Abstract
This paper presents algorithms for traversing product machines which improve on [6] of a speedup ranging from 3 up to 6. New features include a model that generalizes the product machine, resulting in simpler and more efficient representations and computations, as well as optimizations in symbolic image computation. In the latter case, the speed-up ranges from 1.5 to 4.

1 Introduction
A major class of hardware devices is represented by Finite State Machines (FSMs) that are in extensive use as building blocks in a variety of applications, ranging from VLSI circuits to network controllers. Many ASICs are “control-dominated”, i.e., they are best modeled in terms of FSMs, rather than of data path and control part. As high-level synthesis becomes increasingly popular, but nevertheless does not guarantee a priori functional correctness, formal proof of equivalence between implementations of the same specification is an important means for the designer to increase confidence in the tools. Moving beyond the design phase, sequential Automatic Test Pattern Generation (ATPG) remains a challenging issue, as few companies have the convenience to impose rigid scan design rules. An important application of equivalence proof is the exact identification of untestable faults [7].

The theoretical framework for proving the equivalence of two FSM’s consists in building and traversing their product machine [14]. Storing all or part of the product machine for later traversal [11] requires a huge amount of memory and is thus applicable only to small circuits. Explicit enumeration techniques to build it are computationally expensive [4, 13]. Two aspects must be considered if real circuits are to be handled: efficient ways to represent boolean functions and symbolic manipulations that allow to tame state explosion. Binary Decision Diagrams (BDD’s) [3, 2] are currently the most popular answer to the first question, whereas symbolic state exploration techniques seem to be the solution to the second problem [9].

The use of symbolic techniques for traversing FSM’s was first reported in [8] as a breadth-first implicit enumeration of the states that are reachable by a machine starting from a set of reset states. The same research group at Bull developed a technique for symbolic image computation resorting to characteristic functions to represent sets of states. Working on characteristic functions, they proposed an algorithm to find the image of a set of states, i.e., the set of future states. They introduced the concept of function simplification under a constraint and algorithms that convert the representation of a set of states between its characteristic function and its implicit definition as image of a boolean functional vector. By computing the transitive closure of the image relation, they find the set of reachable states, in which to look for a state where a different behavior can be observed on the machines’ primary outputs. Recent advances propose an efficient way of simplifying functions and computing images, with some heuristics based on domain or co-domain partitioning [10].

Extending this method, [6] propose some heuristics to reduce the complexity. In particular, they define a hash table to avoid redundant image computations and they devise a simplification strategy based on PODEM-like forward implications, together with exhaustive checks for special cases. Experimental results on ISCAS’89 circuits [1] are given.

All these methods are based on the exploration of the product machine by means of symbolic computations. None of them, to the best of our knowledge, addresses the issue of optimizing the traversal of the product machine, trying to keep its complexity as near as possible to the traversal of each single machine. Empirical evidence shows that, in most cases, the two machines that compose the product one do not differ significantly, thus this similarity can be exploited to reduce the computational burden. In this paper we present a new model, the General Product Machine (GPM), that simplifies the search for all reachable states. This model benefits from some known relation between a state of the first machine and a corresponding state of the second one. In the case of ATPG this relation is a simple structural correspondence between the good and faulty machines’ state elements. In the case of equivalence proof of FSM’s, the assumption of known state assignments is justified, being a common sub-product of the synthesis and optimization processes.

Section 2 presents the model of the General Product Machine, Section 3 describes the algorithms used in the symbolic traversal of the state space and the advantages that are obtained by using the aforementioned model; Section 4 shows experimental data on the ISCAS’89 benchmark set that allow a comparison
with similar approaches and justify this work. Section 5 eventually draws some conclusions.

2 The General Product Machine

Suppose that for two completely specified FSM's $M = \{I, O, S, A, \lambda, S_0\}$ and $M' = \{I, O, S', A', \lambda', S'_0\}$ a correspondence between the state spaces $S$ and $S'$ exists. In order to compare elements $s \in S$ and $s' \in S'$, we define two functions, $\psi$ and $\varphi$, that map $s$ and $s'$ in a common image space.

A subset $E \subseteq S \times S'$ is defined as:

$$E = \{(s, s') \in S \times S' : \psi(s) = \psi(s')\}.$$

If reachable by the product machine $M \times M'$ belong to $E$, then it is possible to find them a simple and efficient representation.

We propose to express the product machine $M \times M'$ in terms of a new state space, in which the condition $(s, s') \in E$ can easily be stated. Such a new space is defined through a change in the coordinate system. A system transformation from space $S \times S'$, described by variables $s$ and $s'$, to space $S \times D$, with variables $s$ and $d$, is defined by a transformation function $T : S \times S' \rightarrow D$, with the property that, given any couple $(s, d)$, there exists a unique $s'$ satisfying $d = T(s, s')$.

The functions $\psi$ and $\varphi$ can be interpreted as suitable mappings that convert the values of $s \in S$ and $s' \in S'$ to a common space in which they can be compared. A good choice for $T$ is the difference between the reachable states $s$ and $s'$ by the product machine $M \times M'$ defined as:

$$(s, d) = (s, T(s, s')) = (s, s' - s).$$

The existence and uniqueness of $s'$ define a new function $T : S \times D \rightarrow S'$, i.e., the partial inverse of $T$ with respect to $s'$.

The state transition function $\Delta$ imposes a necessary condition on function $\psi$. In fact:

$$\delta(s, s') = T(s, s') = \psi(s) \oplus \varphi(s')$$

where $\oplus$ denotes the XOR operation. In order to have a suitable $T$, $\varphi$ must be invertible, while no restriction is set on $\psi$. As a corollary, we derive that $\varphi(s')$ must be a one-to-one mapping, thus spaces $D$ and $S'$ must have the same dimension. On the other hand, $\psi(s)$ is an arbitrary function, therefore $S$ may have a cardinality other than $D$. We may conclude that, as this model supports the comparison of FSM's with different state space cardinalities, it takes into account many-to-one relations between $S$ and $S'$.

In ATPG applications, we choose $\psi(s) = s$ and $\varphi(s') = s'$, so $d$ records the information about which memory elements of the fault-free and faulty machines differ. In the verification of a resynthesis or optimization process, $\psi$ can take into account redundant state suppression and it will not be an injective function. $\varphi$ reflects different state assignments and will not be a one-to-one mapping, thus invertible. Our experiments show that the ISCAS'89 pairs of equivalent circuits have state assignments differing only for a permutation of their state variables.

We can now define the General Product Machine (GPM) $M_\Delta$ of two FSM's $M$ and $M'$ in connection with the transformation $T$ (Fig. 1):

$$(s, d) = T(s, s')$$

where:

$$\delta_d : S \times D \rightarrow D,$$

$$(s, d) = T(s, s') = \psi(s) \oplus \varphi(s').$$

The equivalence $M \cong M'$ is expressed as:

$$\forall(s, d) \in R_\Delta, \forall z \in I, T((s, d), z) = 0$$

The GPM model is more general than the product machine, in fact we can express the latter as a particular case of the former, by choosing $\psi(s) = 0$ and $\varphi(s') = s'$. The newly introduced $T$ acts as a further degree of freedom that can be exploited to reduce the complexity of the representation. In particular, we can state that: "if some one-to-one relation $s' \rightarrow s'$ holds on the set of reachable states of the product machine $M \times M'$, then there exists some choice of $T$, such that over the entire set $R_\Delta$ of states reachable by the GPM, we have $d = 0."$ The aforementioned result indicates that we can really simplify the representation of $R_\Delta$, whenever the previous assumption holds. The model also deals with reachable states not included in $E$ without introducing additional complexity.

3 Algorithms

Following the symbolic state exploration technique of [8] and using the symbols proposed in [5], we define an operator, called generalized cofactor and image restrictor, that will be extensively used to simplify functions and to compute images. Given a vectorial boolean function $F : X \rightarrow Y$, a characteristic function $c : x \rightarrow B$, and domain values $x \in X$, we define the operation $F \downarrow c$ as a new vectorial boolean function.

![Product Machine](image-url)
The term "1" in (2) is the characteristic function of where y is some arbitrarily chosen element of the image of on-set of choosing y for many undetectable faults in ATPG applications. The characteristic function of those states. The operator BETWEEN returns the simplest BDD such that:

\[ \text{BETWEEN}(\text{New}, \text{Reached}) \subseteq \text{Reached}. \]

State-of-the-art image computation In the algorithm of Fig. 2 the heaviest operation is the computation of the image of the vectorial function \( \Delta \) on the set From.

The algorithm we adopt [9] is based on the recursive Shannon expansion of the image characteristic function. If \( F = \{ F_1, ..., F_n \} \), we define \( F_k = \{ F_k, F_{k+1}, ..., F_n \} \). We can write:

\[ \text{IMG}(F_k, 1) = \{ 0 \} \times \text{IMG}(F_{k+1} \downarrow f_k, 1) \]

In image computation, according to (2), two steps can be identified:

1. operation \( \Delta' = \Delta \downarrow \text{From} \): the vectorial boolean function \( \Delta \) is simplified by a generalized cofactor operation which has also the effect of restricting the full image of \( \Delta \) to its image over the set From;
2. Image = \( \text{IMG}(\Delta', 1) \) is the recursive expansion algorithm of (3).

The following techniques have been proposed in order to speed up image computation:

- domain partitioning: [10] use a divide and conquer strategy in which the domain From is decomposed in smaller parts, each contributing to a subset of the entire image. An alternative partitioning method is based on the decomposition of the co-domain;
- image cache table: a hash table [8] is used to store the results of image subproblems, avoiding redundant computations; it can be extended to recognize a problem despite the presence of negations and permutations;
- implications: the algorithms in [8] improve the termination conditions by taking into account implications among inputs and outputs, according to a PODEM-like technique [12];
- full partitioning: at every recursion step, vector \( F_k \) is possibly partitioned in two or more subsets consisting of functions with disjoint support variables; an effective optimization consists in calculating the subimages, and then taking their cartesian product;
- reached set pruning: whenever the image computation procedure enters a subspace entirely contained in the set of already reached states, recursion can stop; this check is done by traversing the Reached set in parallel to the recursive construction of Image.

Our approach can be characterized by several optimizations, some having already been introduced by the Bull and/or Boulder groups, some being novel. The following ones belong to the former class:

- recursion is based on co-domain partitioning;
- the image is pruned over the set of reachable states;
- generalized cofactor operations take into account implications among variables;
- the \( F_k \) vector is fully partitioned at each step according to its support variables;
- no image table is used.

Novel features are the presence of a \textit{cofactor hash table} which records the results of past cofactorizations and the \textit{subtree recombination check} whenever \( F_{k+1} \uparrow f_k \) equals \( F_{k+1} \uparrow f_k \) on the current subspace, we recognize subtree recombination, thus saving a recursion.
The contribution of the GPM model to simplified image computation. Representation and computation for the product machine is inherently much more complex than for the single machine $M$, although empirical observations show that in most product machines the reachable states satisfy, at least approximately, the relation $\psi(s) = \psi(s')$. This has major consequences on the BDD's representing the sets $\text{From}$, $\text{Reached}$, and $\text{New}$ in the $S \times S'$ space and on the complexity of the image computation algorithm.

Let us assume a simple relation: $s = s'$ to make the comprehension of the results easier and to give pictorial examples. Note, anyway, that the results drawn from the following considerations are general.

Whenever $s = s'$ holds, a BDD representation that doesn't take this explicitly into account will be necessarily redundant. In fact, for every couple of components $s_i$ and $s'_i$ of the state vectors $s$ and $s'$, the BDD reflects that $s_i = s'_i$. In particular, the subtree rooted at $s_i = 0$ must contain a node assessing $s'_i = 0$, and the subtree rooted at $s_i = 1$ will contain $s'_i = 1$ (Fig. 3). We notice that both $s_i$ and $s'_i$ must explicitly appear on every path from the BDD root to any "1" terminal node. Moreover, as the two subtrees of a $s_i$ node are distinct at least in their $s'_i$ children, they will never be shared. The problems outlined above may be reduced by choosing a proper variable ordering, for example $s_1$, $s_2$, $s_3$, ..., but we can't find a suitable order in the general $\psi(s) = \psi(s')$ case: we really need a model exploiting the known relation instead of being conditioned by it.

The above paragraph showed the inherent complexity of sets $\text{From}$, $\text{Reached}$, and $\text{New}$ of the product machine $M \times M'$ compared with the one of the corresponding sets $\text{From}$, $\text{Reached}$, $\text{Image}$ and $\text{New}$ of the single machine $M$. In particular, the representation of $\text{From}$ will influence the run-time performance in two aspects:

1. the cofactoring $(\delta, \delta') \upharpoonright \text{From}$ is much harder to compute than $\delta \upharpoonright \text{From}$ due to the doubled number of $\delta$ components and to the complexity of $\text{From}$;
2. if we compare $\text{IM}(\delta, \delta') \upharpoonright \text{From}, 1)$ with $\text{IM}(\delta \upharpoonright \text{From}, 1)$, we see that in the many cases in which $\delta$ and $\delta'$ don't differ significantly, we are performing several redundant computations:
   - $\delta \upharpoonright \text{From}$ and $\delta' \upharpoonright \text{From}$ share many identical subproblems: without the help of a cofactor table, we would double the cofactor operations;
   - while applying full partitioning, we must discover that $\delta$ and $\delta'$ have disjoint state variable sets and that most of the support variable dependencies present in $\delta$ are replicated in $\delta'$: the partitioning will be significantly slowed;
   - the same arguments also apply to the propagation of implications: whenever an implication concerning $\delta$ is present also in $\delta'$, we should apply it twice.

We conclude that the product machine representation leads to an increase in CPU time of at least two with respect to single machine traversal, as confirmed by experimental results. We may overcome this limit only by explicitly taking into account the similarities between $M$ and $M'$. The GPM model helps in simplifying representation and computation as its $\delta_2$ function records the behavioral difference of $M'$ with respect to $M$: we are explicitly factoring out the similarities, i.e., reducing redundancy.

Let us suppose to build the GPM $M_\Delta$ from a coordinate system transformation $T(s, s') = \psi(s) \oplus \psi(s')$. We expect a gain in efficiency directly related to the degree of exactness of the mapping, we may write: $\Delta \upharpoonright \text{From}$ and $\text{IMG}(\Delta \upharpoonright \text{From})$ boils down to a problem comparable to single machine traversal.

3.1 Representation of From

Over the entire set of states for which $\psi(s) = \psi(s')$ we have $d = 0$: the condition $s_i = s'_i$ becomes $d_i = 0$ (Fig. 3). In Fig. 4 we see the mapping between each node $(a)$ of the single machine's characteristic functions and the corresponding nodes in the product machine $(b)$ and in the General Product Machine $(c)$, when we choose an interleaved ordering of variables and $s_i = s'_i$ holds. The presence of $d$ variables is structurally simple, showing that every subtree sharing in $(a)$ will be present in $(c)$. Exceptions to this simple BDD structure are due to the states $(s,d) \in \text{From}$ for which $d \neq 0$, therefore the gain in representation is directly related to the degree of exactness of the assumed mapping.

Recalling that $\Delta$ contains all the $(s,d)$ states satisfying the mapping, we may write:

\[
\text{From} = ((\text{From} \upharpoonright \Delta) \cap \Delta) \cup ((\text{From} \upharpoonright \Delta) \cap \Delta')
\]

(4)

where $\text{From} \upharpoonright \Delta$ is described only in terms of $s$ variables. As shown in (4), From consists of two subsets, one included in $\Delta$ and the other outside $\Delta$. In most cases the second term is the empty set, thus we will
benefit of domain partitioning techniques on the set $\mathcal{E}$.

3.2 Computation of $\Delta$ and $\Lambda$

According to the definition of the GPM, given a transformation function $T$, the state transition and output functions $\Delta$ and $\Lambda$ are expressed by means of simple function composition of corresponding $M$ and $M'$ functions. The overhead imposed by these compositions is negligible if compared with the computational simplifications that the GPM model supports, provided that a suitable ordering for the variables is used, e.g., $d_1, s_1, d_2, s_2, \ldots$.

3.3 Computation of $\Delta \uparrow \text{From}$

The $\Delta = (\delta, \delta')$ function of the GPM $M_\Delta$ is of a certain degree more complex than the product machine state transition function $\delta, \delta'$: as a result of the compositions, the $\delta_4$ components of $\Delta$ depend on all the state variables $s$ and $d$, while $\delta'$ clearly contains only $s$, not $s$. We might therefore expect that $\delta_4 \uparrow \text{From}$ will be harder to compute than $\delta' \uparrow \text{From}$. In order to overcome this limit, we often explore the $\mathcal{E}$ subset of $\mathcal{S} \times \mathcal{S}'$, we are facing a From set such that From $\subseteq \mathcal{E}$. Using a property of the generalised cofactor operator we may write:

$$\text{From} \subseteq \mathcal{E} \Rightarrow \left\{ \begin{array}{l}
\text{From} = (\text{From} \uparrow \mathcal{E}) \cap \mathcal{E} \\
\Delta \uparrow \text{From} = (\Delta \uparrow \mathcal{E}) \uparrow (\text{From} \uparrow \mathcal{E})
\end{array} \right.$$  

The term $\Delta \uparrow \mathcal{E}$ can be computed in advance and its complexity is comparable to that of the original $\delta$ and $\delta'$ functions, as it doesn't contain $d$ variables. This simple check can reduce the computation of $\Delta \uparrow \text{From}$ to a problem comparable to single machine traversal.

3.4 Computation of $\text{IMG}(\Delta \uparrow \text{From}, 1)$

While discussing the representational power of the GPM model, we focused on the fact that the characteristic functions of From, New and Reached have a high degree of subimage sharing compared to the product machine. We devised a first heuristic, the subtree recombination check, to avoid computing two sub-images when we already know they are equal: this technique constantly reduced the number of recursive calls, as shown in the following section.

A greater simplification is due to the easy way of checking if $\text{Image} \subseteq \mathcal{E}$ before computing it. In particular:

$$\delta_4 \uparrow \text{From} = 0 \Rightarrow \text{IMG}(\Delta \uparrow \text{From}, 1) \subseteq \mathcal{E}$$

When the previous relation is satisfied, we may compute:

$$\text{Image} = \text{IMG}(\delta \uparrow (\text{From} \uparrow \mathcal{E}) \cap \mathcal{E}) \uparrow \mathcal{E}$$

meaning that we simply compute the image of the original $\delta$ function over a set obtained from From by suppressing the $d$ variables, and then append to the resulting BDD the nodes corresponding to $d = 0$.

The previous techniques work whenever the From set is completely contained in $\mathcal{E}$. We can extend their effectiveness by first decomposing From in the part contained in $\mathcal{E}$ and the (usually smaller) remainder. This is a good application of the domain partitioning strategy, and can be described by the following identity:

$$\text{image} = \text{IMG}(\Delta \uparrow \mathcal{E}) \uparrow (\text{From} \uparrow \mathcal{E}), 1) \cup \text{IMG}(\Delta \uparrow \mathcal{E}) \uparrow (\text{From} \uparrow \mathcal{E}), 1)$$  

The concepts presented in the above sections were implemented in an in-house software package written in C (about 4000 lines of source code), including:

- elementary BDD operations, based on the canonical form and ITE operators presented in [2];
- generation of $\delta$ and $\lambda$ functions from circuit netlist to BDD format; some variable ordering heuristics are included, available in the literature;
- reachable state computation, through symbolic breadth-first execution;
- product machine and GPM verification, as described in the previous section.

Experiments were performed on a subset of the ISCAS'89 circuits, with two objectives: to measure the performance of the state generation module in single machine processing, and to compare the verification of couples of equivalent machines to the above case.

The results obtained on a 3-Mips VAXstation 3100 are summarized in two tables. Tab. 1 shows the state generation module: numbers of image computation recursions (IMG) and CPU times (in seconds) are reported without and with the cofactor-table and subtree recombination optimisations. The speed-up with respect to [6] ranges from 1.5 to 4. Tab. 2 shows verification data for the seven couples of equivalent ISCAS'89 circuits: product machine (PM) traversal without and with cofactor table, and GPM traversal are described. The first case is two to three times slower than state generation for the single machine (as in [6]), the second one has a better performance due to the cofactor table and the last one has very little overhead, compared to single machine processing. The speed-up of the optimized case with respect to [6] ranges from 3 to 6.
fying synthesized synchronous circuits and may be of considerable help in ATPG, e.g., for redundant fault identification. Their limits are due to the complexity of the circuits and to the number of states, but, as the same applies for synthesis, this is yet one more proof of the parallel and interleaved evolutions of verification and synthesis.

Table 1 – Reachable state set computation

<table>
<thead>
<tr>
<th>Circuit pairs</th>
<th>PM not opt</th>
<th>PM opt</th>
<th>GPM opt</th>
</tr>
</thead>
<tbody>
<tr>
<td>s361-s362</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s363-s365</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s366-s368</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s369-s371</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s372-s374</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s375-s377</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s378-s380</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s381-s383</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s384-s386</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s387-s389</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s390-s392</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s393-s395</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s396-s398</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s399-s401</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s402-s404</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s405-s407</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s408-s410</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s411-s413</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s414-s416</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s417-s419</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s420-s422</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s423-s425</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s426-s428</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s429-s431</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s432-s434</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s435-s437</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s438-s440</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s441-s443</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s444-s446</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>s447-s449</td>
<td>111</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 2 – Equivalence verification

5 Conclusions

The main theoretical contribution of this paper is a generalisation of the product machine model, defining a suitable reference frame transformation, a General Product Machine (GPM) can be obtained, that entails considerable simplifications in representation and computation. This model covers the traditional product machines as a particular case.

Optimized forward-time processing symbolic image computation procedures allow us to obtain a speed-up ranging from 1.5 to 4 with respect to the experimental results of [6] on the same set of ISCAS '89 circuits [1]. The combined use of such techniques and of the GPM model yields a speed-up ranging from 3 to 6 for FSM verification.

Symbolic FSM and product machine traversal techniques are thus a suitable answer to the need for verifying synthesized synchronous circuits and may be of considerable help in ATPG, e.g., for redundant fault identification. Their limits are due to the complexity of the circuits and to the number of states, but, as the same applies for synthesis, this is yet one more proof of the parallel and interleaved evolutions of verification and synthesis.

References