Optimized Synthesis of Self-Testable Finite State Machines

Bernhard Eschermann, Hans-Joachim Wunderlich

Universität Karlsruhe, Institut für Rechnerentwurf und Fehlertoleranz
Zirkel 2, 7500 Karlsruhe, F.R. Germany

ABSTRACT

In this paper a synthesis procedure for self-testable finite state machines is presented. Testability is already considered while transforming the behavioral description of the circuit into a structural description. To this end a novel state encoding algorithm as well as a modified self-test architecture are developed. Experimental results show that this approach leads to a significant reduction of hardware overhead.

Keywords: VLSI design validation, synthesis for testability, sequential circuits, built-in self-test.

1 INTRODUCTION

1.1 Motivation

Scan design methods and designs based on self-test registers, e.g. BILBO´s, are used to alleviate the problem of testing sequential circuits. Both approaches have in common that supplementary hardware only used for testing purposes is added *after* the functional design of the circuit is finished. It is tried to keep the hardware overhead low by integrating the additional components within the original circuit as much as possible.

The overhead can be reduced further, if testability is considered *during* the functional design of the circuit ("synthesis for testability"). The advantages of this approach are twofold: Firstly, the 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. Secondly, logic design decisions can be targeted towards obtaining circuits, which are easily testable "by construction". Several approaches to synthesis for testability have already been investigated and are reviewed in the sequel.

1.2 State of the Art

A finite state machine (FSM) can be tested functionally by a sequence of test patterns, which validates the existence of all the states and the correctness of all the state transitions. Such a *checking experiment* [Henn 64] can be simplified by adding extra inputs, outputs and/or state transitions to the FSM (e.g. [FuKi 74], [Prad 83], [HaMc 84], [SaDa 86], [WaMM 87]). Devadas et al. proposed a special state assignment strategy [DMNS 88a], that allows to reduce the test length by replacing the exhaustive checking with deterministic test patterns using a single stuck-at fault model. Later on the same authors published an approach for designing sequentially irredundant FSMs, which can also be tested for all single stuck-at faults [DMNS 90]. Cheng and Agrawal were able to reduce the number of flipflops in a partial scan path by using a special state assignment algorithm [ChAg 89].

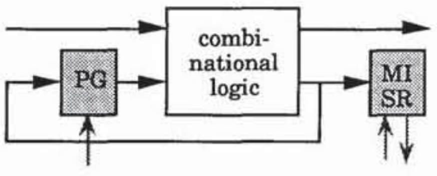
On-line checking of FSMs can be done by verifying the state code sequence with a parallel signature register. Leveugle and Saucier [LeSa 89] simplified this process by encoding the states in such a way, that the signature only depends on the state reached by the FSM and not on the exact transition sequence, with which the state was reached. Chuang and Gupta [ChGu 89] suggested to use a parallel self-test strategy for sequential circuits, in which the signatures collected in a multiple-input linear feedback shift register are also used as test patterns. The problem that certain states are not reachable in self-test mode is avoided by assigning the state codes in a clever way.

This paper focuses on the synthesis of self-testable FSM´s. In the following subsections we first present our target architecture, and illustrate its merits with a small example. The description of an FSM state assignment algorithm in section 2, which makes optimal use of the self-test hardware, forms the core of this paper. Self-testing circuits generally employ linear feedback shift registers for pattern generation. The impact of choosing a particular feedback polynomial on the state encoding is discussed. Testing the modified self-test structure is similar to testing the modified scan path structure in [EsWu 90] and therefore not elaborated in this paper.

1.3 Target Architecture

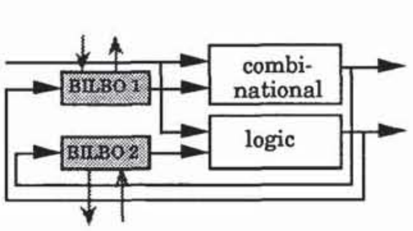
A circuit is made self-testable by adding a source of test patterns and a response analyzer to the circuit. One approach is to use a built-in logic block observer (BILBO) [KöMZ 79]. The system flipflops are redesigned, so that they can function as a linear feedback shift register (LFSR), which is able to both generate test patterns and to compact test responses into a signature. The mode of the BILBO is determined by external control signals. While LFSR´s generating pseudo-random patterns are widely used as pattern generators, many circuits are not amenable to random tests due to faults with low detection probabilities. By using pattern generators with optimized signal probabilities (e.g. GURT´s [Wund 87]), higher fault coverages can be obtained with reasonable test lengths.

Fig. 1 shows three possible self-test architectures[1]. In architecture a) [Benn 84], [WaMc 87] the system flipflops can be configured as autonomous LFSR, which generates test patterns for the combinational logic (PG: pattern generator). The responses are compacted in a separate multiple-input LFSR (MISR). If the signature analyzer is implemented as a BILBO, it can also be used to generate test patterns for other circuits on the chip. Architecture b) [Benn 84] is applicable, if the logic can be partitioned into two parts, such that the state signals generated by one part are only used as inputs in the other part. In a first test session BILBO 1 is employed as pattern generator, BILBO 2 as signature analyzer, in a second session the functions are interchanged. Unfortunately, for FSM´s such a partitioning is rarely possible. Architecture c) [BaMc 82], [BeMa 84] saves a separate pattern generator, it uses the outputs of the signature analyzer as test patterns instead. The drawback is that the characteristics of these test patterns may be undesirable [ChGu 89]. In the sequel we will restrict ourselves to architecture a), since it is the only one usable for arbitrary circuits and able to guarantee a high fault coverage.

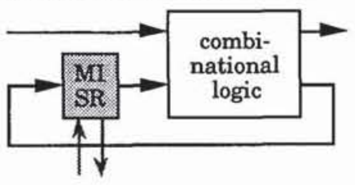


a) Separate pattern generation and signature registers

[1] Pattern generation and response analysis for the primary inputs and outputs are not shown, since they are identical for all the self-test architectures.



b) Combinational logic partitioned



c) Signature register used to generate test patterns

Fig. 1. Architectures of self-testable sequential circuits

The test pattern generation register PG is characterized by its ability to cycle through certain states on its own. 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. If the next state code is produced by the PG 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. 2 illustrates a possible realization of this idea. An additional output signal L/S determines, whether the state machine flipflops behave like ordinary D-flipflops or function in pattern generation mode. In this mode the state register generates the next state on its own, the next state signals asserted by the combinational logic can be set to arbitrary values.

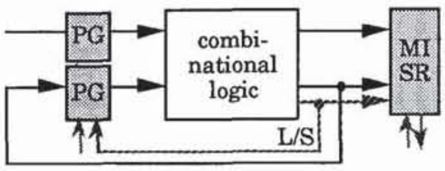


Fig. 2. Modified self-test architecture

The main problem is to find a state assignment reducing the combinational logic of this modified self-test structure. Conventional state assignment algorithms are certainly not optimal, 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 PG register, 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 algorithms have to be regarded concurrently during state assignment.

1.4 Illustrative Example

A variety of state assignment algorithms has been proposed to minimize the area of the combinational logic needed to implement an FSM. An extensive survey can be found in [Esch 90]. Most of the newer algorithms for two-level combinational logic follow a strategy proposed by DeMicheli et al. [DeMi 86]: First, logic minimization is applied to a symbolic representation of the FSM. The effect of this *symbolic minimization*[2] is to group together the states that are mapped by some input into the same next state and output. If these groups of states are encoded in a minimal subspace of Boolean r-space (r: encoding length), the number of product terms of a two-level implementation is reduced. It is the task of the subsequent encoding step to satisfy these *coding constraints*.

To illustrate this process, the FSM of Fig. 3a is used. If each symbolic state is replaced by a binary code word, in which each state is represented by one bit position (Fig. 3b), and this binary representation is minimized with a standard minimization algorithm, the result in Fig. 3c is obtained. This *minimized symbolic cover* contains the coding constraints that have to be satisfied later on.

state	inp.	n.state	out.
A	00	A	1
	01	B	0
	1-	B	1
B	00	A	0
	01	C	0
	1-	B	1
C	00	A	1
	01	A	0
	1-	B	1

s	i	ns	o
100	00	100	1
100	01	010	0
100	1-	010	1
010	00	100	0
010	01	001	0
010	1-	010	1
001	00	100	1
001	01	100	0
001	1-	010	1

s	i	ns	o
101	00	100	1
111	1-	010	1
100	01	010	0
010	00	100	0
010	01	001	0
001	01	100	0

Fig. 3.
a) FSM description b) symbol. descr. c) coding constraints

The first *symbolic implicant* in Fig. 3c contains the following information: If states A and C are given adjacent codes (Hamming distance equal to 1), the binary implicants can be merged in the same way as their symbolic counterparts are. The constraint in line 2 is trivially satisfied by any encoding. With the state assignment of Fig. 4a, the two-level logic of Fig. 4b is obtained. The example illustrates that conventional state assignment methods only use those implicants of the minimized symbolic cover, in which groups of present state symbols appear.

state A:	01
state B:	10
state C:	11

s	i	ns	o
-1	00	01	1
--	1-	10	1
01	01	10	0
10	00	01	0
10	01	11	0
11	01	01	0

Fig. 4. a) state assignment b) PLA personalization

With the self-test architecture of Fig. 2, the next state need not necessarily be produced by the combinational logic of the FSM. The self-test hardware can be utilized to take over this function if the next state code of the FSM is equal to the next pattern generated by the PG register. For the state encoding of Fig. 4a and the LFSR feedback polynomial $1+x+x^2$, the PG register built from this LFSR is able to generate the state transitions $A \rightarrow B$, $B \rightarrow C$, and $C \rightarrow A$. The symbolic implicants in line 3, 5 and 6 of Fig. 3c, not usable for minimization in conventional state assignment algorithms, can now be completely saved, if the signal L/S = 0 (cf. Fig. 2) makes the LFSR operate in pattern generation mode, and L/S = 1 makes it load the next state signals from the combinational logic. The result is shown in Fig. 5.

s	i	ns	o	L/S
101	00	100	1	1
111	1-	010	1	1
100	01	---	0	0
010	00	100	0	1
010	01	---	0	0
001	01	---	0	0

s	i	ns	o	L/S
-1	00	01	1	1
--	1-	10	1	1
10	00	01	0	1

Fig. 5. a) min. symb. description b) PLA personalization

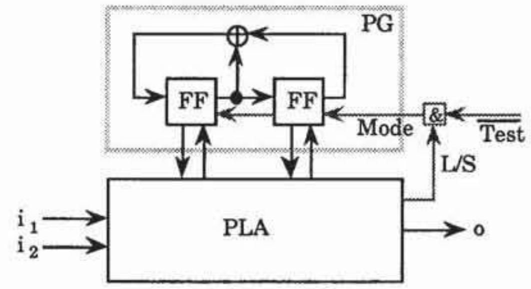


Fig. 5. c) Resulting circuit structure

[2] To simplify the presentation, in this paper actually only "disjoint minimization" [DeMi 86] is used.

2 SOLUTION OF THE STATE ASSIGNMENT PROBLEM

Our approach to state encoding is based on an analytical formulation of the state assignment problem. The main advantage of this approach is that a global optimization is performed. The heuristic algorithms usually employed for satisfying the coding constraints all use some kind of a greedy ordering, which is likely to lead to a good solution in most cases, but sometimes also fails in achieving this objective. Additionally, it is very easy to accommodate the additional PG register constraints in the analytic formulation. Beforehand, some matrices have to be introduced. Let s be the number of states of the FSM, and r the number of bits used for the encoding of these states.

Definition 1: The *adjacency matrix* A of a minimized symbolic cover is an $s \times s$ matrix of non-negative integer entries a_{ik}. For $i \neq k$ the value a_{ik} corresponds to the number of lines, in which state i and k both occur in a coding constraint; the diagonal entries a_{ii} are set to 0.

Definition 2: The *distance matrix* D is a $2^r \times 2^r$ matrix with non-negative integer entries d_{jl}, where d_{jl} is equal to the Hamming distance of codes j and l minus 1 for $j \neq l$ and $d_{jj} = 0$.

Similar adjacency and Hamming distance values have already been used in many state assignment algorithms (see [Arms 62] for one of the first references). To describe the effect of the PG register, two other matrices become necessary.

Definition 3: The *successor matrix* S of a minimized symbolic cover is an $s \times s$ matrix of non-negative integer entries s_{ik}, where s_{ik} is the number of symbolic implicants with an isolated present state i and next state k.

Definition 4: The *cycle matrix* C of an r-bit pattern generator is a $2^r \times 2^r$ matrix with entries $c_{jl} \in \{0,1\}$, where $c_{jl} = 0$, if code l is the successor of code j in the sequence generated by the pattern generator, and $c_{jl} = 1$ otherwise.

Matrix A collects the information about pairs of *states* from the minimized symbolic cover, matrix S deals with the successor relationships obtainable from the symbolic implicants not specifying any state groups. Matrices D and C contain the information about pairs of *codes* necessary for the encoding process[3].

For the example of section 1.4 and the minimized symbolic cover in Fig. 3c, the matrices in Fig. 6 are obtained. In matrices A and S the first row/column corresponds to state "A", the second to state "B" and the third to state "C". The order of entries in matrices D and C corresponds to the codes 00, 01, 10 and 11 in that sequence. For matrix C, again the LFSR with a feedback polynomial $1+x+x^2$ was used. Note that A and D are symmetric matrices, whereas S and C are not.

$$A = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{bmatrix} \qquad D = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

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

Fig. 6. Example matrices A, S, D and C

To formulate the state assignment problem analytically, a cost function is needed that captures all the necessary ingredients of a good assignment. In a good assignment, all the states appearing together in many symbolic implicants are assigned to codes with small Hamming distances, preferably to adjacent codes. Let X be an assignment matrix with Boolean entries $x_{ij} \in \{0,1\}$, $x_{ij} = 1$ if code j is assigned to state i and 0 otherwise. Then a partial cost of

$$\alpha\,(i,j,k,l) := \frac{1}{2}\, a_{ik}\, d_{jl}\, x_{ij}\, x_{kl}$$

is incurred (cf. Fig. 7) by the assignment of a pair of non-adjacent codes j and l ($d_{jl} = d_{lj} > 0$) to states appearing together in symbolic implicants ($a_{ik} = a_{ki} > 0$). The factor $\frac{1}{2}$ takes the symmetry of matrices A and D into account.

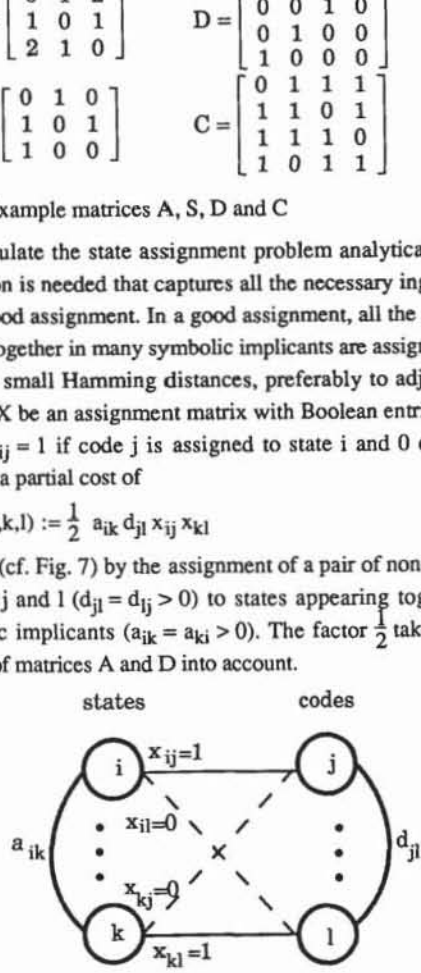


Fig. 7. Non-adjacency cost α (i,j,k,l)

[3] Two more matrices are necessary, if "covering relations" from a true *symbolic* minimization are to be considered.

If transitions not minimizable with the help of adjacency constraints ($s_{ik} > 0$) cannot be realized with the help of the PG register ($c_{jl} = 1$) by assigning code j to state i ($x_{ij} = 1$) and code l to state k ($x_{kl} = 1$), this corresponds to a cost of

$$\tau\,(i,j,k,l) := s_{ik}\, c_{jl}\, x_{ij}\, x_{kl}$$

(cf. Fig. 8).

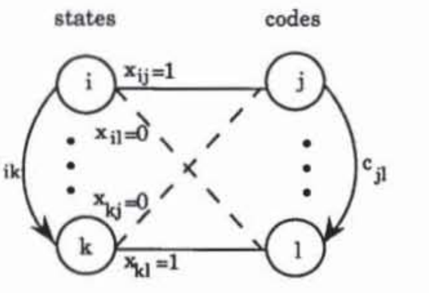


Fig. 8. Non-PG transition cost τ (i,j,k,l)

The problem of finding an appropriate assignment matrix X, such that state pairs appearing together in symbolic implicants are encoded with adjacent codes and that the remaining state transitions are preferably produced by switching to pattern generation mode, then can be formulated as a 0-1 integer program:

$$(1)\quad \min(X){:}\ \sum_{i,j,k,l}(\alpha(i,j,k,l)+\tau(i,j,k,l)) = \sum_{i,j,k,l}\left(\tfrac{1}{2}a_{ik}d_{jl}+s_{ik}c_{jl}\right)\ x_{ij}x_{kl}$$

$$(2)\quad \text{w.r.t.}\quad \sum_{i} x_{ij} \le 1\quad \forall j$$

$$(3)\quad \sum_{j} x_{ij} = 1\quad \forall i$$

$$(4)\quad x_{ij} \in \{0,1\}\quad \forall\, i,j$$

In this formulation the complete cost of the assignment was obtained by adding the cost of transitions not reducible by logic minimization to the cost of transitions not realizable by switching to pattern generation mode and summing these cost values over all states and codes. Adjacency and successor relationships can also be given different weights by minimizing the sum of cost values $k_1\,\alpha\,(i,j,k,l) + k_2\,\tau\,(i,j,k,l)$, with arbitrary constants $k_1, k_2 \ge 0$. The i + j linear constraints (2) and (3) ensure that each state is assigned exactly one code and that each code is assigned to at most one state.

Problem (1) - (4) is a well-known combinatorial optimization problem, the *quadratic assignment problem* [GaNe 72]. Although it was proven to be NP-complete [GaJo 79], because of its relevance for many applications a lot of effort was spent to develop feasible solution methods (see [Burk 84] for an overview). Exact algorithms using implicit enumeration techniques can cope with problems up to 15-20 states. For larger problems, we chose to use a heuristic approach similar to [BuRe 84]. The algorithm does not rely on a heuristic ordering of coding constraints or codes to be encoded; the cost function to be minimized always provides a global view on the complete encoding.

For the example of section 3, the minimal cost obtained with the assignment in Fig. 4a

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

is

$$\sum_{i,j,k,l}\left(\tfrac{1}{2}a_{ik}\,d_{jl} + s_{lk}\,c_{jl}\right)\,x_{ij}\,x_{kl} \;=\; 2$$

with a cost of 1 for non-adjacencies (the Hamming distance between the codes for "A" and "B" is 2) and a cost of 1 for other transitions not realizable with the PG register (B $\rightarrow$ A).

The proposed state encoding method can be used for multi-level combinational circuits with only minor modifications. In [DMNS 88b] a first state assignment algorithm targeted towards multi-level implementations was proposed. It is based on minimizing a cost function with the same mathematical structure as the non-adjacency part of the cost function used above. The only difference is that the values a_{ik} are not derived by symbolic minimization but by estimating the number of common cubes that can be extracted from the resulting combinational network.

3 CHOICE OF FEEDBACK POLYNOMIAL

Until now the characteristics of the pattern generator were only needed to determine the cycle matrix C. Therefore any test pattern generator describable by such a matrix can be accommodated. In practice the choice will be made based on the necessary fault coverage and the hardware overhead involved. In the sequel we will consider the important case of LFSR´s with primitive feedback polynomials (cf. [Pete 72]).

An LFSR with feedback polynomial $p_i(x)$ produces a code sequence which, when reversed, is equal to the sequence produced by the reciprocal feedback polynomial $p_i'(x)$. Hence, the cycle matrix C_i' obtained for $p_i'(x)$ is equal to the transposed cycle matrix C_i for $p_i(x)$. If the indices of the codes are permuted in such a way, that C_i' becomes equal to C_i, the corresponding permutations in the distance matrix D will reverse all the lines of D. Accordingly, the state assignments obtained with different polynomials vary, so do the complexities of the resulting combinational logic. To obtain the optimal solution requires finding out, which LFSR structure is best suited to the FSM under consideration. Choosing the feedback polynomial of the LFSR by minimizing the cost function of section 4 over all C_i thus provides an additional degree of freedom.

In order to minimize LFSR area, minimum weight polynomials are often preferred. An interesting fact is that it does not matter, whether a standard implementation or a modular implementation [McCl 86] of the LFSR is chosen, as long as a polynomial $p(x) = 1 + x^i + x^n$, $1 \leq i < n$, with only one 2-input XOR-gate in the feedback structure of the LFSR is used. The reason is that in this case both implementations only differ in the sequence of flipflops (see Fig. 9), which is irrelevant to state assignment and logic minimization. Matrix D remains unchanged, because the permutation of coding columns does not influence the Hamming distances of codes.

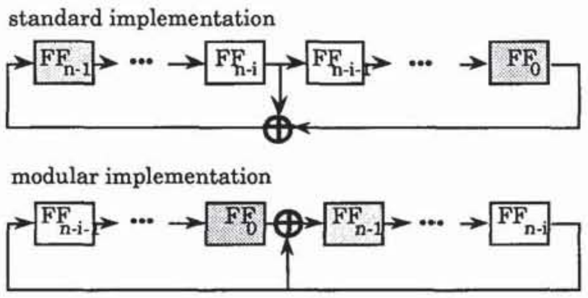


Fig. 9. Standard and modular LFSR with $p(x) = 1 + x^i + x^n$

4 RESULTS

The complete logic synthesis process for self-testable FSM's is summarized in Fig. 10. Starting from a behavioral description, the coding constraints are generated either by symbolic minimization (for 2-level combinational logic) or with the approach presented in [DMNS 88b] (for multi-level logic). They are analyzed and translated into the matrices A, D and S. The default encoding length is $r_0 = \lceil \log_2 s \rceil$, but any other number $r > r_0$ can be specified as well. Matrix C is computed for a first candidate PG register. With these matrices the algorithm for quadratic assignment is started. The resulting cost function value can be compared for different PG registers. The PG register with the lowest cost is chosen. Afterwards logic minimization (either 2-level or multi-level) can be performed and a layout for the self-testable FSM can be generated.

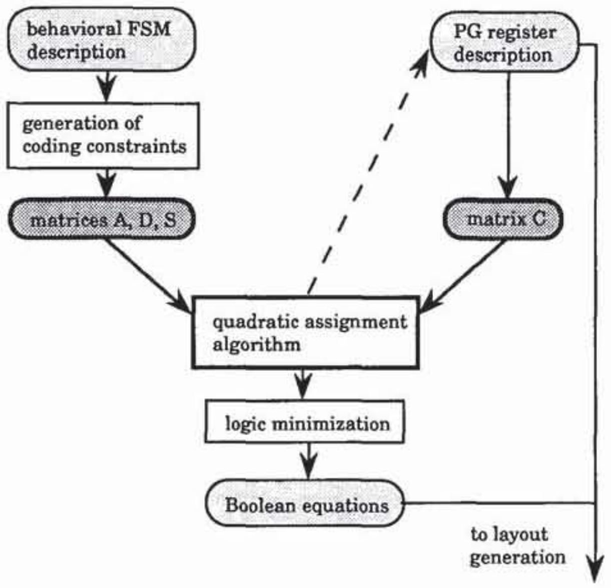


Fig. 10. Synthesis process for self-testable FSM's

We performed various experiments by running FSM benchmark examples from the MCNC Workshop on Logic Synthesis [MCNC 88] 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] (2-level logic, [ViSa 89]) and MUSTANG[5] (multi-level logic, [DMNS 88b]) from the Octtools distribution of the University of California, Berkeley [OCT 89] for this purpose. Afterwards, we used our combined synthesis for testability (SfT) approach.

The combinational logic was optimized using identical logic minimization procedures for the conventional and the SfT approach. This is particularly important for comparing multi-level logic results, since multi-level minimizers are generally interactive and it would be impossible to distinguish between improvements due to a modified circuit structure / state assignment on the one hand and a more intensive logic minimization on the other hand. Some results are summarized in Table 1. The column "i/o/s" gives the number of input variables, output variables, and states, respectively. For the two-level implementations of the combinational logic the number of PLA product terms, for multi-level logic the number of literals is given. CPU

[4] More exactly, the encoding option "ihybrid" was used.

[5] Both, the fanin-oriented algorithm and the fanout-oriented algorithm were run. The best result was taken.

times for the state assignment were in the range of minutes on a 3 MIPS machine.

example		2-level (# pt)		m-level (# lit)	
name	i/o/s	NOVA	SfT	MUST	SfT
scf	27/56/121	146	136	822	773
planet	7/19/48	91	83	578	569
tbk	6/3/32	149	59	547	496
styr	9/10/30	94	93	594	543
dk16	2/3/27	59	57	270	241
donfile	2/1/24	29	33	160	74
ex1	9/19/20	48	44	280	253
s1	8/6/20	80	81	351	236
s1a	8/6/20	76	65	248	171
ex2	2/2/19	29	27	149	132
kirkman	12/6/16	64	54	176	146
bbsse	7/7/16	30	27	121	134
mark1	5/16/15	20	17	108	94
dk512	1/3/15	18	17	70	48
ex4	6/9/14	19	16	77	70
modulo12	1/1/12	13	9	35	29

Table 1. Summary of benchmark results

For many examples significant savings are possible, although, because of output incompatibilities, not all the transitions realized with the PG register can indeed be saved. An excellent illustration of this effect can be found in Table 1: the only difference between the examples *s1* and *s1a* is that for *s1a* all the outputs are "0", so output incompatibilities do not play a role, in contrast to *s1*. The problem is particularly important for 2-level combinational logic; it could be alleviated by separating the next state logic and the output logic. Sometimes there is a tradeoff between satisfying adjacency constraints and utilizing PG transitions. As it is unlikely that there exists a cost function accurately modeling this effect without requiring exponential effort (logic minimization is NP-complete [GaJo 79]), it may happen that utilizing PG transitions actually increases the amount of hardware needed (cf. *donfile* and *s1*). In this case experimenting with different weighting factors k_1 and k_2 in cost function (1) can help.

5 CONCLUSIONS

Self-testable circuits can provide a satisfactory solution to the problem of testing digital systems, if they are realizable with a reasonable amount of extra hardware. In this paper we presented an approach to reduce the overhead for self-testable control units by finding a useful application of the self-test hardware in system mode. This way the combinational logic for implementing the system functionality can be simplified. Self-test thus may become more competitive with other design for testability methods for area-critical applications.

ACKNOWLEDGEMENTS

The help of Prof. Burkard and Dr. Rendl from the Technical University of Graz (Austria), who provided us with some standard programs for quadratic assignment, is gratefully acknowledged.

LITERATURE

Aman 87	R. Amann: Algorithmic Design Methods for Combined PLA/ROM Controllers (in German); Fortschrittberichte VDI, nr. 68, 1987.
Arms 62	D. B. Armstrong: A Programmed Algorithm for Assigning Internal Codes to Sequential Machines; IRE Trans. on Electronic Computers, vol. EC-11, pp. 466-472, 1962.
BaMc 82	P. H. Bardell, W. H. McAnney: Self-Testing of Multichip Logic Modules; Proc. IEEE Int. Test Conference, pp. 200-204, 1982
BeMa 84	F. P. Beucler, M. J. Manner: HILDO: The Highly Integrated Logic Device Observer; VLSI Design, pp. 88-96, June 1984.
Benn 84	R. G. Bennetts: Design of Testable Logic Circuits; Addison-Wesley, 1984.
BuRe 84	R. E. Burkard, F. Rendl: A Thermodynamically Motivated Simulated Procedure for Combinatorial Optimization Problems; European Journal of Operational Research, vol. 17, pp. 169-174, 1984.
Burk 84	R. E. Burkard: Quadratic Assignment Problems; European Journal of Operational Research, vol. 15, pp. 283-289, 1984.
ChAg 89	K.-T. Cheng, Vishwani D. Agrawal: Design of Sequential Machines for Efficient Test Generation; Dig. Tech. Papers International Conference on Computer-Aided Design, pp. 358-361, 1989.
ChGu 89	C. C. Chuang, A. K. Gupta: The Analysis of Parallel BIST by the Combined Markov Chain (CMC) Model; Proc. IEEE Int. Test Conference, pp. 337-343, 1989.
DeBS 85	G. DeMicheli, R. K. Brayton, A. Sangiovanni-Vincentelli: Optimal State Assignment for Finite State Machines; IEEE Trans. on Computer-Aided Design, vol. CAD-4, no. 3, pp. 269-285, 1985.
DeMi 86	G. DeMicheli: Symbolic Design of Combinational and Sequential Logic Circuits Implemented by Two-Level Logic Macros; IEEE Trans. on Computer-Aided Design, vol. CAD-5, pp. 597-616, 1986.

DMNS 88a S. Devadas, H.-K. Ma, A. R. Newton, A. Sangiovanni-Vincentelli: Synthesis and Optimization Procedures for Fully and Easily Testable Sequential Machines; Proc. Int. Test Conference, pp. 621-630, 1988.

DMNS 88b S. Devadas, H.-K. Ma, A. R. Newton, A. Sangiovanni-Vincentelli: MUSTANG: State Assignment of Finite State Machines Targeting Multilevel Logic Implementations; IEEE Trans. on Computer-Aided Design, vol. CAD-7, pp. 1290-1300, 1988.

DMNS 90 S. Devadas, H. T. Ma, A. R. Newton, A. Sangiovanni-Vincentelli: Irredundant Sequential Machines Via Optimal Logic Synthesis; IEEE Trans. on Computer-Aided Design, vol. CAD-9, pp. 8-18, 1990.

Esch 90 B. Eschermann: State Assignment Methods for Synchronous Sequential Circuits; Progress in Computer Aided VLSI Design, G. Zobrist (ed.), Ablex, Norwood, 1990.

EsWu 90 B. Eschermann, H.-J. Wunderlich: A Synthesis Method to Reduce Scan Design Overhead; First European Design Automation Conference, 1990.

FuKi 74 H. Fujiwara, K. Kinoshita: Design of Diagnosable Sequential Machines Utilizing Extra Outputs; IEEE Trans. on Computers, vol. C-23, pp. 138-145, 1974.

GaJo 79 M. Garey, D. Johnson: Computers and Intractability; Freeman, New York 1979.

GaNe 72 R. Garfinkel, G. Nemhauser: Integer Programming; Wiley, New York, 1972.

HaMc 84 S. Hassan, E. McCluskey: Pseudo-Exhaustive Testing of Sequential Machines Using Signature Analysis; Proc. Int. Test Conference, pp. 320-326, 1984.

Henn 64 F. C. Hennie: Fault Detecting Experiments for Sequential Circuits; Proc. 5th Annual Symp. Switching Circuit Theory and Logical Design, pp. 95-110, 1964.

KöMZ 79 B. Könemann, J. Mucha, G. Zwiehoff: Built-In Logic Block Observation Techniques; Proc. IEEE Int. Test Conference, pp. 37-41, 1979

LeSa 89 R. Leveugle, G. Saucier: Optimized Synthesis of Dedicated Controllers with Concurrent Checking Capabilities; Proc. Int. Test Conference, pp. 355-363, 1989.

McCl 86 E. J. McCluskey: Logic Design Principles with Emphasis on Testable Semicustom Circuits; Prentice-Hall, Englewood Cliffs, 1986.

MCNC 88 R. Lisanke: Logic Synthesis and Optimization Benchmarks, Version 2.0; Microelectronics Center of North Carolina, 1988.

OCT 89 Octtools Distribution 3.3, Electronics Research Laboratory, University of California, Berkeley, 1989.

Pete 72 W. W. Peterson, E. J. Weldon: Error-Correcting Codes; MIT Press, Cambridge, 1972.

Prad 83 D. K. Pradhan: Sequential Network Design Using Extra Inputs for Fault Detection; IEEE Trans. on Computers, vol. C-32, pp. 319-323, 1983.

SaDa 86 K. K. Saluja, R. Dandapani: A Built-In Self-Testable Design Method for Sequential Circuits; Proc. 16th Int. Symp. Fault Tolerant Computing, pp. 312-317, 1986.

ViSa 89 T. Villa, A. Sangiovanni-Vincentelli: NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementations; Proc. 26th Design Automation Conference, pp. 327-332, 1989.

WaMc 87 L.-T. Wang, E. J. McCluskey: Built-in Self-Test for Sequential Machines; Proc. Int. Test Conference, pp. 334-341, 1987.

WaMM 87 L.-T. Wang, E. J. McCluskey, S. Mourad: Shift Register Testing of Sequential Machines; Proc. 17th Int. Symp. Fault-Tolerant Computing, pp. 66-71, 1987.

Wund 87 H.-J. Wunderlich: Self-Test Using Unequiprobable Random Patterns; Proc. 17th Int. Symp. Fault-Tolerant Computing, pp. 258-263, 1987.