An Efficient Procedure for the Synthesis of Fast Self-Testable Controller Structures

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

Abstract

The BIST implementation of a conventionally synthesized controller in most cases requires the integration of an additional register only for test purposes. This leads to some serious drawbacks concerning the fault coverage, the system speed and the area overhead. A synthesis technique is presented which uses the additional test register also to implement the system function by supporting self-testable pipeline-like controller structures. It will be shown, that if the need of two different registers in the final structure is already taken into account during synthesis, then the overall number of flipflops can be reduced, and the fault coverage and system speed can be enhanced. The presented algorithm constructs realizations of a given finite state machine specification which can be trivially implemented by a self-testable structure. The efficiency of the procedure is ensured by a very precise characterization of the space of suitable realizations, which avoids the computational overhead of previously published algorithms.

1 Introduction

The increasing demand for highly reliable micro-electronic systems in various safety-critical applications prompts for extremely high quality standards, which have to be guaranteed by refined testing techniques. Built-in self-test (BIST) is of particular importance, because it allows an efficient production testing and the capabilities for pattern generation and test response evaluation on chip can also be used for periodic maintenance tests [1].

Often the BIST is implemented by so-called multi-functional 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 registers have been developed for random, deterministic, pseudo-exhaustive, and weighted random pattern testing [4, 9, 17, 20, 26]. However, in general it is not possible to use one multi-functional register for test pattern generation and test response evaluation concurrently, since this way the required properties of the test patterns can only be ensured in some special cases [12, 14, 19]. In most cases two different registers are necessary to generate the patterns and to compress the test responses. Conventional synthesis procedures for controllers and even most of the advanced synthesis techniques for sequentially irredundant and easily testable controllers do not take into account this fact [2, 5, 6, 7, 10, 11, 13, 22, 23, 24]. They usually provide a circuit structure as shown in figure 1, which has to be complemented by an extra test register for BIST (see figure 2).

![Figure 1: Conventionally synthesized controller structure.](image1)

![Figure 2: Required modifications for BIST.](image2)

During test mode the register T is used as pattern generator and the multi-functional register R is configured as signature analyzer, during system mode T must be transparent. This configuration with an extra register only for test purposes has some serious drawbacks:

1) The number of flipflops must be doubled.

This work was supported by EP 7107 ARCHIMEDES.
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 above mentioned drawbacks can be avoided if the required additional test register T is also used to implement the system function. This results in a pipeline-like structure as shown in figure 3 and described in detail in [18].

![Figure 3: Controller target structure.](image)

A state of the controller is represented by the contents of both registers R1 and R2, and the state transition function is implemented by two independent combinational circuits C1 and C2. Clearly 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 formation in two sessions by alternatively using one of the registers R1 and R2 of the proposed target structure, hence the structure of figure 3 needs less flipflops than the original circuit of figure 1. This allows higher clock rates and thus leads to a higher system performance.

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 [18] a computationally expensive search procedure was used for a feasibility study of the approach. In this paper a computationally efficient synthesis procedure for target structures as shown in figure 3 is presented, which makes the approach generally applicable. In contrast to known approaches trying to reduce dependencies between state variables by appropriate state coding the presented procedure relies on algebraic structure theory to address the problem already at the finite state machine level [25, 8]. The theoretical basis of the work is a very precise characterization of the search space.

The organization of the paper is as follows: In section 2 the main results of [18] are briefly summarized. Subsequently in section 3 the main theorem for a precise characterization of the search space is proven and an efficient search procedure is developed. Section 4 provides experimental results achieved for a collection of finite state machine benchmarks [McEl 93]. Conclusions are given in section 5.

## 2 Partition Pairs and Self-Testable Realizations

The problem of synthesizing controller structures compatible with BIST can be reduced to the problem of constructing suitable realizations of the original finite state machine specifications.

Throughout this work it is assumed that a controller is fully specified as a mealy-type finite state machine M = (S, I, O, δ, λ) with a finite non-empty set of states S, a finite non-empty set of inputs I, a finite non-empty set of outputs O, a next state function δ: S × I → S and an output function λ: S × I → O.

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

**Definition 1:** Let M = (S, I, O, δ, λ) be a finite state machine. M is called a finite state machine supporting a self-testable structure, if and only if there are sets S1 and S2 and functions δ1: S1 × I → S2, δ2: S2 × I → S1, such that S = S1 × S2 and δ(s1, s2), i) = (δ2(s2, i), δ1(s1, i)) holds for all s = (s1, s2) ∈ S and i ∈ I.

Obviously the straightforward implementation of such a finite state machine with logic blocks for δ1, δ2 and λ and registers for S1 and S2 provides a self-testable structure as shown in figure 3.

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. A precise definition of the term realization can be found in [16, 18].

In [18] it has been shown that self-testable realizations for a finite state machine M = (S, I, O, δ, λ) correspond to specific pairs of equivalence relations on the set of states S. In the following equivalence relations on S will always be considered as subsets τ ⊆ S x S. This way the set theoretic operators τ ∩ (intersection) and τ ∪ (union) are defined for equivalence relations and there is a partial ordering on the set of equivalence relations given by τ ⊆ (sub-
set). The intersection of two equivalence relations is again an equivalence relation, but the union need not be transitive. Therefore an operator $\cdot^+$ is defined for equivalence relations by $\varphi_1 + \varphi_2 := (\varphi_1 \cup \varphi_2)^1$, where $\varphi^1$ denotes the transitive closure of a relation $\varphi$. For an equivalence relation $\varphi \subseteq S \times S$ and an element $s \in S$ the corresponding equivalence class is denoted by $[s]_\varphi$. The set $S/\varphi$ of equivalence classes completely specifies $\varphi$, and for convenience we describe $\varphi$ mostly by $S/\varphi$ and not by enumerating all the pairs.

**Definition 2**: Let $M = (S, I, O, \delta, \lambda)$ be a finite state machine, and let $\varphi, \varphi' \subseteq S \times S$ be equivalence relations on $S$. $(\varphi, \varphi')$ is called a partition pair for $M$, if and only if $((s, t) \in \varphi) \Rightarrow \forall i \in I: (\delta(s, i), \delta(t, i)) \in \varphi'$ holds. If $\varphi$ is a partition pair, too, then $(\varphi, \varphi')$ is called a symmetric partition pair.

For the construction of self-testable realizations symmetric partition pairs are of special interest. The following theorem has been shown in [18]:

**Theorem 1**: Let $M = (S, I, O, \delta, \lambda)$ be a finite state machine. Let $\varphi$ denote the equivalence of states and let $\varphi, \varphi' \subseteq S \times S$ be equivalence relations on $S$. $(\varphi, \varphi')$ is called a partition pair for $M$, if and only if $((s, t) \in \varphi) \Rightarrow \forall i \in I: (\delta(s, i), \delta(t, i)) \in \varphi'$ holds. If $\varphi$ is a partition pair, too, then $(\varphi, \varphi')$ is called a symmetric partition pair.

For the construction of self-testable realizations symmetric partition pairs are of special interest. The following theorem has been shown in [18]:

**Theorem 1**: Let $M = (S, I, O, \delta, \lambda)$ be a finite state machine. Let $\varphi$ denote the equivalence of states and let $\varphi, \varphi' \subseteq S \times S$ be equivalence relations on $S$. $(\varphi, \varphi')$ is called a partition pair for $M$, if and only if $((s, t) \in \varphi) \Rightarrow \forall i \in I: (\delta(s, i), \delta(t, i)) \in \varphi'$ holds. If $\varphi$ is a partition pair, too, then $(\varphi, \varphi')$ is called a symmetric partition pair.

Then $M^*$ is a finite state machine supporting a self-transitive closure of a relation $\delta$. Theorem 1 has been shown in [18]:

**Theorem 1**: Let $M = (S, I, O, \delta, \lambda)$ be a finite state machine. Let $\varphi$ denote the equivalence of states and let $\varphi, \varphi' \subseteq S \times S$ be equivalence relations on $S$. $(\varphi, \varphi')$ is called a partition pair for $M$, if and only if $((s, t) \in \varphi) \Rightarrow \forall i \in I: (\delta(s, i), \delta(t, i)) \in \varphi'$ holds. If $\varphi$ is a partition pair, too, then $(\varphi, \varphi')$ is called a symmetric partition pair.

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

**Example 1**: Figure 4 shows the next state table of a small finite state machine. By $S/\varphi = \{\{1, 2\}, \{3, 4\}\}$ and $S/\varphi' = \{\{1, 4\}, \{2, 3\}\}$ a symmetric partition pair $(\varphi, \varphi')$ with $\varphi \cap \varphi' \subseteq \varphi$ is defined.

\[
\begin{array}{c|c|c|c}
S & 1 & 0 \\
\hline
1 & 2 & 3 \\
\hline
2 & 1 & 4 \\
\hline
3 & 4 & 1 \\
\hline
4 & 3 & 2 \\
\end{array}
\]

Figure 4: Effect of a partition pair $(\varphi, \varphi')$ on the next state table of a finite state machine.

The resulting mappings $\delta_1: S/\varphi' \times I \rightarrow S/\varphi'$ and $\delta_2: S/\varphi' \times I \rightarrow S/\varphi$, which provide the state transition function $\delta^*$, are shown in Figure 5.

\[
\begin{array}{c|c|c|c}
S/\varphi' & 1 & 0 \\
\hline
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c}
S/\varphi' & 1 & 0 \\
\hline
\hline
\end{array}
\]

Figure 5: Tables for $\delta_1: S/\varphi' \times I \rightarrow S/\varphi$ and $\delta_2: S/\varphi' \times I \rightarrow S/\varphi$.

If $[1]_\varphi$ and $[3]_\varphi$ are both encoded by 1 and $[2]_\varphi$ are encoded by 0, then the constructed finite state machine $M^* = (S/\varphi' \times S/\varphi', I, O, \delta^*, \lambda^*)$ can be implemented by the structure shown in figure 6.

Figure 6: Structure of $M^*$.

Theorem 1 has two consequences for the synthesis of self-testable controllers. Firstly, there is always a trivial self-testable realization for a given finite state machine $M = (S, I, O, \delta, \lambda)$, since the identity relation $\{\{s\}\} \subseteq S \times S$ provides a symmetric partition pair $(\{\{s\}\}, \{\{s\}\})$ with $\{\{s\}\} \cap \{\{s\}\} \subseteq \{\{s\}\}$. The resulting finite state machine $M^* = (S/\{\{s\}\} \times S/\{\{s\}\}, I, O, \delta^*, \lambda^*)$ corresponds to simply „doubling“ the original machine. Secondly, the problem of finding an optimal self-testable realization with small registers of about equal size can be formulated as follows:

**Problem OSTR (Optimal Self-Testable Realization)**: Let $M = (S, I, O, \delta, \lambda)$ be a finite state machine. Find a symmetric partition pair $(\varphi, \varphi')$ with $\varphi \cap \varphi' \subseteq \varphi$, such that

(i) $|\log_2 |S/\varphi'|-1|$ is minimal, and

(ii) $|S/\varphi'| - 1$ is minimal

for all pairs satisfying (i).

In the next section an algorithm for OSTR will be developed, which is based on an efficient enumeration procedure for symmetric partition pairs.

### 3 An Efficient Algorithm for OSTR

In the previous section it has been shown that the problem of synthesizing self-testable controllers can be reduced to finding suitable symmetric partition pairs. The algorithm to be developed in this paragraph rigorously exploits the symmetry requirement in order to restrict the search to a small number of candidate pairs. It is based on the notion of „mm-pairs“ which will be introduced in the sequel.
Definition 3: Let $M = (S, I, O, \delta, \lambda)$ be a finite state machine and let $\tau, \varphi \subseteq S \times S$ be equivalence relations. Then $m(\tau)$ denotes the C-minimal equivalence relation, such that $(\tau, m(\tau))$ is a partition pair, and $M(\varphi)$ denotes the C-maximal equivalence relation, such that $(M(\varphi), \varphi)$ is a partition pair. $(\tau, \varphi)$ is called an $\text{Mm-pair}$, if both $M(\varphi) = \tau$ and $m(\tau) = \varphi$ hold. If $m(\tau) = \tau$ and $m(\varphi) = \varphi$ are both true, then $(\tau, \varphi)$ is called an $\text{mm-pair}$.

The \text{Mm-pairs} for a finite state machine can be regarded as the skeleton for the set of all partition pairs [16], and the basic procedure for OSTR described in [18] used an enumeration procedure for the set of \text{Mm-pairs}. Since \text{Mm-pairs} need not be symmetric, this resulted in an unnecessary computational overhead. In the following it will be shown that the set of symmetric partition pairs can be directly characterized by \text{mm-pairs}. By definition of the m-operator an \text{mm-pair} is a symmetric partition pair. To show that it is sufficient to concentrate on \text{mm-pairs} some more theoretical effort is required. The first result derived from the symmetry property concerns the iterative construction of an mm-pair from an initial relation $\rho$. With $m(\rho)$ denoting the relation $m(m^{-1}(\rho))$, where $m^0(\rho) := \rho$, the following lemma can be shown.

Lemma 1: Let $M = (S, I, O, \delta, \lambda)$ be a finite state machine, and let $\rho \subseteq S \times S$ be an equivalence relation. Then the relations

$$m_{\text{even}}(\rho) := \sum_{i=0}^{\infty} m^{2i}(\rho) \quad \text{and} \quad m_{\text{odd}}(\rho) := \sum_{i=0}^{\infty} m^{2i+1}(\rho)$$

constitute an \text{mm-pair} $(m_{\text{even}}(\rho), m_{\text{odd}}(\rho))$. Furthermore, if $(\tau, \varphi)$ is any symmetric partition pair with $\rho \subseteq \tau$, then $m_{\text{even}}(\rho) \subseteq \tau$ and $m_{\text{odd}}(\rho) \subseteq \varphi$.

Some useful properties of the operators $m_{\text{even}}$ and $m_{\text{odd}}$ can inductively be derived from the corresponding properties of the m-operator proven in [16].

Observation 1: Let $M = (S, I, O, \delta, \lambda)$ be a finite state machine, and let $\tau, \varphi \subseteq S \times S$ be equivalence relations. Then $m_{\text{even}}(\tau + \varphi) = m_{\text{even}}(\tau) + m_{\text{even}}(\varphi)$ and $m_{\text{odd}}(\tau + \varphi) = m_{\text{odd}}(\tau) + m_{\text{odd}}(\varphi)$.

Observation 2: Let $M = (S, I, O, \delta, \lambda)$ be a finite state machine, and let $\tau \subseteq \varphi \subseteq S \times S$ be equivalence relations. Then $m_{\text{even}}(\varphi) \subseteq m_{\text{even}}(\tau)$ and $m_{\text{odd}}(\tau) \subseteq m_{\text{odd}}(\varphi)$.

The following characterization of symmetric partition pairs is an immediate consequence of lemma 1.

Theorem 2: Let $M = (S, I, O, \delta, \lambda)$ be a finite state machine, and let $\tau, \varphi \subseteq S \times S$ be equivalence relations. The pair $(\tau, \varphi)$ is a symmetric partition pair, if and only if there is an mm-pair $(\tau^*, \varphi^*)$ with $\tau^* = \tau$ and $\varphi^* \subseteq \varphi$. With respect to problem OSTR it is important to note, that an mm-pair $(\tau^*, \varphi^*)$ has the minimal intersection of all pairs in $\{ (\tau, \varphi) | \tau^* = \tau^* \text{ and } \varphi^* \subseteq \varphi \subseteq M(\varphi) \}$, i.e. if $\tau^* \cap \varphi^* \not\subseteq \tau$, then $\tau \not\subseteq \varphi \subseteq \tau$ for all pairs in $\{ (\tau, \varphi) | \tau^* = \tau^* \text{ and } \varphi^* \subseteq \varphi \subseteq M(\varphi) \}$. Consequently, if the set of \text{mm-pairs} does not provide a better solution than $(\tau^*, \varphi^*)$ for OSTR, then there is no better solution at all. Furthermore, using theorem 2 it is possible to derive all solutions from the solutions found in the set of \text{mm-pairs}. Therefore the search space can be restricted to \text{mm-pairs}, and the algorithm for OSTR will be based on an efficient procedure to enumerate the set of \text{mm-pairs}.

It is possible to construct the set of \text{mm-pairs} from the base relations $\mathcal{B}(S) := \{ \rho_{s,t} | s, t \in S \}$, where $\rho_{s,t} := \delta \cap \{ (s, t), (t, s) \}$ is the equivalence relation identifying the states $s$ and $t$ in $S$ and distinguishing all other states.

By lemma 1 an \text{mm-pair} $(\tau, \varphi)$ is characterized by $m_{\text{even}}(\tau) = \tau$ and $m_{\text{odd}}(\varphi) = \varphi$. Thus the set of all mm-pairs for a finite state machine $M = (S, I, O, \delta, \lambda)$ is described by $\mathcal{R}_{\text{mm}}(M) := \{ (m_{\text{even}}(\tau), m_{\text{odd}}(\varphi)) | \tau \subseteq S \times S \}$ is an equivalence relation. Obviously, $\tau = \sum_{(s,t)\in\tau} \rho_{s,t}$ is true for any equivalence relation, and $\mathcal{R}_{\text{mm}}(M) = \{ (m_{\text{even}}(\tau), m_{\text{odd}}(\varphi)) | \tau \in \mathcal{B}(S) \}$.

Therefore $\mathcal{R}_{\text{mm}}(M)$ can be constructed by enumerating all possible sums of relations in $\mathcal{B}(S)$ and calculating the corresponding mm-pair with the help of the $m_{\text{even}}$ and the $m_{\text{odd}}$-operator. However, this would require a computational effort of $O(2^{|S|^2})$. In fact, only those relations in $\mathcal{B}(S)$ have to be considered which lead to different mm-pairs. To make this more precise a partial ordering „$\leq$“ on equivalence relations is introduced as follows:

Definition 4: Let $M = (S, I, O, \delta, \lambda)$ be a finite state machine, and let $\tau, \varphi \subseteq S \times S$ be equivalence relations. Then $\varphi$ is said to dominate $\tau$ ($\tau \leq \varphi$), if and only if $m_{\text{even}}(\varphi) \subseteq m_{\text{even}}(\tau)$. $\tau$ and $\varphi$ are called $\text{mm-equivalent}$ ($\tau \sim \varphi$), if both $\tau \leq \varphi$ and $\varphi \leq \tau$ are true.

It is easily verified that „$\leq$“ is in fact an equivalence relation, and observations 1 and 2 provide the following observation:

Observation 3: Let $M = (S, I, O, \delta, \lambda)$ be a finite state machine, and let $\tau, \varphi \subseteq S \times S$ be equivalence relations. Then $\tau \leq \varphi$ implies $m_{\text{even}}(\varphi) = m_{\text{even}}(\tau)$.

For mm-equivalent relations $\tau \sim \varphi$ observation 3 yields $m_{\text{even}}(\tau) = m_{\text{even}}(\varphi) + \varphi = m_{\text{even}}(\varphi)$ and therefore the same mm-pair is obtained from $\tau$ and $\varphi$. To construct the set of all mm-pairs it is sufficient to consider the quotient space $\mathcal{B}(S)/\sim$.
Then the set of mm-pairs for M is characterized by
\[ R_{mm}(M) = \{ (m_{e\text{ven}}(\eta), m_{o\text{dd}}(\eta)) \mid \eta = \sum_{\mathcal{P} \in \mathcal{D}} \mathcal{P} \subseteq (S)/\sim \}. \]

Making use of theorem 3 and observation 1 the following basic search tree \((V, E)\) can be constructed to enumerate all mm-pairs. First the quotient \((S)/\sim\) is computed. This computation also provides the set \(\eta_{\text{even}} := \{ m_{\text{even}}(\eta) \mid \eta \in (S)/\sim \}. \) With \(\eta_{\text{even}} = \{ \eta_1, \eta_2, \ldots \} \) ordered arbitrarily \((V, E)\) is defined by:

\[ V := \mathcal{P}(\eta_{\text{even}}) \]

\[ E := \{ (\mathcal{P}_1, \mathcal{P}_2) \in V \times V \mid \mathcal{P}_2 = \mathcal{P}_1 \cup \{ \eta_k \} \text{ with } k > \max \{ i \mid \eta_i \in \mathcal{P}_1 \} \} \]

The root of the search tree is \(\emptyset\). Each vertex in this tree corresponds to a subset \(\mathcal{P} \subseteq \eta_{\text{even}}\) and provides an mm-pair \((\eta_0, m(\eta_0))\) with \(\eta_0 := \sum_{\eta_i \in \mathcal{P}} \eta_i.\) The tree has \(O(2^{\#\eta_{\text{even}}})\) vertices and in general \(\#\eta_{\text{even}}\) is much smaller than \(\#S^2\), which already leads to an enormous reduction of the computational effort compared to the straightforward approach. In addition to that, observation 3 shows that edges \((\mathcal{P}^1, \mathcal{P}^2)\) with \(m_{\text{even}}(\mathcal{P}^2) \subseteq \mathcal{P}^1 \) can be omitted, since in this case \(m_{\text{even}}(m_{\text{even}}(\eta_0) + m_{\text{even}}(\eta_0)) = m_{\text{even}}(\eta_0).\) Therefore a reduced tree \((V^*, E^*)\) with

\[ E^* := \{ (\mathcal{P}^1, \mathcal{P}^2) \in V \times V \mid \mathcal{P}^2 = \mathcal{P}^1 \cup \{ \eta_k \} \text{ with } k > \max \{ i \mid \eta_i \in \mathcal{P}_1 \} \text{ and } \eta_k \not\subseteq \sum_{\eta_i \in \mathcal{P}_1} \eta_i \} \]

is sufficient to get all mm-pairs.

To solve problem OSTR the tree \((V^*, E^*)\) can be traversed using a breadth-first or depth-first strategy. For each vertex \(\mathcal{P}\) the relation \(\eta_0 := \sum_{\eta_i \in \mathcal{P}} \eta_i\) is calculated. If \(m(\eta_0) \cap \delta \subseteq \phi\) is true, then the mm-pair \((\eta_0, m(\eta_0))\) is a solution for OSTR. The costs

\[ \left\lfloor \log_2 \#S/m(\eta_0) \right\rfloor + \left\lfloor \log_2 \#S/\eta_0 \right\rfloor \text{ and } \left| \frac{\#S/m(\eta_0)}{\#S/\eta_0} - 1 \right| \]

are calculated for this pair and compared to the lowest costs obtained so far. Finally the solution with minimal costs is selected to realize the specification.

The specific requirements of problem OSTR provide another criterion to prune the search tree and make the basic procedure computationally more efficient.

**Lemma 2:** Let \(M = (S, I, O, \delta, \lambda)\) be a finite state machine, and let \((V^*, E^*)\) be the search tree defined above.

For \((\mathcal{P}_1, \mathcal{P}_2) \in E\) let \(\nu_1 := \sum_{\eta_i \in \mathcal{P}_1} \eta_i\) and \(\nu_2 := \sum_{\eta_i \in \mathcal{P}_2} \eta_i.\) If \(m(\eta_1) \cap \nu_1 \subseteq \phi,\) then \(m(\eta_2) \cap \nu_2 \subseteq \phi.\)

As a consequence of lemma 2, once a node \(\mathcal{P}\) in the searchtree with \(m(\eta_0) \cap \nu \subseteq \phi\) is reached, all of its successors have this property and the subtree rooted at \(\mathcal{P}\) can be discarded.

### 4 Experimental results

The algorithm for problem OSTR described in section 3 has been implemented as a depthfirst procedure and has been applied to the finite state machine benchmarks distributed for the International Workshop on Logic Synthesis ’93 [21]. For incompletely specified finite state machines don’t care transition were fixed to transitions with next state = present state and output don’t cares were set to zero. For 23 examples a nontrivial solution of OSTR (i.e. a solution different from \((\text{id}, \text{id}, \text{id})\)) could be found. Table 1 shows the results in more detail for some of these examples.

<table>
<thead>
<tr>
<th>Name</th>
<th>#S1</th>
<th>#S2</th>
<th>#FFs (conventional)</th>
<th>#FFs (OSTR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbara</td>
<td>10</td>
<td>7</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>bbsse*</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>beecount*</td>
<td>7</td>
<td>4</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>dk16</td>
<td>27</td>
<td>21</td>
<td>23</td>
<td>10</td>
</tr>
<tr>
<td>dk512</td>
<td>15</td>
<td>11</td>
<td>14</td>
<td>8</td>
</tr>
<tr>
<td>ex1*</td>
<td>20</td>
<td>17</td>
<td>19</td>
<td>10</td>
</tr>
<tr>
<td>mark1*</td>
<td>15</td>
<td>11</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>planet*</td>
<td>48</td>
<td>44</td>
<td>45</td>
<td>12</td>
</tr>
<tr>
<td>s1488</td>
<td>48</td>
<td>44</td>
<td>45</td>
<td>12</td>
</tr>
<tr>
<td>s1494</td>
<td>48</td>
<td>44</td>
<td>45</td>
<td>12</td>
</tr>
<tr>
<td>s208</td>
<td>18</td>
<td>10</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>shiftreg</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>sse*</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>tav</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>tbk</td>
<td>32</td>
<td>16</td>
<td>16</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 1: Results of depthfirst search procedure for problem OSTR (incompletely specified examples are marked with an asterisk).
columns 5 and 6 list the required number of flipflops for a conventional BIST and for an optimized BIST structure based on the presented approach. The results show that for eight examples the number of flipflops required for the optimized structure is less than for a conventional BIST. For shiftreg and tav even the lower bound $|S_1| \cdot |S_2| = |S_1|$ is achieved and the number of flipflops is reduced to 50%. In terms of hardware costs the gain is even higher, because the transparent register for the conventional solution is more costly to implement than the registers used for the presented approach.

Since the overall hardware costs for both alternatives can only be compared after state coding and logic minimization, a state coding algorithm is currently being developed which takes advantage of the self-testable decomposition. A prototype version of this algorithm followed by ESPRESSO was used to determine the required PLA area for the circuits $C_1$, $C_2$ and the output function $\lambda$ [5]. The PLA area for the circuit $C$ of a conventional implementation was determined by applying NOVA to the finite state machine specifications [24]. In both cases the PLA area was estimated by $(2i + o)p$, with $i$, $o$, $p$ denoting the number of inputs, the number of outputs and the number of product terms, respectively. Table 2 shows the results.

Table 2: Comparison of PLA areas for conventional implementations and optimized self-testable realizations.

<table>
<thead>
<tr>
<th>Name</th>
<th>area for $C_1$</th>
<th>area for $C_2$</th>
<th>area for $\lambda$</th>
<th>total area $\Sigma$</th>
<th>area for $C$</th>
<th>$\Sigma / C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbara</td>
<td>320</td>
<td>320</td>
<td>108</td>
<td>748</td>
<td>550</td>
<td>1.36</td>
</tr>
<tr>
<td>bbsse</td>
<td>351</td>
<td>252</td>
<td>546</td>
<td>1149</td>
<td>990</td>
<td>1.16</td>
</tr>
<tr>
<td>beecount</td>
<td>126</td>
<td>90</td>
<td>8</td>
<td>224</td>
<td>228</td>
<td>0.98</td>
</tr>
<tr>
<td>dk16</td>
<td>1045</td>
<td>931</td>
<td>432</td>
<td>2408</td>
<td>1584</td>
<td>1.52</td>
</tr>
<tr>
<td>dk512</td>
<td>224</td>
<td>182</td>
<td>114</td>
<td>520</td>
<td>532</td>
<td>1.61</td>
</tr>
<tr>
<td>ex1</td>
<td>1053</td>
<td>1161</td>
<td>1815</td>
<td>4029</td>
<td>2652</td>
<td>1.52</td>
</tr>
<tr>
<td>planet</td>
<td>2272</td>
<td>2304</td>
<td>3420</td>
<td>7996</td>
<td>4539</td>
<td>1.76</td>
</tr>
<tr>
<td>s1488</td>
<td>2738</td>
<td>2736</td>
<td>3355</td>
<td>8794</td>
<td>7473</td>
<td>1.18</td>
</tr>
<tr>
<td>s1494</td>
<td>2808</td>
<td>2736</td>
<td>3528</td>
<td>9072</td>
<td>7897</td>
<td>1.15</td>
</tr>
<tr>
<td>s208</td>
<td>176</td>
<td>176</td>
<td>210</td>
<td>562</td>
<td>780</td>
<td>0.72</td>
</tr>
<tr>
<td>shifreg</td>
<td>3</td>
<td>12</td>
<td>3</td>
<td>18</td>
<td>96</td>
<td>0.19</td>
</tr>
<tr>
<td>sse</td>
<td>351</td>
<td>252</td>
<td>546</td>
<td>1149</td>
<td>990</td>
<td>1.16</td>
</tr>
<tr>
<td>tav</td>
<td>3</td>
<td>3</td>
<td>128</td>
<td>134</td>
<td>198</td>
<td>0.68</td>
</tr>
</tbody>
</table>

Table 3 compares the computational effort for the presented algorithm and the basic algorithm used in [18]. For both procedures the size $|V|$ of the search tree and the number of the nodes that have to be investigated to find the optimal solution are listed.

Table 3: Computational effort for exhaustive search.

| Name     | $1S_1$ | $\log_2 |V|$ | # nodes investigated $\log_2 |V|$ [18] | # nodes investigated [18] |
|----------|--------|--------|-----------------|------------------|
| bbara    | 10     | 7      | 19              | 43               | 815             |
| bbsse    | 16     | 43     | timeout         | -                | -               |
| beecount | 7      | 15     | 85              | -                | -               |
| dk16     | 27     | 20     | 15437           | 206              | 337041          |
| dk512    | 15     | 15     | 221             | 56               | 343853          |
| ex1      | 20     | 10     | 53              | -                | -               |
| mark1    | 15     | 45     | timeout         | -                | -               |
| planet   | 48     | 9      | 75              | -                | -               |
| s1488    | 48     | 9      | 75              | -                | -               |
| s1494    | 48     | 9      | 75              | -                | -               |
| s208     | 18     | 28     | 255             | -                | -               |
| shifreg  | 8      | 17     | 811             | 7                | 49              |
| tav      | 4      | 5      | 21              | 7                | 47              |
| tbk      | 32     | 11     | 55              | -                | -               |

The results confirm that the presented approach is able to characterize the search space significantly better than the procedure used in [18]. For example for dk16 the search tree can be reduced from $2^{206}$ to $2^{20}$ nodes. Moreover, the impact of lemma 2 on the computational effort can be seen clearly. For almost all examples the optimal solution could be found by investigating only a small number of nodes. The limit of 500 000 nodes (timeout) was reached only by bbsse, mark1 and sse. But as table 1 indicates, for these examples a large number of possible solutions could be found before the limit was reached. All examples which did not provide a non-trivial solution could be identified quickly.
5 Conclusions

Pipeline-like controller structures implement the states of the specification by two different multi-functional system registers. A self-test can be performed in two sessions without any extra hardware by alternatingly using the registers for test pattern generation and signature analysis. This architecture increases the fault coverage as well as the system performance.

An efficient algorithm has been presented to generate minimal pipelined realizations from state transition diagrams. The proposed algorithm relies on algebraic structure theory to construct realizations which can be trivially implemented by a self-testable structure. The efficiency of the procedure is ensured by a very precise characterization of the space of suitable realizations, the theoretical basis of which is the newly introduced concept of mm-pairs.

The experimental results show that in general a shorter critical path and thus a higher performance can be expected. In many cases the number of flipflops is less than the respective number for a conventional BIST, and in some cases even the total PLA area is reduced. This confirms that not only higher speed and fault coverage can be obtained this way, but also area can be saved.

References

14 R. Gage: Structured CBIST in ASICS; Proc. IEEE Int. Test Conf., Baltimore, Maryland, 1993, pp. 332-338
21 K. McElvain: IWLS'93 Benchmark Set: Version 4.0, distributed as part of the IWLS'93 benchmark distribution

7