, output variables, and states, respectively. The column "SSR Trans." gives the number of FSM transitions realized by using the SSR as pattern generator.

The combinational logic was optimized using an identical logic minimization procedure for the two approaches. This is particularly important for comparing multilevel logic results, since multilevel minimizers are generally interactive and it would be impossible to distinguish between improvements owing to a modified circuit structure/state assignment on the one hand and a more intensive logic minimization on the other. Therefore it does not make much sense to compare results for different multilevel state assignments unless a common sequence of minimization steps is used. We used the standard misII [30] script for multilevel minimization. Some of our results are summarized in Table 1. For the two-level implementations of the combinational logic the number of PLA product terms is given; for multilevel logic the number of literals is given. Compared with a conventional self-test solution, the area for implementing the state registers stays the same. CPU time for the state assignment was in the range of minutes on a SUN 3/60, which was usually less than in the time needed for logic minimization, particularly for a multilevel nonredundant implementation. It should be noted, however, that the number of states grows exponentially with the number of flip-flops, such that this approach is not suitable for circuits such as data paths.

For many examples, significant savings over a conventional self-test solution are possible; however, because of output incompatibilities, not all the transitions realized with the SSR can be saved. An excellent illustration of this effect can be found in Table I: the only difference between the examples $s1$ and $s1a$ is that for $s1a$ all the outputs are "0" (actually rendering this FSM useless except for benchmarking purposes), so output incompatibilities do not play a role, in contrast to $s1$. The problem is particularly important for two-level combinational logic;

it could be alleviated by separating the next state logic and the output logic. Sometimes there is a trade-off between satisfying adjacency constraints and utilizing SSR transitions. Since it is unlikely that there exists a cost function accurately modeling this effect without requiring exponential effort, it may happen that utilizing SSR transitions actually increases the amount of hardware needed (cf. *s1*). In this case experimenting with different weighting factors k_1, k_2, and k_3 in cost function (1) can help.

Results for scan designs have previously been presented [16]. In the case where scan path registers are used, for all combinational fault models 100% fault coverage can be achieved (cf. subsection IV-A) provided that the minimizer generates irredundant combinational logic. As with all scan path approaches, the length of an input sequence to detect a fault increases linearly with the number of flip-flops.

Since an SSR can only produce one next state for each present state code, the approach is especially suited to FSM's with sparse state transition graphs, i.e., few transitions per state, because then a large percentage of the transitions can be mapped to SSR transitions. Larger industrial controllers typically exhibit this structure (cf. [26]). For the smaller benchmark examples not listed in Table I, which have strongly connected state transition graphs, in general only slight improvements are achievable.

V. Conclusions

To be able to test nontrivial sequential circuits at a reasonable cost, a pertinent design approach has to be chosen. Integrating design for testability into the synthesis process, instead of modifying the circuit after its functional design is completed, offers several advantages. The amount of additional hardware can be reduced by utilizing the test circuitry in system mode. The barriers to designing testable circuits are lowered, because the test circuitry is no longer considered a postdesign overhead.

Circuits with scan paths or self-testable circuits can provide a satisfactory solution to the problem of testing sequential circuits. In this paper we have presented a synthesis method which utilizes the increased functionality of the storage elements of such circuits in the design of finite state machines. Both scan paths and self-test registers can be interpreted as "smart" state registers, capable of producing certain state transitions on their own. To make the best use of such registers we have proposed a state encoding strategy. An analytic formulation of the state assignment problem facilitates the use of several alternative coding conditions. It provides a unified framework for both two-level (PLA) and multilevel implementations. Choosing an appropriate feedback polynomial for LFSR pattern generators makes it possible to exploit a new minimization potential for self-testable FSM's. The approach was applied to a collection of benchmark FSM's. Compared with conventional FSM implementations with scan paths or self-test registers, significant savings can be achieved. In its current form the approach, however, is not applicable to the synthesis of sequential circuits with a large number of storage elements such as data paths.

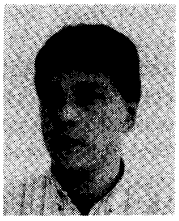
Acknowledgment

The authors are indebted to Prof. Dr. D. Schmid for his continuous support and encouragement. The help of Prof. Burkard and Dr. Rendl of the Technical University of Graz (Austria), who provided standard programs for quadratic assignment, is gratefully acknowledged. Finally the authors would like to thank the reviewers, particularly reviewer #1, whose detailed comments helped to improve the presentation.

References

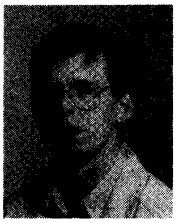
[1] V. D. Agrawal and K.-T. Cheng, "Test function specification in synthesis," in *Proc. 27th Design Automat. Conf.*, 1990, pp. 235–240.
[2] R. Amann, B. Eschermann, and U. G. Baitinger, "PLA based finite state machines using Johnson counters as state memories," in *Proc. IEEE Int. Conf. Computer Design*, 1988, pp. 267–270.
[3] D. Armstrong, "A programmed algorithm for assigning internal codes to sequential machines," *IRE Trans. Electronic Computers*, vol. EC-11, pp. 466–472, 1962.
[4] R. Bennetts, *Design of Testable Logic Circuits*. Reading, MA: Addison-Wesley, 1984.
[5] R. Burkhard and U. Derigs, *Assignment and Matching Problems*. New York: Springer, 1980.
[6] R. Burkhard and T. Bönniger, "A heuristic for quadratic Boolean programs with applications to quadratic assignment problems," *European J. Operational Res.*, vol. 13, pp. 374–386, 1983.
[7] R. Burkard, "Quadratic assignment problems," *European. J. Operational Res.*, vol. 15, pp. 283–289, 1984.
[8] R. Burkard and F. Rendl, "A thermodynamically motivated simulated procedure for combinatorial optimization problems," *European J. Operational Res.*, vol. 17, pp. 169–174, 1984.
[9] C. Chuang and A. Gupta, "The analysis of parallel BIST by the combined Markov chain (CMC) model," in *Proc. Int. Test Conf.*, 1989, pp. 337–343.
[10] W. Daehn, "Deterministische Testmustergenerierung für den eingebauten Selbsttest von integrierten Schaltungen," *NTG-Fachberichte 82*, pp. 16–19, 1983.
[11] G. DeMicheli, R. Brayton, and A. Sangiovanni-Vincentelli, "Optimal state assignment for finite state machines," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, no. 3, pp. 269–285, 1985.
[12] G. DeMicheli, "Symbolic design of combinational and sequential logic circuits implemented by two-level logic macros," *IEEE Trans. Computer-Aided Design*, vol. CAD-5, pp. 597–616, 1986.
[13] S. Devadas, H. Ma, R. Newton, and A. Sangiovanni-Vincentelli, "MUSTANG: State assignment of finite state machines targeting multilevel logic implementations," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 1290–1300, 1988.
[14] S. Devadas, H. Ma, R. Newton, and A. Sangiovanni-Vincentelli, "Irredundant sequential machines via optimal logic synthesis," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 8–18, 1990.
[15] E. Eichelberger and T. Williams, "A logic design structure for LSI testability," in *Proc. 14th Design Automat. Conf.*, 1977, pp. 462–468.
[16] B. Eschermann and H. Wunderlich, "A synthesis method to reduce scan design overhead," in *Proc. 1st European Design Automat. Conf.*, 1990, p. 671.
[17] B. Eschermann and H. Wunderlich, "Optimized synthesis of self-testable finite state machines," in *Dig. 20th Int. Symp. Fault-Tolerant Comput.*, 1990, pp. 390–397.
[18] M. Garey and D. Johnson, *Computers and Intractability*. New York: Freeman, 1979.
[19] R. Garfinkel and G. Nemhauser, *Integer Programming*. New York: Wiley, 1972.
[20] T. Gheewala, "CrossCheck: A cell based VLSI testability solution," in *Proc. 26th Design Automat. Conf.*, 1989, pp. 706–709.
[21] S. Golomb, *Shift Register Sequences*. Oakland, CA: Holden-Day, 1967.

[22] P. Hortensius *et al.*, "Cellular automata-based pseudorandom number generators for built-in self-test," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 842–859, 1989.
[23] J. Huertas and J. Quintana, "A new method for the efficient state-assignment of PLA-based sequential machines," *Dig. Int. Conf. Computer-Aided Design*, 1988, pp. 156–159.
[24] K. Kim, D. Ha, and J. Tront, "On using signature registers as pseudorandom pattern generators in built-in self-testing," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 919–928, Aug. 1988.
[25] B. Könemann, J. Mucha, and G. Zwiehoff, "Built-in logic block observation techniques," in *Proc. Int. Test Conf.*, 1979, pp. 37–41.
[26] R. Leveugle and G. Saucier, "Optimized synthesis of concurrently checked controllers," *IEEE Trans. Comput.*, vol. 39, pp. 419–425, 1990.
[27] R. Lisanke, *Logic Synthesis and Optimization Benchmarks*, Version 2.0; MCNC, 1988.
[28] E. McCluskey, *Logic Design Principles.* Englewood Cliffs, NJ: Prentice-Hall, 1986.
[29] A. Miczo, "The sequential ATPG: A theoretical limit," in *Proc. Int. Test Conf.*, 1983, pp. 143–147.
[30] Octtools Distribution 3.3, Electronics Research Laboratory, University of California, Berkeley, 1989.
[31] W. Peterson and E. Weldon, *Error-Correcting Codes.* Cambridge, MA: MIT Press, 1972.
[32] T. Villa and A. Sangiovanni-Vincentelli, "NOVA: State assignment of finite state machines for optimal two-level logic implementations," in *Proc. 26th Design Automat. Conf.*, 1989, pp. 327–332.
[33] L. Wang and E. McCluskey, "Built-in self-test for sequential machines," in *Proc. Int. Test Conf.*, 1987, pp. 334–341.
[34] W. Wolf, K. Keutzer, and J. Akella, "A kernel-finding state assignment algorithm for multi-level logic," in *Proc. 25th Design Automat. Conf.*, 1988, pp. 433–438.
[35] M. Williams and J. Angell, "Enhancing testability of large-scale integrated circuits," *IEEE Trans. Comput.*, vol. C-22, pp. 46–60, 1973.



Bernhard Eschermann was born in 1963. He received the Dipl.-Ing. degree in electrical engineering from the University of Karlsruhe, Germany, in 1987, the M.S. degree in electrical engineering and computer sciences from the University of California at Berkeley in 1988, and the Dr. rer. nat. (Ph. D.) degree in computer science from the University of Karlsruhe in 1991.

In 1987 he was a summer associate with Siemens AG, Munich, and McKinsey & Comp., Inc., Düsseldorf. From 1988 to 1991 he worked at the Institut für Rechnerentwurf und Fehlertoleranz, University of Karlsruhe, and at the Computer Science Research Center (FZI) in Karlsruhe, where he headed the Automation of Circuit Design group in 1991. Recently he joined the University of Siegen, Germany. His present research interests include logic and high-level synthesis, design for testability and fault-tolerant design.



Hans-Joachim Wunderlich (A'86) received the Dipl.-Math. degree in mathematics from the University of Freiburg, Germany, in 1981, and the Dr.rer.nat. (Ph.D.) degree in computer science from the University of Karlsruhe in 1986.

In 1982 he was a consultant at the Fraunhofer Institute of Industrial Engineering, Stuttgart, where he worked in the field of operations research. In 1983 he joined the Institut für Rechnerentwurf und Fehlertoleranz, University of Karlsruhe, where he headed a research group on the automation of circuit design and test since 1986. Currently Dr. Wunderlich is a full professor at the University of Siegen, Germany. His research interests include computer-aided design for testability, test generation, and digital simulation.