Proceedings EDAC/ETC/EuroAsic, Paris 1994

Synthesis of Self-Testable Controllers

Sybille Hellebrand, Hans-Joachim Wunderlich
Institute of Computer Structures
University of Siegen
Germany

Abstract

The paper presents a synthesis approach for pipeline-like controller structures. These structures allow to implement a built-in self-test in two sessions without any extra test registers. Hence the additional delay imposed by the test circuitry is reduced, the fault coverage is increased, and in many cases the overall area is minimal, too. The self-testable structure for a given finite state machine specification is derived from an appropriate realization of the machine. A theorem is proven that such realizations can be constructed by means of partition pairs. An algorithm to determine optimal realizations is developed and benchmark experiments are presented to demonstrate the applicability of the presented approach .

1 Introduction

The application of microelectronic systems in safety-critical areas, e.g. in avionics or medicine, demands extremely high quality standards, and thus refined testing techniques. The problem of implementing efficient tests providing a complete or very high fault coverage is particularly difficult for controllers because of their irregular structure and the reduced observability and controllability of internal states. Conventionally the circuit structure for a controller is synthesized from a finite state machine specification performing state coding and logic minimization [5, 6, 12, 23, 22]. But even if advanced synthesis techniques are used to generate sequentially irredundant controllers, the necessary test sequences might be prohibitively long [11, 2, 21]. To overcome this problem either additional test functions have to be considered during synthesis or testability features such as built-in self-test (BIST) have to be added to the synthesized structure [7, 9, 1]. With respect to safety-critical applications BIST is of special importance, since the capabilities for test pattern generation and test response evaluation on chip can also be used for periodic maintenance tests.

This work was supported by EP 7107 ARCHIMEDES.

Usually the BIST is implemented by so-called multifunctional test registers like the well-known BILBO which are able to work as a system register, to generate test patterns and to compress the test responses by signature analysis. Such test registers have been developed for random, deterministic, pseudo-exhaustive and weighted random patterns [19, 10, 4, 25, 17]. However, the circuit structure obtained from conventional synthesis procedures as shown in figure 1 is not a priori compatible with BIST, as during self-testing the register should generate patterns and evaluate test responses concurrently.

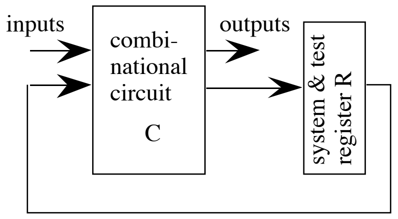

Figure 1: Result of conventional synthesis procedure.

This kind of parallel self-test, where the signatures are used as test patterns, is only feasible in a few cases, but in general the required properties of the test patterns cannot be guaranteed [18, 13]. In most cases the signatures are not exhaustive, (weighted) random or even deterministic, and an additional test register is usually required (figure 2).

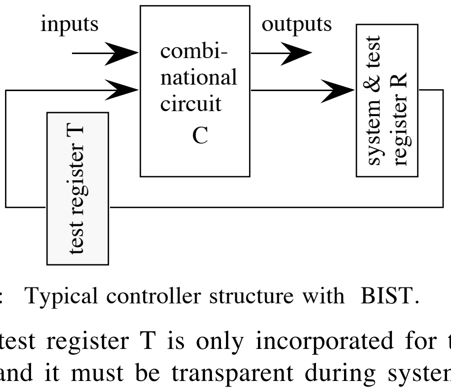

Figure 2: Typical controller structure with BIST.

The test register T is only incorporated for test purposes, and it must be transparent during system mode. This is a common self-test architecture, for variations see also [1]. But all these configurations have some serious drawbacks:

1) The number of flipflops must be doubled.
2) In system mode the test register T must be transparent or bypassed. This prolongs the critical path and may slow down the system speed of the controller.
3) There are faults on the feedback lines from R to the inputs of C which are not detected, as these lines are not completely exercised during self-test. This holds, even if the connections between R and T are tested in an additional step.

The last two disadvantages can be circumvented by doubling not only the flipflops but also the combinational circuitry (see figure 3). If both copies of R are initialized to the same values, the structure of figure 3 implements the same machine as the structure of figure 1. None of the registers needs to be transparent during system mode and there is no additional delay imposed this way. The self-test can be performed in two sessions by alternatively using one of the registers for pattern generation and the other for signature analysis. Moreover, as there is no transparency mode or bypassing a complete fault coverage is possible.

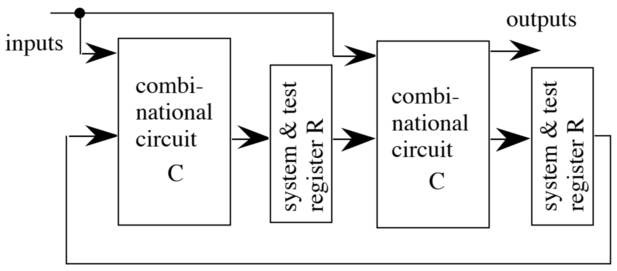

Figure 3: Self-testable controller structure with doubled system register and combinational circuitry.

The main drawback of the solution shown in figure 3 is the high hardware overhead. In this paper a synthesis technique is presented which reduces this overhead by implementing two different combinational networks C_1 and C_2 and two different registers R_1 and R_2 (see figure 4).

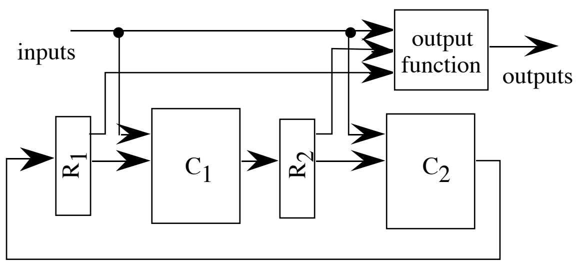

Figure 4: Optimized self-testable controller structure.

In general the registers R and T of figure 1 to 3 are wider than the registers R_1 and R_2, hence the structure of figure 4 needs less flipflops than the self-testable structures of figure 2 and 3. Furthermore, also the combinational circuits C_1 and C_2 are smaller than the original circuit. It will be shown that in many cases not only fault-coverage and speed are increased, but also the hardware overhead for integrating a self-test is reduced. In addition to that this architecture is also compatible with synthesis techniques which use autonomous transitions of the test register as system transitions [14].

It is important to note that this structure is different from structures provided by decomposition techniques where the resulting submachines contain internal feedback loops [16, 3, 15]. In contrast to known approaches trying to reduce dependencies between state variables by appropriate state coding the presented work addresses the problem already at the finite state machine level [24, 8]. Based on algebraic structure theory for a given finite state machine specification a realization is constructed which supports a self-testable structure as shown in figure 4. State coding and logic minimization are then applied to this realization.

The rest of the paper is organized as follows: In section 2 the notion of finite state machines supporting self-testable structures is introduced and the problem of synthesizing optimal self-testable controllers is stated as an optimization problem at the finite state machine level. Subsequently in section 3 the existence of suitable finite state machine realizations is related to the existence of partition pairs with additional properties, and an algorithm is developed which solves the problem stated in section 2. Section 4 provides experimental results. Conclusions and comments on future work are given in section 5.

2 Basic definitions and problem statement

In this section the problem of synthesizing self-testable controllers is reduced to an optimization problem at the finite state machine level. To allow a precise problem statement first some basic definitions are summarized and the notion of finite state machines supporting self-testable structures is introduced. Throughout this work it is assumed that controllers are fully specified as mealy-type finite state machines.

Definition 1: A mealy-type finite state machine (fsm) is a 5-tupel $M = (S, I, O, \delta, \lambda)$, where S is a finite non-empty set of states, I a finite non-empty set of inputs and O a finite non-empty set of outputs. $\delta: S \times I \rightarrow S$ is called the transition (or next state) function and $\lambda: S \times I \rightarrow O$ the output function of M.

The functions δ and λ are represented by a state transition table. An entry in row s and column i represents the values $\delta(s, i) / \lambda(s, i)$. This table is sometimes split into a next state table and an output table with entries $\delta(s, i)$ and $\lambda(s, i)$, respectively. Figure 5 shows an example, which is used throughout this paper.

To guarantee that a finite state machine can be implemented by a self-testable structure as shown in figure 4 it is necessary to require some additional properties.

S \ I	1	0
1	3/1	1/1
2	2/0	4/0
3	1/1	3/0
4	4/0	2/1

Figure 5: Example finite state machine specification.

Definition 2: Let $M = (S, I, O, \delta, \lambda)$ be a finite state machine. M is called a finite state machine supporting a self-testable structure, if and only if there are sets S_1 and S_2 and functions δ_1: $S_1 \times I \rightarrow S_2$, δ_2: $S_2 \times I \rightarrow S_1$, such that $S = S_1 \times S_2$ and $\delta((s_1, s_2), i) = (\delta_2(s_2, i), \delta_1(s_1, i))$ holds for all $s = (s_1, s_2) \in S$ and $i \in I$.

Obviously the straightforward implementation of such a finite state machine provides a self-testable structure with Registers R_1 and R_2 for the sets S_1 and S_2, combinational circuits C_1 and C_2 implementing the functions δ_1 and δ_2 and an output function λ.

Self-testable controllers can therefore be synthesized from finite state machine specifications in two steps. First a finite state machine which realizes the specification and which supports a self-testable structure is constructed and then state coding and logic minimization algorithms are applied to this realization. The term realization is used in the sense of definition 3.

Definition 3: Let $M = (S, I, O, \delta, \lambda)$ and $M^* = (S^*, I^*, O^*, \delta^*, \lambda^*)$ be two finite state machines. M^* realizes M, if and only if there is a tripel (α, ι, ζ) of mappings α: $S \rightarrow S^*$, ι: $I \rightarrow I^*$ and ζ: $O^* \rightarrow O$, such that $\delta^*(\alpha(s), \iota(i)) = \alpha(\delta(s, i))$ and $\zeta(\lambda^*(\alpha(s), \iota(i))) = \lambda(s, i)$ holds for all $s \in S$ and $i \in I$.

For a given finite state machine several realizations supporting self-testable structures might exist. To obtain self-testable controllers with small registers of about equal size the following problem has to be solved:

OSTR (**O**ptimal **S**elf-**T**estable **R**ealization): Let $M = (S, I, O, \delta, \lambda)$ be a finite state machine. Find a realization $M^* = (S_1^* \times S_2^*, I^*, O^*, \delta^*, \lambda^*)$ supporting a self-testable structure, such that

(i) $\lceil \log_2 |S_1^*| \rceil + \lceil \log_2 |S_2^*| \rceil$ is minimal, and

(ii) $\left| \dfrac{|S_1^*|}{|S_2^*|} - 1 \right|$ is minimal

for all solutions satisfying (i).

In the next section a constructive approach to solve this problem is presented.

3 An algorithm for OSTR based on partition pairs

The algorithm proposed in this section constructs a solution for problem OSTR by means of partition pairs. Before a detailed description is given the concept of partition pairs is repeated shortly and a theorem providing the theoretical basis for the presented algorithm is proven.

In the following equivalence relations on the set of states S of a finite state machine will always be considered as subsets $\pi \subset S \times S$. This way the set theoretic operators „$\cap$“ (intersection) and „$\cup$“ (union) are defined for equivalence relations and there is a partial ordering on the set of equivalence relations given by „$\subset$“ (subset). For an equivalence relation $\pi \subset S \times S$ and an element $s \in S$ the corresponding equivalence class is denoted by $[s]_\pi$. The set S/π of equivalence classes completely specifies π, and for convenience we define π mostly by S/π and not by enumerating all the pairs.

Definition 4: Let $M = (S, I, O, \delta, \lambda)$ be a finite state machine, and let $\pi, \tau \subset S \times S$ be equivalence relations on S. (π, τ) is called a partition pair for M, if and only if

$$(s, t) \in \pi \Rightarrow \forall\, i \in I: (\delta(s, i), \delta(t, i)) \in \tau \qquad (*)$$

holds. If (τ, π) is a partition pair, too, then (π, τ) is called a symmetric partition pair.

Condition (*) ensures that the state transition function δ maps equivalence classes under the relation π to uniquely determined equivalence classes under τ, and thus induces a well defined mapping $[\delta]$: $S/\pi \rightarrow S/\tau$ between the quotient spaces. For the solution of problem OSTR symmetric partition pairs are of special interest. The following theorem can be shown:

Theorem 1: Let $M = (S, I, O, \delta, \lambda)$ be a finite state machine. Let ε denote the equivalence of states and let (π, τ) be a symmetric partition pair for M satisfying $\pi \cap \tau \subset \varepsilon$. Let $M^* = (S^*, I^*, O^*, \delta^*, \lambda^*)$ be defined by

(i) $S^* := S/\pi \times S/\tau$, $I^* := I$, $O^* := O$,

(ii) $\delta^*((s_1, s_2), i)) := (\delta_2(s_2, i), \delta_1(s_1, i))$ with $\delta_1(([s]_\pi, i)) := [\delta(s, i)]_\tau$ and $\delta_2([s]_\tau, i)) := [\delta(s, i)]_\pi$ and

(iii) $\lambda^*((s_1, s_2), i) :=$
$$\begin{cases} \lambda(s,i) & \text{if } s_1 \cap s_2 \neq \emptyset \text{ and } s \in s_1 \cap s_2 \\ o^* & \text{else} \end{cases},$$
where $o^* \in O$ is an arbitrary output value.

Then M^* is a finite state machine supporting a self-testable structure which realizes M.

Proof: The functions δ_1 and δ_2 are well-defined, since (π, τ) is a symmetric partition pair for M. The function λ^* is well defined because of $\pi \cap \tau \subset \varepsilon$. Obviously M^* supports a self-testable structure. With mappings α : $S \rightarrow S^*$, $\alpha(s) := ([s]_\pi, [s]_\tau)$, ι : $I \rightarrow I^*$, $\iota(i) := i$ and ζ : $O^* \rightarrow O$, $\zeta(o) := o$ the equations $\delta^*(\alpha(s), \iota(i)) = \alpha(\delta(s, i))$ and $\zeta(\lambda^*(\alpha(s), \iota(i))) = \lambda(s, i)$ hold by definition of δ^* and λ^*, i.e. M^* realizes M. ❑

Theorem 1 is illustrated by the following example.

Example 1: Figure 6 shows a symmetric partition pair for the finite state machine of figure 5. It can be easily verified that for $S/\pi = \{\{1, 2\}, \{3, 4\}\}$ and $S/\tau = \{\{1, 4\}, \{2, 3\}\}$ equivalence classes under π are mapped by δ to uniquely determined equivalence classes under τ and thus (π, τ) is a partition pair. The same is true for (τ, π) and $\pi \cap \tau = \{(1,1), (2,2), (3,3), (4,4)\} \subset \varepsilon$.

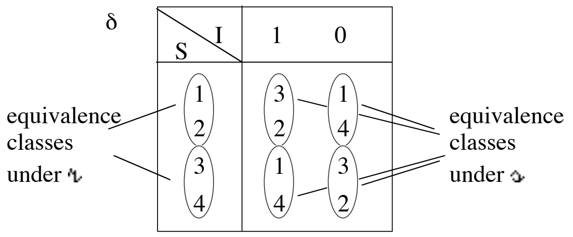
δ
S
I
1
0
equivalence classes under π
equivalence classes under τ
1 2 3 4
3 2 1 4
1 4 3 2

Figure 6: Effect of (π, τ) on the next state table of the finite state machine of figure 5.

The resulting mappings δ_1: $S/\pi \times I \rightarrow S/\tau$ and δ_2: $S/\tau \times I \rightarrow S/\pi$, which provide the state transition function δ^*, are shown in figure.

δ_1

S/π \ I	1	0
$[1]_\pi$	$[2]_\tau$	$[1]_\tau$
$[3]_\pi$	$[1]_\tau$	$[2]_\tau$

δ_2

S/τ \ I	1	0
$[1]_\tau$	$[3]_\pi$	$[1]_\pi$
$[2]_\tau$	$[1]_\pi$	$[3]_\pi$

Figure 7: Tables for δ_1: $S/\pi \times I \rightarrow S/\tau$ and δ_2: $S/\tau \times I \rightarrow S/\pi$.

If $[1]_\pi$ and $[1]_\tau$ are both encoded by 1 and $[3]_\pi$ and $[2]_\tau$ are encoded by 0, then the constructed finite state machine $M^* = (S/\pi \times S/\tau, I, O, \delta^*, \lambda^*)$ can be implemented by the structure shown in figure 8. ❑

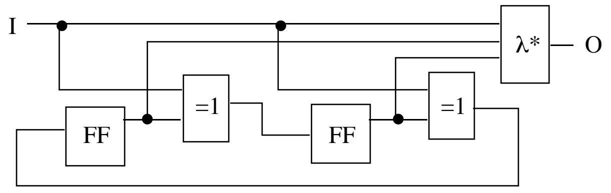

Figure 8: Structure of M*.

Theorem 1 has two consequences for the solution of problem OSTR. Firstly, there is always a trivial solution for problem OSTR, since the identity relation $\mathit{id} \subset S \times S$ provides a symmetric partition pair $(\mathit{id}, \mathit{id})$ with $\mathit{id} \cap \mathit{id} \subset \varepsilon$. The resulting finite state machine $M^* = (S/\mathit{id} \times S/\mathit{id}, I, O, \delta^*, \lambda^*)$ corresponds to „doubling" the original machine as shown in figure 3. Secondly, the problem of finding an optimal self-testable realization for a given finite state machine $M = (S, I, O, \delta, \lambda)$ reduces to the problem of finding a symmetric partition pair (π, τ) with $\pi \cap \tau \subset \varepsilon$, such that

(i) $\lceil \log_2 |S/\pi| \rceil + \lceil \log_2 |S/\tau| \rceil$ is minimal, and

(ii) $\left| \frac{|S/\pi|}{|S/\tau|} - 1 \right|$ is minimal for all pairs satisfying (i).

To solve this problem a search procedure has been developed which makes use of the lattice structure of the set of partition pairs. In fact, it will be shown that the search space can be mainly reduced to so-called Mm-pairs.

Definition 5: Let $M = (S, I, O, \delta, \lambda)$ be a finite state machine and let $\pi, \tau \subset S \times S$ be equivalence relations on the set of states. Then $m(\pi)$ denotes the $\subset$-minimal equivalence relation, such that $(\pi, m(\pi))$ is a partition pair, and $M(\tau)$ denotes the $\subset$-maximal equivalence relation, such that $(M(\tau), \tau)$ is a partition pair. (π, τ) is called an Mm-pair, if both $M(\tau) = \pi$ and $m(\pi) = \tau$ hold.

The Mm-pairs for a finite state machine M form a lattice, which has been studied intensively by [16]. The Mm-lattice can be regarded as the skeleton for the set of all partition pairs. The correspondence between Mm-pairs and symmetric partition pairs is described by the following theorem.

Theorem 2: Let $M = (S, I, O, \delta, \lambda)$ be a finite state machine, and let $\pi, \tau \subset S \times S$ be equivalence relations. The pair (π, τ) is a symmetric partition pair, if and only if there is an Mm-pair (π^*, τ^*) with $m(\tau^*) \subset \pi \subset \pi^*$ and $\tau^* \subset \tau \subset M(\pi^*)$, which is also a symmetric partition pair.

Proof: Let (π, τ) be a symmetric partition pair. Because (π, τ) is a partition pair, by [16] there is an Mm-pair (π^*, τ^*) with $\pi \subset \pi^*$ and $\tau^* \subset \tau$. Since (τ, π) is also a partition pair, $\tau^* \subset \tau$ implies $\big((s, t) \in \tau^* \Rightarrow (s, t) \in \tau\big)$ and consequently $\big(\forall\, i \in I: \big(\delta(s, i), \delta(t, i)\big) \in \pi \subset \pi^*\big)$ is true, which proves, that (τ^*, π^*) is also a partition pair. By [16] this provides $m(\tau^*) \subset \pi \subset \pi^*$ and $\tau^* \subset \tau \subset M(\pi^*)$.

If, conversely, there is an Mm-pair (π^*, τ^*) which is a symmetric partition pair, then by [16] (π, τ) with $\pi \subset \pi^*$ and $\tau^* \subset \tau$ is a partition pair and also (τ, π) with $\tau \subset M(\pi^*)$ and $m(\tau^*) \subset \pi$. ❑

Consequently, if there is no Mm-pair for a finite state machine M which is a symmetric partition pair, then there is no symmetric partition pair for M. Furthermore Mm-pairs mostly provide more balanced realizations because of $\pi \subset \pi^*$ and $\tau^* \subset \tau$. With respect to problem OSTR it is important to note, that for an Mm-pair (π^*, τ^*) the pair $(m(\tau^*), \tau^*)$ has the minimal intersection of all pairs in $\{(\pi, \tau) \mid m(\tau^*) \subset \pi \subset \pi^* \text{ and } \tau^* \subset \tau \subset M(\pi^*)\}$, i.e. if $m(\tau^*) \cap \tau^* \not\subset \varepsilon$, then $\pi \cap \tau \not\subset \varepsilon$ for all pairs in $\{(\pi, \tau) \mid m(\tau^*) \subset \pi \subset \pi^* \text{ and } \tau^* \subset \tau \subset M(\pi^*)\}$.

The Mm-lattice for a finite state machine $M = (S, I, O, \delta, \lambda)$ can be calculated from certain basis relations $\rho_{s,t}$, where $\rho_{s,t} := \mathit{id} \cup \{(s, t), (t, s)\}$ is the equivalence relation identifying the states s and t in S and distinguishing all other states [16]. Based on the procedure described in [16] and on the conclusions drawn from theorem 2 a search tree (V, E) for problem OSTR is constructed as follows:

First the set $\mathcal{M} := \{m(\rho_{s,t}) \mid s, t \in S\}$ is generated and ordered arbitrarily ($\mathcal{M} = \{m_1, m_2, \ldots, \}$). The nodes of the searchtree correspond to subsets $\mathcal{N} \subset \mathcal{M}$. A node

$\mathcal{N}_{new}$ is a successor of a node $\mathcal{N}_{old}$, if and only if $\mathcal{N}_{new} = \mathcal{N}_{old} \cup \{\mu_k\}$ with $k > \max\{i \mid \mu_i \in \mathcal{N}_{old}\}$, i. e.:

$V := \mathcal{P}(\mathcal{M})$

$E := \Big\{ (\mathcal{N}_1, \mathcal{N}_2) \in V \times V \mid \mathcal{N}_2 = \mathcal{N}_1 \cup \{\mu_k\}$ with $k > \max\{i \mid \mu_i \in \mathcal{N}_1\} \Big\}$

The root of the search tree is $\emptyset$.

For each node $\mathcal{N}$ in the search tree $\pi := (\bigcup \mathcal{N})^t$ and $M(\pi)$ are calculated, where τ^t denotes the transitive closure of a relation τ. By [16] $(M(\pi), \pi)$ is an Mm-pair. If $(\pi, M(\pi))$ is also a partition pair and $M(\pi) \cap \pi \subset \varepsilon$, then $(M(\pi), \pi)$ provides a solution for OSTR and the costs

$$\lceil \log_2 |S/M(\pi)| \rceil + \lceil \log_2 |S/\pi| \rceil \text{ and } \left| \frac{|S/M(\pi)|}{|S/\pi|} - 1 \right|$$

are calculated. If $M(\pi) \cap \pi \not\subset \varepsilon$, then $m(\pi)$ is calculated. By theorem 2 $(m(\pi), \pi)$ is a symmetric partition pair with $m(\pi) \cap \pi \subset M(\pi) \cap \pi$. If $m(\pi) \cap \pi \subset \varepsilon$, then $(m(\pi), \pi)$ is a solution for OSTR and the costs are calculated for this pair. Finally the solution with minimal costs is selected to realize the specification.

This basic search procedure is of very high complexity, since the number of nodes in the searchtree is $|V| = O(2^{|S|^2})$. But the following lemma provides a criterion to prune the search tree.

Lemma 1: Let $M = (S, I, O, \delta, \lambda)$ be a finite state machine, and let (V, E) be the search tree defined above. For a node $(\mathcal{N}_1, \mathcal{N}_2) \in E$ let $\pi_1 := (\bigcup \mathcal{N}_1)^t$ and $\pi_2 := (\bigcup \mathcal{N}_2)^t$. If $m(\pi_1) \cap \pi_1 \not\subset \varepsilon$, then $m(\pi_2) \cap \pi_2 \not\subset \varepsilon$.

Proof: By definition of the searchtree $\pi_1 \subset \pi_2$, and by [16] this implies $m(\pi_1) \subset m(\pi_2)$, and thus $m(\pi_1) \cap \pi_1 \subset m(\pi_2) \cap \pi_2$. ❑

As a consequence of lemma 1, once a node $\mathcal{N}$ in the searchtree with $M(\pi) \cap \pi \not\subset \varepsilon$ is reached, all of its successors have this property and the subtree rooted at $\mathcal{N}$ can be discarded. As demonstrated by the experimental results described in the next section this leads to an enormous reduction of the computational effort.

4 Experimental results

The algorithm for problem OSTR described in section 3 has been implemented as a depthfirst procedure and has been applied to most of the fully specified finite state machine benchmarks distributed for the International Workshop on Logic Synthesis '93 [20]. The results are shown in table 1. Column 2 contains the number of states in the original finite state machine, and columns 3 and 4 contain the number of states in the factors S_1 and S_2 of the best realization found. Columns 5 and 6 list the required number of flipflops for a conventional BIST and for a BIST with the optimized structure by the presented synthesis approach. Except for tbk, for all examples the exact solution for OSTR could be calculated. For tbk the solution obtained within a given timelimit is shown.

Name	$\lvert S \rvert$	$\lvert S_1 \rvert$	$\lvert S_2 \rvert$	# FFs conv. BIST	# FFs pipeline structure
bbara	10	7	7	8	6
bbtas	6	6	6	6	6
dk14	7	7	7	6	6
dk15	4	4	4	4	4
dk16	27	24	24	10	10
dk17	8	8	8	6	6
dk27	7	6	7	6	6
dk512	15	14	15	8	8
mc	4	4	4	4	4
s1	20	20	20	10	10
shiftreg	8	4	2	6	3
tav	4	2	2	4	2
tbk*)	32	16	16	10	8

Table 1: Results of depthfirst search procedure for OSTR.
*) timeout

The practical impact of lemma 1 on the computational effort is demonstrated in table 2. Column 3 lists the overall number $|V|$ of nodes in the searchtree for OSTR in contrast to the number of nodes that had to be investigated when pruning the searchtree according to lemma 1 (column 4).

Name	$\lvert S \rvert$	$\lvert V \rvert$	# nodes investigated
bbara	10	2^{43}	815
bbtas	6	2^{16}	175
dk14	7	2^{10}	19
dk15	4	2^{4}	7
dk16	27	2^{206}	337041
dk17	8	2^{20}	63
dk27	7	2^{16}	203
dk512	15	2^{56}	343853
mc	4	2^{7}	13
s1	20	2^{162}	323
shiftreg	8	2^{8}	49
tav	4	2^{7}	47

Table 2: Impact of lemma 1 on the computational effort.

The results in table 1 show that for eight examples a nontrivial solution for OSTR, i.e. a solution with $|S_1| < |S|$ or $|S_2| < |S|$, could be found. For shiftreg and tav even the lower bound $|S_1| \cdot |S_2| = |S|$ is achieved. In these eight examples the combined networks C_1 and C_2 need to implement less state transitions than the original network C. Depending on the implementation style significant hardware savings are obtained compared to simply doubling C as shown in figure 3, whereby the advantages with respect to fault coverage and speed are retained. In four examples even the number of flipflops required for a self-testable pipelined controller is smaller than the number required for a conventional BIST.

5 Conclusions and future work

A method has been presented for implementing self-testable controllers without doubling the system registers during test mode. The proposed pipeline-like structure does not contain any direct feedback loops and is partitioned by two system registers. During self-test these registers perform test pattern generation and signature analysis alternatively. This architecture reduces the delay imposed by bypassing test registers and increases the fault coverage.

A synthesis procedure has been presented for generating minimal pipelined realizations from state transition diagrams. In most cases this optimized solution is superior to simply doubling the registers and combinational networks, and in many cases the number of flipflops is less than it is required for a conventional BIST. This indicates that not only higher speed and fault coverage is obtainable this way, but also area can be saved.

Future work will concentrate on modifying the state transition diagram to obtain functionally equivalent machines whose self-testable realizations lead to better solutions of problem OSTR.

6 References

1 V. D. Agrawal, C. R. Kime, K. K. Saluja: A Tutorial on Built-In Self-Test, Part 1: Principles, IEEE Design and Test of Comp., Vol. 10, No. 1, March 1993, pp. 73-82

2 P. Ashar, S. Devadas: Irredundant Interacting Sequential Machines Via Optimal Logic Synthesis, IEEE Trans. on CAD, Vol. 10, No. 3, March 1991, pp. 311-325

3 P. Ashar, S. Devadas, A. R. Newton: A Unified Approach to the Decomposition and Re-decomposition of Sequential Machines, Proc. 27th Design Automation Conf., 1990, pp. 601-606

4 Z. Barzilai, D. Coppersmith, A. L. Rosenberg: Exhaustive Generation of Bit Patterns with Applications to VLSI Self-Testing, IEEE Trans. on Comp., Vol. c-32, No. 2, Feb. 1983, pp. 190 - 194

5 R. K. Brayton, G. D. Hachtel, C. T. McMullen: Logic Minimization Algorithms for VLSI Synthesis, Kluwer Academic Publishers, Boston 1984

6 R. K. Brayton et al.: MIS: A Multiple-Level Logic Optimization System, IEEE Trans. on CAD, Vol. CAD-6, No. 6, 1987, pp. 1062-1081

7 V. D. Agrawal, K.-T. Cheng: Finite State Machine Synthesis with Embedded Test Function, Journal of Electronic Testing Theory and Applications, Vol. 1, No. 3, Oct. 1990, pp. 221-228

8 K.-T. Cheng, V. D. Agrawal: State Assignment for Testable Design, Int. Journal of Computer Aided VLSI Design, Vol. 3., March 1991

9 S. T. Chakradar, S. Kanjilal, V. D. Agrawal: Finite State Machine Synthesis with Fault Tolerant Test Function, Proc. 29th Automation Conf., Anaheim, Ca., 1992, pp. 562-567

10 W. Daehn, J. Mucha: Hardware Test Pattern Generation for Built-In Test, Proc. IEEE Int. Test Conf., Philadelphia, 1981, pp. 100 - 113

11 S. Devadas, K. Keutzer: A Unified Approach to the Synthesis of Fully Testable Sequential Machines, IEEE Trans. on CAD, Vol. 10, No. 1, Jan. 1991, pp. 39-50

12 S. Devadas et al.: MUSTANG: State Assignment of Finite State Machines Targeting Mulitlevel Logic Implementations, IEEE Trans. on CAD, Vol. 7, No. 12, Dec. 1988, pp. 1290-1300

13 B. Eschermann, H.-J. Wunderlich: Parallel Self-Test and the Synthesis of Control Units, Proc. 2nd European Test Conf., Munich, 1991

14 B. Eschermann, H.-J. Wunderlich: Optimized Synthesis Techniques for Testable Sequential Circuits, IEEE Trans. on CAD, Vol. 11, No. 3, March 1992, pp. 301-312

15 Martin Geiger, Thomas Müller-Wipperfürth: FSM Decomposition Revisited: Algebraic Structure Theory Applied to MCNC Benchmark FSMs, Proc. 28th Design Automation Conf., San Francisco, 1991, pp. 182-185

16 J. Hartmanis, R. E. Stearns: Algebraic Structure Theory of Sequential Machines, Prentice Hall, Englewood Cliffs, 1966

17 S. Hellebrand, H.-J. Wunderlich, O. Haberl: Generating Pseudo-Exhaustive Vectors for External Testing, Proc. Int. Test Conf., Washingtion, D. C., 1990, pp. 670-679

18 K. Kim, D. Ha, J. Tront: On Using Signature Registers as Pseudorandom Pattern Generators in Built-in Self Testing, IEEE Trans. on CAD, Vol. 7, 1988, pp. 919-928

19 B. Koenemann, J. Mucha, G. Zwiehoff: Built-in Logic Block Observation Techniques, Proc. IEEE Int. Test Conf., Cherry Hill, N. J., 1979, pp. 37 - 41

20 K. McElvain: IWLS'93 Benchmark Set: Version 4.0, distributed as part of the IWLS'93 benchmark distribution

21 I. Pomeranz, S. M. Reddy: Design and Synthesis for Testability of Synchronous Sequential Circuits Based on Strong-Connectivity, Proc. IEEE 23rd Int. Symp. on Fault-Tolerant Computing, Toulouse, 1993, pp. 492-501

22 G. Saucier, M. C. De Paulet, P. Sicard: ASYL: A Rule-Based System for Controller Synthesis, IEEE Trans. on CAD, Vol. CAD-6, No. 6, Nov. 1987, pp. 1088-1097

23 T. Villa, A. Sangiovanni-Vincentelli: NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementations, Proc. 26th Design Automation Conf., Las Vegas, 1989, pp. 327-332

24 P. Weiner, E. J. Smith: Optimization of Reduced Dependencies for Synchronous Sequential Machines, IEEE Trans. on Electronic Comp., Vol. EC-16, No. 6, Dec. 1967, pp. 835-847

25 H.-J. Wunderlich: Self Test Using Unequiprobable Random Patterns, Proc. IEEE 17th Int. Symp. on Fault-Tolerant Computing, Pittsburgh, 1987, pp. 258-263