Decomposition and Technology Mapping of Speed-Independent Circuits Using Boolean Relations

Jordi Cortadella, *Member, IEEE,* Michael Kishinevsky, *Senior Member, IEEE,* Alex Kondratyev, *Senior Member, IEEE,* Luciano Lavagno, *Member, IEEE,* Enric Pastor, and Alexandre Yakovlev, *Member, IEEE*

***Abstract*—This paper presents a new technique for decomposition and technology mapping of speed-independent circuits. An initial circuit implementation is obtained in the form of a netlist of complex gates, which may not be available in the design library. The proposed method iteratively performs *Boolean decomposition* of each such gate F into a two-input combinational or sequential gate G available in the library and two gates H_1 and H_2 simpler than F, while preserving the original behavior and speed-independence of the circuit. To extract functions for H_1 and H_2 the method uses *Boolean relations* as opposed to the less powerful algebraic factorization approach used in previous methods. After logic decomposition, the overall library matching and optimization is carried out. Logic resynthesis, performed after speed-independent signal insertion for H_1 and H_2, allows for sharing of decomposed logic. Overall, this method is more general than the existing techniques based on restricted decomposition architectures, and thereby leads to better results in technology mapping.**

***Index Terms*—Asynchronous circuits, Boolean relations (BR's), logic decomposition, speed independence, technology mapping.**

I. Introduction

SPEED-INDEPENDENT circuits, originating from Muller's work [13], are *hazard-free under the unbounded gate delay model*. With the recent progress in developing efficient analysis and synthesis techniques, supported by computer-aided design (CAD) tools, the implementability of this subclass of circuits has become significantly more practical, bearing in mind the advantages of speed-independent designs, such as their greater temporal robustness and self-checking properties.

The basic ideas about synthesis of speed-independent circuits from event-based models, such as signal transition graphs (STG's) and change diagrams, are described, e.g., in [4], [6], and [10]. They provide general conditions for logic implementability of specifications into *complex gates* with arbitrary fanin and internal feedback.

To achieve greater practicality, synthesis of speed-independent circuits has to rely on more realistic assumptions about implementation logic. Thus, more recent work has been focused on the development of logic decomposition techniques. It falls into two categories. One of them includes attempts to achieve logic decomposition through the use of *standard architectures*, such as the *standard-C* architecture mentioned below. The other group comprises work targeting the decomposition of complex gates directly, by finding a behavior-preserving interconnection of simpler gates. In both cases, the major functional issue, in addition to logic simplification, is that the decomposed logic must not violate the original *speed-independent specification*. This criterion makes the entire body of research in logic decomposition and technology mapping for speed-independent circuits quite specific compared with its synchronous counterparts.

Two examples of the first category [1], [9] present initial attempts to move from complex gates to a more structured implementation. The basic circuit architecture includes *C* elements, acting as latches, and combinational logic, responsible for the computation of the excitation functions for the latches. This logic is assumed to consist of *AND* gates with potentially unbounded fanin and unlimited input inversions and bounded fanin *OR* gates. Necessary and sufficient conditions for implementability of circuits in such an architecture, called the *standard-C* architecture, have been formulated in [1], [9]. They are called *monotonic cover* (MC) requirements. The intuitive objective of the MC conditions is to make the first level (AND) gates work in a *one-hot* fashion with acknowledgment through one of the C-elements. Following this approach, various methods for speed-independent decomposition and technology mapping into *implementable* libraries have been developed, e.g., in [16] and [8]. The former method only decomposes existing gates (e.g., a three-input *AND* into two two-input *AND*'s), without any further search of the implementation space. The latter method extends the decomposition to more complex (algebraic) divisors, but does not tackle the limitation inherent in the initial MC architecture.

The best representative of the second category appears to be the work of Burns [3]. It provides general conditions for speed-independent decomposition of complex (sequential) elements into two sequential elements, or a sequential and a combina-

Manuscript received December 16, 1997; revised July 31, 1998. This work was supported by the ACiD-WG under Grant ESPRIT 21949, CICYT TIC 98-0410-C02-01 and 98-0949-C02-01, by Acción integrada U.K. under Grant HB1997-0208, by the British Council Spain MDR under Grant 2463 (1998/99), and by EPSRC under Grant GR/K70175/L24038. This paper was recommended by Associate Editor F. Brglez.

J. Cortadella is with the Department of Software, Universitat Politécnica de Catalunya, 08034 Barcelona, Spain.

M. Kishinevsky is with the Strategic CAD Lab, Intel Corporation, Hillsboro OR 97124 USA.

A. Kondratyev is with the Department of Computer Hardware, The University of Aizu, 965-80 Aizu-Wakamatsu, Japan.

L. Lavagno is with the DIEGM, Universitá di Udine, 33100 Udine, Italy.

E. Pastor is with the Department of Computer Architecture, Universitat Politécnica de Catalunya, 08034 Barcelona, Spain.

A. Yakovlev is with the Department of Computing Science, University of Newcastle upon Tyne, NE1 7RU Newcastle upon Tyne, U.K.

Publisher Item Identifier S 0278-0070(99)06221-1.

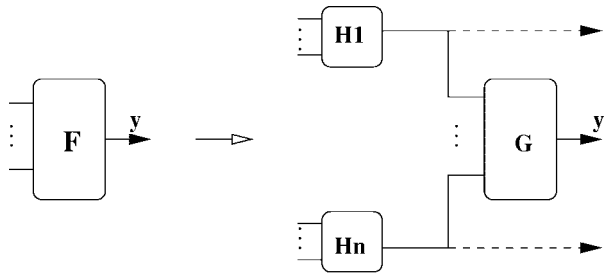


Fig. 1. General framework for speed-independent decomposition.

tional element. Notably, these conditions are analyzed using the original unexpanded behavioral model, thus improving the efficiency of the method. This work is, in our opinion, a big step in the right direction, but addresses mainly correctness issues. It does not describe how to use the efficient correctness checks in an optimization loop, and does not allow the sharing of a decomposed gate by different signal networks. The latter issues were successfully resolved in [8], but only within a standard architecture approach.

Technology mapping for circuits working in fundamental mode [17] can be achieved by deriving a hazard-free two-level sum-of-products [14] and obtaining a multilevel form by hazard-nonincreasing transformations [18]. However, these transformations cannot be generally applied for the decomposition of speed-independent circuits without introducing new hazards.

In [15], technology mapping for speed-independent circuits is done by merely identifying sets of simple logic gates that can be implemented as a complex gate, but no logic decomposition is performed when a function cannot be implemented as a complex gate.

In our present work, we are considering a more general framework which allows the use of *arbitrary gates and latches* available in the library to decompose a complex gate function, as shown in Fig. 1. In that respect, we are effectively making progress toward the more flexible second approach. The basic idea of this new method is as follows.

An initial complex gate is characterized by its function F. The result of decomposition is a library component designated by G and a set of (possibly still complex) gates labeled H_1, H_2, ..., H_n. The latter are decomposed recursively until all elements are found in the library and optimized to achieve the lowest possible cost. We, thus, by and large put no restrictions on the implementation architecture in this work. However, as will be seen further, for the sake of practical efficiency, our implemented procedure deals only with the two-input gates and/or latches to act as G-elements in the decomposition. The second important change of this work compared to [8] is that the new method is based on a full scale Boolean decomposition rather than just on algebraic factorization. This allows us to widen the scope of implementable solutions and improve on area cost. Future work will tackle performance-oriented decomposition.

Our second goal in generalizing the C-element-based decomposition has been to allow the designer to use more *conventional types of latches*, e.g., D-latches and SR-latches, instead of C-elements that may not exist in conventional

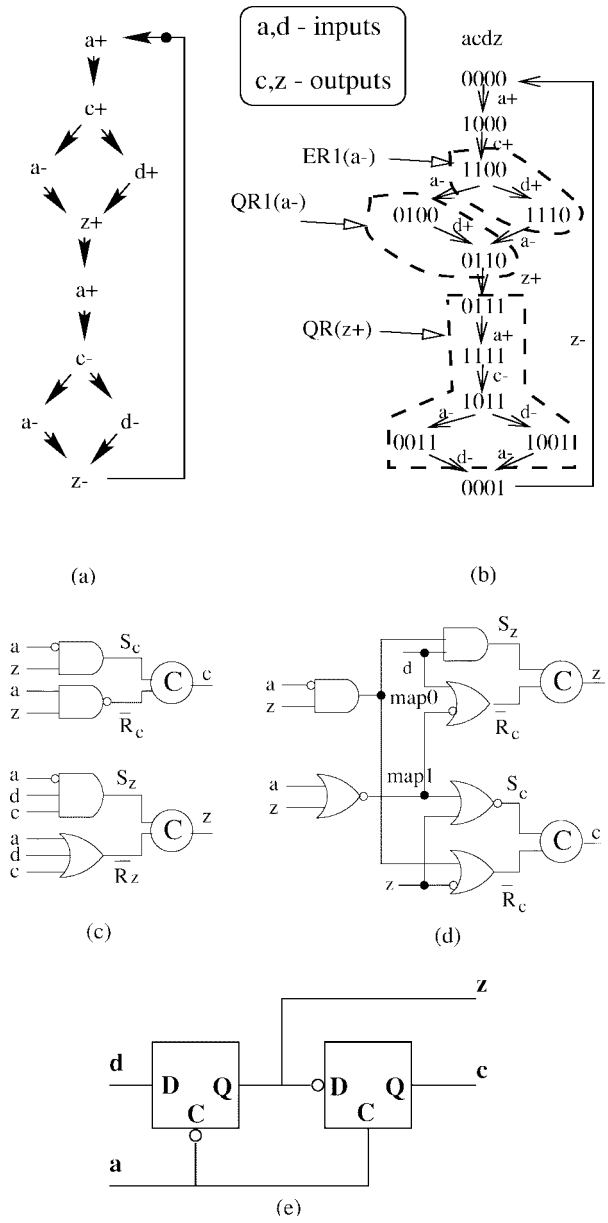


Fig. 2. An example of (a) STG, (b) SG, and (c)–(e) their implementation (benchmark `hazard.g`).

standard-cell libraries. Furthermore, as our experimental results show (see Section VI), in many cases the use of standard latches instead of C-elements helps improving the circuit implementations considerably.

The power of this new method can be appreciated by looking at the example `hazard.g` taken from a set of asynchronous benchmarks. The original STG specification and its state graph are shown in Fig. 2(a) and (b). The initial implementation using the "standard C-architecture" and its decomposition using two input gates by the method described in [7] are shown in Fig. 2(c) and (d). Our new method produces a much cheaper solution with just two D-latches, shown in Fig. 2(e). Despite the apparent triviality (for an experienced human designer!) of this solution, none of the previously existing automated tools have been able to obtain it. Also note that the D-latches are used in a *speed-independent*

fashion, and are thus free from meta-stability and hazard problems.[1]

The paper is organized as follows. Section II introduces the main theoretical concepts and notation. Section III presents an overview of the method. Section IV describes the major aspects of our Boolean relation (BR)-based decomposition technique in more detail. Section V briefly describes its algorithmic implementation. Experimental results are presented in Section VI, which is followed by conclusions and ideas about further work.

II. Background

In this section we introduce theoretical concepts and notation required for our decomposition method. First, we define *state graphs* (SG's), which are used for logic synthesis of speed-independent circuits. The SG itself may of course be generated from a more compact, user-oriented model, such as the STG. The SG provides the logic synthesis procedure with all the information necessary for deriving Boolean functions for complex gates. Second, the SG is used for a property-preserving transformation, called *signal insertion*. The latter is performed when a complex gate is decomposed into smaller gates, and the new signals must be guaranteed to be speed-independent, i.e., hazard-free in input/output mode using the unbounded gate delay model.

A. State Graphs and Logic Implementability

An SG is a labeled directed graph whose nodes are called *states*. Each arc of an SG is labeled with an *event*, that is a rising $(a+)$ or falling $(a-)$ transition of a signal a in the specified circuit. We also allow notation $a*$ if we are not specific about the direction of the signal transition. Each state is labeled with a vector of signal values. An SG is *consistent* if its state labeling v: $S \rightarrow \{0,1\}^n$ is such that: in every transition sequence from the initial state, rising and falling transitions alternate for each signal. Fig. 2(b) shows the SG for the Signal Transition Graph in Fig. 2(a), which is consistent. We write $s \xrightarrow{a} (s \xrightarrow{a} s')$ if there is an arc from state s (to state s') labeled with a.

The set of all signals whose transitions label SG arcs are partitioned into a (possibly empty) set of inputs, which come from the environment, and a set of outputs or state signals that must be implemented. In addition to consistency, the following two properties of an SG are needed for their implementability in a speed-independent logic circuit.

The first property is *speed-independence*. It consists of three parts: determinism, commutativity and output-persistence. An SG is called *deterministic* if for each state s and each label a there can be at most one state s' such that $s \xrightarrow{a} s'$. An SG is called *commutative* if whenever two transitions can be executed from some state in any order, then their execution always leads to the same state, regardless of the order. An event a^* is called persistent in state s if it is enabled at s and remains enabled in any other state reachable from s by firing an event b^* different from a^*. An SG is called *output-persistent* if its output signal events are persistent in all states and no output signal event can disable input events. Any transformation, such as insertion of new signals for decomposition, if performed at the SG level, may affect all three properties.

[1] For example, all transitions on the input must be acknowledged by the output before the clock can fall and close the latch. Thus, there is no problem with setup and hold times as long as the propagation time from D to Q is larger than both setup and hold times, which is generally the case.

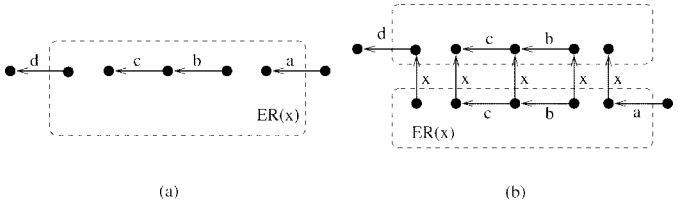


Fig. 3. Event insertion scheme: (a) before insertion and (b) after insertion.

The second requirement, *complete state coding* (CSC), is a necessary and sufficient condition for the existence of a logic circuit implementation. A consistent SG satisfies the CSC property if for every pair of states s, s' such that $v(s) = v(s')$, the set of output events enabled in both states is the same (the SG in Fig. 2(b) is *output-persistent* and has CSC). CSC does not however restrict the type of logic function implementing each signal. It requires that each signal is cast into a *single atomic* gate. The complexity of such a gate can however go beyond that provided a concrete library or technology.

The concepts of excitation regions and quiescent regions are essential for transformation of SG's, in particular for inserting new signals into them. A set of states is called an *excitation region* (ER) for event a^* [denoted by $ER(a^*)$] if it is the set of states such that $s \in ER(a^*) \Leftrightarrow s \xrightarrow{a^*}$. The *quiescent region* (QR) [denoted by $QR(a^*)$] of a transition a^*, with excitation region $ER(a^*)$, is the set of states in which a is stable and keeps the same value, i.e., for $ER(a+)$ $(ER(a-))$, a is equal to one(zero) in $QR(a+)$ $(QR(a-))$. An example of an ER and two examples of QR's are shown in Fig. 2(b).

B. Property-Preserving Event Insertion

Our decomposition method is essentially behavioral—the creation of new signals at the structural (logic) level must be matched by an insertion of their transitions at the behavioral (SG) level. Event insertion is an operation on an SG which selects a subset of states, splits each of them into two states and creates, on the basis of these new states, an excitation region for a new event. Fig. 3 shows the chosen insertion scheme, analogous to that used by most authors in the area [19]. We shall say that an inserted signal a *is acknowledged* by a signal b, if b is one of the signals delayed by the insertion of a. The same terminology will be used for the corresponding transitions. For example, d acknowledges x in Fig. 3.

State signal insertion must preserve the *speed-independence* of the original specification. The events corresponding to an inserted signal x are denoted $x*$, $x+$, $x-$, or, if no confusion occurs, simply by x. Let A be a deterministic, commutative SG and let A' be the SG obtained from A by inserting event x. We say that an insertion state set $ER(x)$ in A is a *speed-independence preserving set (SIP-set)* iff: 1) for each event a in A, if a is persistent in A, then it remains persistent in

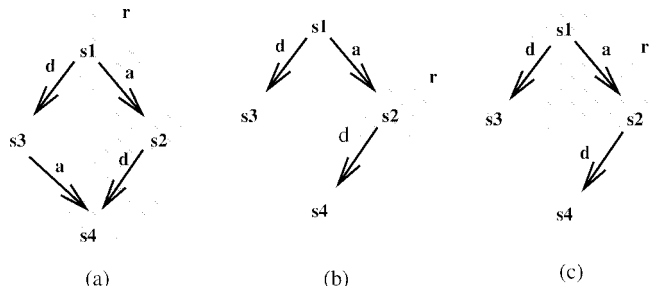


Fig. 4. Possible violations of SIP conditions.

A', and 2) A' is deterministic and commutative. The formal conditions for the set of states r to be a SIP-set can be given in terms of intersections of r with the so-called state diamonds of SG [5]. These conditions are illustrated by Fig. 4, where all possible cases of illegal intersections of r with state diamonds are shown. The first (rather inefficient) method for finding SIP-sets based on a reduction to the satisfiability problem was proposed in [19]. An efficient method based on the theory of regions has been described in [5].

Assume that the set of states S in an SG is partitioned into two subsets which are to be encoded by means of an additional signal. This new signal can be added in order to either satisfy the CSC condition, or to break up a complex gate into a set of smaller gates. In the latter case, the new signal represents the output of the intermediate gate added to the circuit. Let r and $\overline{r} = S - r$ denote the blocks of such a partition. For implementing such a partition we need to insert transitions of the new signals in the *border states* between r and $\overline{r}$.

The *input border* of a partition block r, denoted by $IB(r)$, is informally the subset of states of r by which r is entered. We call $IB(r)$ *well-formed* if there are no arcs leading from states in $r - IB(r)$ to states in $IB(r)$. If a new signal is inserted using an input border, which is not well-formed, then the consistency property is violated. Therefore, if an input border is not well formed, its well-formed speed-independent preserving closure can be constructed, as described by the algorithm presented in [7].

The insertion of a new signal can be formalized with the notion of *I-partition* ([19] used a similar definition). Given an SG, A, with a set of states S, an I-partition is a partition of S into four blocks: $\{ER(x+), QR(x+), ER(x-),$ and $QR(x-)\}$. $QR(x+)$ and $QR(x-)$ define the sets of states in which x will have the stable value one and zero, respectively. $ER(x+)$ and $ER(x-)$ define the excitation regions of x in the new SG A'. In order to distinguish between the sets of states for the excitation and quiescent regions of the inserted signal x in the original SG A and the new SG A', we will refer to them as $ER_A(x*)$, $ER_{A'}(x*)$, $QR_A(x*)$, and $QR_{A'}(x*)$, respectively. If the insertion of x preserves consistency and persistency, then the only transitions crossing boundaries of the blocks are the following: $QR_A(x-) \rightarrow ER_A(x+) \rightarrow QR_A(x+) \rightarrow ER_A(x-) \rightarrow QR_A(x-)$.

Example 1: Fig. 5 shows three different cases of the insertion of a new signal x into the SG for the `hazard.g` example. The insertion using $ER_A(x+)$ and $ER_A(x-)$ of Fig. 5(a) does not preserve speed-independence as the SIP set conditions are violated for $ER_A(x+)$ [a violation of the type shown in Fig. 4(b)].

When signal x is inserted with the excitation regions shown in Fig. 5(b) then its positive switching is acknowledged by transitions $a-$, $d+$, while its negative switching is acknowledged by transition $z-$. The corresponding excitation regions satisfy the SIP conditions and the new SG A', obtained after insertion of signal x, is shown in Fig. 5(b). Note that the acknowledgment of $x+$ by transitions $a-$, $d+$ results in delaying *some input signal* transitions in A' until $x+$ fires. This changes the original I/O interface for SG A, because it requires the environment to look at a new signal before it can change a and d. This is generally incorrect (unless we are also separately finding an implementation for the environment or we are working under appropriate timing assumptions), and hence this insertion is rejected.

The excitation regions $ER_A(x+)$ and $ER_A(x-)$ shown in Fig. 5(c) are SIP sets. They are well-formed and comply with the original I/O interface because positive and negative transitions of signal x are acknowledged only by output signal z. This shoice of insertion is thus valid.

C. Signal Insertion Guided by Function

In the case of logic decomposition, the insertion of new signals is guided by Boolean functions either obtained by algebraic division (as in [8]) or by Boolean decomposition of complex functions. We next summarize the method for function-guided signal insertion presented in [8] and also used in this paper. The method will be illustrated with the example of Fig. 6.

Given a function F, the set of states is initially partitioned into the blocks r and $\overline{r}$ corresponding to the states in which $F = 1$ and $F = 0$, respectively. In the example of Fig. 6, signal x corresponding to the function $F = c + d$ must be inserted. Fig. 6(a) depicts the partition of the states with regard to function F.

Next, the I-partition is calculated by defining the input borders of r and $\overline{r}$ and extending them until they become SIP sets. This extension is not unique and several solutions can be derived [5]. Fig. 6(b) depicts one of the solutions. The final I-partition defines $ER(x+)$, $ER(x-)$, $QR(x+)$, and $QR(x-)$.

Finally, the events $x+$ and $x-$ are inserted following the scheme depicted in Fig. 3. The final SG is shown in Fig. 6(c). The new signal x can now be implemented by the function $x = c + d$.

In the example, $z+$ is delayed by $x+$ whereas $z-$ is delayed by $x-$. Thus, signal x is acknowledged by z.

The insertion of a new signal x implies changes in the implementation of some other signals as well. In general:

- *signals delayed by* x change their implementations, since x becomes a new trigger signal for them and
- the implementation of the *signals not delayed by* x is valid in the new SG. However, it might be the case that simpler implementations are possible when the new signal x is included in their support. Hence x can contribute to the simplification of these "non-triggered" signals as well.

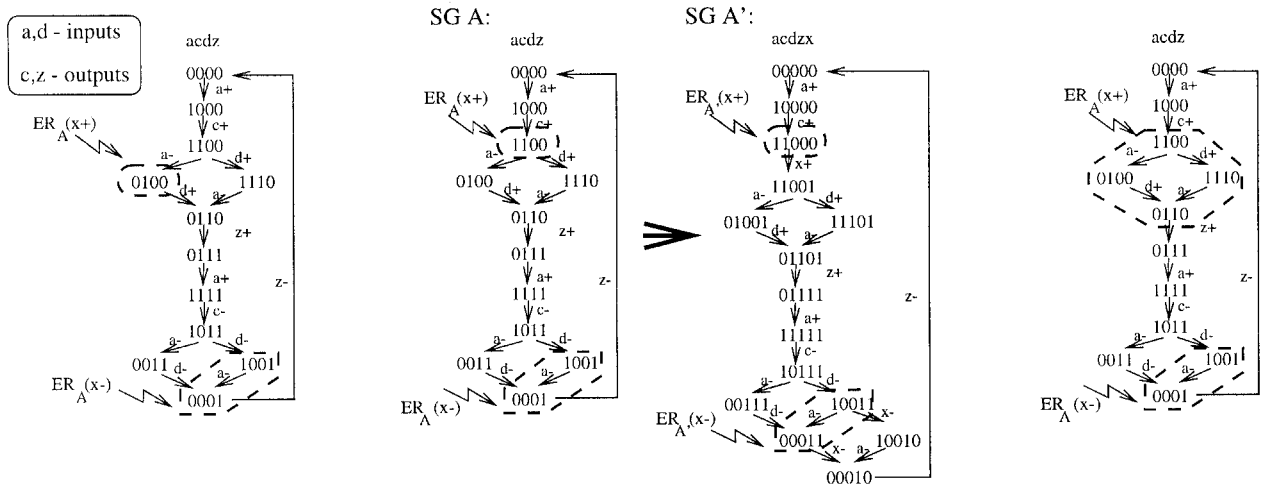


Fig. 5. Different cases of signal insertion for benchmark `hazard.g`: (a) violating the SIP-condition, (b) changing the I/O interface, and (c) correct insertion.

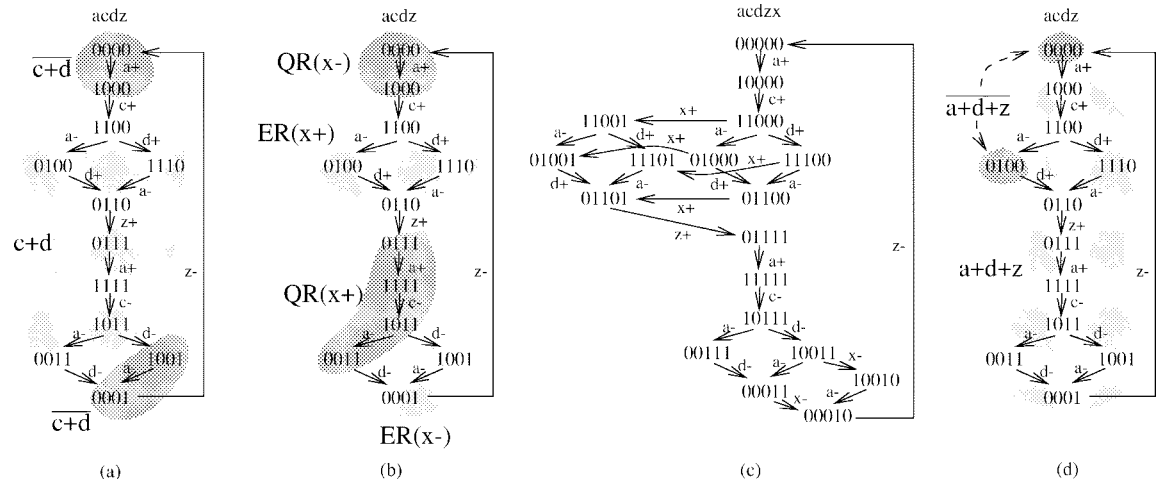


Fig. 6. (a) Bipartition by function $x = c + d$, (b) I-partition with SIP ER's, (c) insertion of signal x, and (d) bipartition for function $x = a + d + z$.

Henceforth, we will say that a function F *is speed independent* in an SG if an I-partition with SIP ER's can be found for F. In our example, $F = c+d$ is a speed-independent function. Fig. 6(d) shows a bipartition of the set of states for function $F = a + d + z$. It can be easily seen that this function is not speed independent in this SG as no I-partition with SIP ER's can be found. This is due to the isolation of state 0100 in a diamond, that generates a forbidden configuration similar to the one of Fig. 4(b).

D. Basic Definitions About Boolean Functions and Relations

An important part of our decomposition method is finding appropriate candidates for characterization (by means of Boolean covers) of the sets of states $ER_A(x+)$ and $ER_A(x-)$ for the inserted signal x. For this, we need to reference here several important concepts about Boolean functions and relations [12].

An incompletely specified (scalar) Boolean function is a functional mapping F: $B^n \rightarrow \{0, 1, -\}$, where $B = \{0, 1\}$ and "$-$" is a *don't care* value. The subsets of domain B^n in which F holds the zero, one, and don't care value are, respectively, called the *OFF-set*, *ON-set*, and *DC-set*. F is *completely specified* if its DC-set is empty. We shall further always assume that F is a completely specified Boolean function unless we explicitly say otherwise.

Let $F(x_1, x_2, \ldots, x_n)$ be a Boolean function of n Boolean variables. The set $X = \{x_1, x_2, \ldots, x_n\}$ is called the *support* of the function F. In this paper, we shall mostly be using the notion of *true support*, which is defined as follows. A variable $x \in X$ is *essential* for function F (or F is dependent on x) if there exist at least two minterms $v1$, $v2$ different only in the value of x, such that $F(v1) \neq F(v2)$. The set of essential variables for a Boolean function F is called the *true support* of F and is denoted by $sup(F)$. It is clear that for an arbitrary Boolean function its support may not be the same as the true support. E.g., for a support $X = \{a, b, c\}$ and a function $F(X) = \overline{b} + c$ the true support of $F(X)$ is $sup(F) = \{b, c\}$, i.e., only a subset of X.

Let $F(X)$ be a Boolean function $F(X)$ with support $X = \{x_1, x_2, \ldots, x_n\}$. The *cofactor* of $F(X)$ with respect to x_i $(\overline{x_i})$ is defined as $F_{x_i} = F(x_1, x_2, \ldots, x_i = 1, \ldots, x_n)$ $(F_{\overline{x_i}} = F(x_1, x_2, \ldots, x_i = 0, \ldots, x_n)$, respectively). The well-known Shannon expansion of a Boolean function $F(X)$ is

based on its cofactors: $F(X) = x_i F_{x_i} + \overline{x_i} F_{\overline{x_i}}$. The existential abstraction of a function $F(X)$ with respect to x_i is defined as $\exists_{x_i} F = F_{x_i} + F_{\overline{x}_i}$. The existential abstraction can be naturally extended to a set of variables. The *Boolean difference*, or *Boolean derivative*, of $F(X)$ with respect to x_i is defined as $\delta F/\delta x_i = F_{x_i} \oplus F_{\overline{x_i}}$.

A function $F(x_1, x_2, \ldots, x_i, \ldots, x_n)$ is *unate in variable* x_i if either $F_{\overline{x_i}} \leq F_{x_i}$ or $F_{x_i} \leq F_{\overline{x_i}}$ under ordering $0 \leq - \leq 1$. In the former case ($F_{\overline{x_i}} \leq F_{x_i}$) it is called *positive unate in* x_i, in the latter case *negative unate in* x_i. A function that is not unate in x_i is called *binate* in x_i. A function is (positive/negative) *unate* if it is (positive/negative) *unate* in all its support variables. Otherwise it is binate. For example, the function $F = a + b + \overline{c}$ is positive unate in variable a because $F_a = 1 \geq F_{\overline{a}} = b + \overline{c}$.

For an incompletely specified function $F(X)$ with a DC-set, let us define the DC function F_{DC}: $B^n \rightarrow B$ such that $ON(F_{DC}) = DC(F)$. We will say that a function $\tilde{F}$ is an *implementation of* F if $F \cdot \overline{F_{DC}} \leq \tilde{F} \leq F + F_{DC}$.

A *Boolean relation* is a relation between Boolean spaces [2], [12]; it can be seen as a generalization of a Boolean function, where a point in the domain B^n can be associated with several points in the codomain. More formally, a BR R is $R \subseteq B^n \times \{0,1\}^m$. Sometimes, we shall also use the "$-$" symbol as a shorthand in denoting elements in the codomain vector, e.g., 10 and 00 will be represented as one vector -0. BR's play an important role in multilevel logic synthesis [12], and we shall use them in our decomposition method.

Consider a set of Boolean functions $\mathcal{H} = \{H_1, H_2, \ldots, H_m\}$. Let $R \subseteq B^n \times \{0,1\}^m$ be a BR with the same domain as functions from $\mathcal{H}$. We will say that $\mathcal{H}$ is *compatible* with R if for every point v in the domain of R the vector of values $(v, H_1(v), H_2(v), \ldots, H_m(v))$ is an element of R. An example of compatible functions will be given in Section IV.

III. Overview of the Method

In this section we describe our proposed method for sequential decomposition of speed-independent circuits aimed at technology mapping. It consists of three main steps:

1) synthesis via decomposition based on BR's;
2) signal insertion and generation of a new SG;
3) library matching.

The first two steps are iterated until all functions are decomposed into implementable gates or no further progress can be made. At each iteration only one noninput signal is decomposed by inserting a new internal signal (Step 2). Resynthesis of all noninput signals is performed after the insertion of a new signal (Step 1). Finally, Step 3 collapses decomposed gates and matches them with library gates.

The pseudocode for the technology mapping algorithm is given in Fig. 7. At each iteration of the main loop, a new signal is inserted. In order to define a function for this new signal, a set of valid decompositions is calculated for each noninput signal (line 2 in Fig. 7) and the best one is kept (line 3). Finally the most complex function from all the decompositions is implemented as a new signal (lines 5 and 6). Choosing the most complex function at each step allows this function to become a candidate for decomposition in the next iteration. Thus, we decompose the largest gates first.

```
  do
1:  foreach non-input signal x do
      solutions(x):=∅;
2:    foreach gate G ∈ {latches, and2, or2} do
        solutions(x):=solutions(x) ∪ decompositions(x,G);
      endfor
3:    best_H(x) := Best SIP candidate from solutions(x);
    endfor
4:  if foreach x, best_H(x) is implementable
    or foreach x, best_H(x) is empty then exit loop;
5:  Let H be the most complex best_H(x);
6:  Insert new signal z implementing H and derive new SG;
  forever
7:Library matching;
```

Fig. 7. Algorithm for logic decomposition and technology mapping.

The algorithm terminates when all noninput signals are implementable with gates from the library or when no more signals can be further decomposed (line 4).

The proposed method breaks an initial complex gate implementation of an SG, *starting from the gate output*,[2] by using *sequential* (if its function is self-dependent, i.e., it has internal feedback) or *combinational* gates.

This method is complementary to the one proposed in [7] in which the decomposition was performed by using algebraic divisors of the current implementations of the output signals, and thus decomposition was performed *from the inputs* of complex gates.

Given a vector X of SG signals and given one noninput signal $y \in X$ (in general the function $F(X)$ for y may be self-dependent), we try to decompose function $F(X)$ into (line 2 of algorithm in Fig. 7):

- a combinational or sequential gate with function $G(Z, y)$, where Z is a vector of newly introduced signals,
- a vector of combinational[3] functions $\mathcal{H}(X)$ for signals Z, so that $G(\mathcal{H}(X), y)$ implements $F(X)$.

Moreover, we require the newly introduced signals to be speed-independent (line 3).

The problem of representing the flexibility in the choice of the $\mathcal{H}$ functions as BR's has been explored, in the context of combinational logic minimization, by [22] among others. Here we extend its formulation to cover also *sequential* gates (in Sections IV-A and IV-C). This is essential in order to overcome the limitations of previous methods for speed-independent circuit synthesis that were based on a specific architecture. Now we are able to use a broad range of sequential elements, like set and reset dominant SR latches, transparent D latches, and so on. We believe that overcoming this limitation of previous methods, that could only use C elements and dual-rail SR-latches, is one of the major strengths of this work. Apart from dramatically improving some experimental results, it allows one to use a "generic" standard-cell library, that generally includes SR and D latches (but not C

[2]Note that after decomposition terminates, technology mapping can be performed indifferently starting from the inputs or from the outputs.

[3]The restriction that $\mathcal{H}(X)$ be combinational will be partially lifted in Section IV-C.

elements), without the need to design and characterize any new asynchronous-specific gates.

The algorithm proceeds as follows. We start from an SG and derive a logic function for all its noninput signals (line 1). We then perform an implementability check for each such function as a library gate. The largest nonimplementable function is selected for decomposition. In order to limit the search space, we currently try as candidates for G (line 2):

- all the sequential elements in the library (assumed to have two inputs at most, again in order to limit the search space);
- two-input *AND*, *OR* gates with all possible input inversions.

The flexibility in the choice of functions $\mathcal{H} = (H_1, H_2)$ is defined by a BR, that represents the solution space of $F(X) = G(\mathcal{H}(X))$, as described in Section IV-A.

The set of function pairs (H_1, H_2) compatible with the BR is then checked for speed-independence (line 3), as described in Section II-C. This additional requirement has forced us to implement a new BR minimizer, that returns a *set of* compatible functions, as outlined in Section V-A. If both are not speed-independent, the pair is immediately rejected.

Then, both H_1 and H_2 are checked for approximate (as discussed above) implementability in the library, in increasing order of estimated cost. We have two cases:

1) both are speed-independent and implementable: in this case the decomposition is accepted,
2) otherwise, the most complex implementable H_i is selected, and the other one is merged with G.

Choosing the most complex function, as mentioned above, experimentally helps with keeping the decomposition balanced. Note that at this stage we can also implement H_1 or H_2 as a *sequential gate* if the *sufficient* conditions described in Section IV-C are met.

The procedure is iterated as long as there is progress or until everything has been decomposed (line 4). Each time a new function H_i is selected to be implemented as a new signal, it is inserted into the SG (line 6) and resynthesis is performed in the next iteration.

The incompleteness of the method is essentially due to the greedy heuristic search that accepts the smallest implementable or nonimplementable but speed-independent solution. Therefore, a speed-independent solution could be missed, e.g., if it corresponds to a redundant cover as shown in Example 2. In order to enable the generation of redundant decompositions in our implementation, the method based on BR's is combined with a method based on monotonic covers that performs a direct search for speed-independent covers [9], [1], but is also incomplete due to a very restricted decomposition structure based on C-elements. We believe that an exhaustive enumeration of all speed-independent solutions with backtracking would be complete (but impractical), by a relatively straightforward extension of the results in [20] that applied only to circuits without external inputs.

Note that in this paper we assume that input inverters (bubbles) attached to the inputs of other gates are "fast," and therefore bubbles can be added to the fanin of any hazard-free gate available in the library. The implementation is guaranteed to be hazard-free under the following conservative assumption: the delay of a bubble with fanin signal x is less than the delay of any other gate, with possibly input bubbles, with x in its fanin [9].[4]

At the end, we perform a Boolean matching step ([11]) to recover area and delay (line 7). This step can merge together the simple two-input combinational gates that we have conservatively used in the decomposition into a larger library gate. It is guaranteed not to introduce any hazards if the matched gates are atomic.

IV. Logic Decomposition Using Boolean Relations

A. Specifying Permissible Decompositions with BR's

As discussed above, in this paper we apply BR's to the following problem.

Given an incompletely specified Boolean function $F(X)$ for signal y, $y \in X$, decompose it into two levels $y = G(Z, y)$; $Z = \mathcal{H}(X)$ such that $G(\mathcal{H}(X), y)$ implements $F(X)$ and functions G and $\mathcal{H}$ have a simpler implementation than F (any such $\mathcal{H}$ will be called permissible).

Note that the first-level function $\mathcal{H}(X) = \{H_1(X), \ldots, H_n(X)\}$ is a multioutput logic function, specifying the behavior of internal nodes of the decomposition, $Z = \{z_1, \ldots, z_n\}$.[5]

The final goal is a function decomposition to a form that is easily mappable to a given library. Hence only functions available in the library are selected as candidates for G. Then at each step of decomposition a small mappable piece (function G) is cut from the potentially complex and unmappable function F. For a selected G all permissible implementations of function $\mathcal{H}$ are specified with a BR and then via minimization of BR a few best compatible functions are obtained. All of them are verified for speed-independence by checking SIP-sets. The one which is speed-independent and has the best estimated cost is selected.

Since the support of function F can include the output variable y, it can specify sequential behavior. In the most general case we perform two-level *sequential* decomposition such that both function G and function $\mathcal{H}$ can be sequential, i.e., contain their own output variables in their supports. The second level of the decomposition is made sequential by selecting a latch from the library as a candidate gate, G. The technique for deriving a sequential solution for the first level $\mathcal{H}$ is described in Section IV-C.

We next show by example how all permissible implementations of a decomposition can be expressed with BR's.

Example 2: Consider the STG in Fig. 8(a), whose SG appears in Fig. 9. Signals a, c, and d are inputs and y is an output. A possible implementation of the logic function for y is $F(a, c, d, y) = ac\overline{d} + y(\overline{c} + \overline{d})$. Let us decompose this function using as G a reset-dominant Rs-latch represented by the equation $y = G(R, S, y) = \overline{R}(S + y)$ [see Fig. 8(b)].

[4]This is likely to be satisfied if the layout program keeps such inverters very close to their fanout gates.

[5]For simplicity we consider the decomposition problem for a single-output binary function F, although a generalization for the multioutput and multivalued functions would be straightforward.

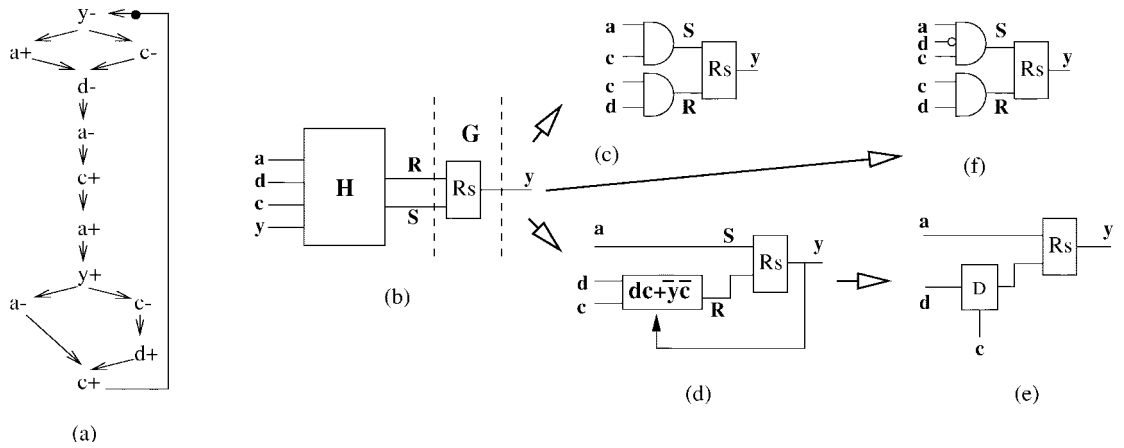
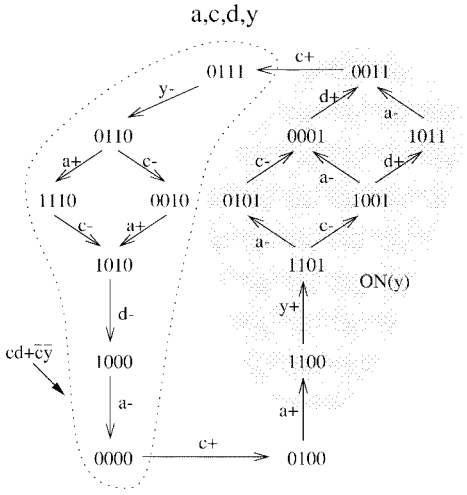


Fig. 8. Sequential decomposition for function $y = ac\overline{d} + y(\overline{c} + \overline{d})$.



$acdy$	F	$R\ S$
0000	0	1–,–0
0001	1	0–
0010	0	1–,–0
0011	1	0–
0100	0	1–,–0
0101	1	0–
0110	0	1–,–0
0111	0	1–
1000	0	1–,–0
1001	1	0-
1010	0	1–,–0
1011	1	0–
1100	1	01
1101	1	0–
1110	0	1–,–0
1111	–	–

Fig. 9. State graph and decomposition of signal y by an RS latch.

TABLE I
BOOLEAN RELATIONS FOR DIFFERENT GATES

Region	C-element H_1, H_2	D-latch C, D	Rs R, S
$ER(y+)$	11	11	01
$QR(y+)$	$\{1-,-1\}$	$\{0-,-0\}$	$0-$
$ER(y-)$	00	10	1
$QR(y-)$	$\{0-,-0\}$	$\{0-,-0\}$	$\{1-,-0\}$
unreachable	— —	— —	— —
Region	**Sr S, R**	**AND H_1, H_2**	**OR H_1, H_2**
$ER(y+)$	$1-$	11	$\{1-,-1\}$
$QR(y+)$	$\{1-,-0\}$	11	$\{1-,-1\}$
$ER(y-)$	01	$\{0-,-0\}$	00
$QR(y-)$	$0-$	$\{0-,-0\}$	00
unreachable	— —	— —	— —

At the first step we specify the permissible implementations for the first level functions $R = H_1$ and $S = H_2$ by using the BR specified in the table of Fig. 9. Consider, for example, vector a, c, d, $y = 0000$. It is easy to check that $F(0,0,0,0) = 0$. Hence, for vector 0000 the table specifies that $(R,S) = \{11,10,00\} = \{1-,-0\}$, i.e., any implementation of R and S must keep for this input vector *either* R at one *or* S at zero, since these are the necessary and sufficient conditions for the Rs-latch to keep the value zero at the output y, as required by the specification. On the other hand, only the solution $R = 0$, $S = 1$ is possible for the input vector 1100, which corresponds to setting the output of the Rs-latch to one. The BR solver will find, among others, the two solutions illustrated in Fig. 8(c)–(d): 1) $R = cd + \overline{y}\,\overline{c}$; $S = a$ and 2) $R = cd$; $S = ac\overline{d}$. Solution (2) is not speed-independent, and therefore only solution (1) will be included as a SIP candidate. Another speed-independent solution, (3) $R = cd$; $S = ac\overline{d}$ [Fig. 8(f)] corresponding to a redundant cover will be included into the list of SIP candidates by the monotonic cover method. It differs from solution 2) by adding a redundant literal to function S. The monotonic cover method [9], [1] performs direct search for speed-independent cubes and covers for each excitation region of a signal assuming a restricted decomposition structure based on a C-element or a dual-rail RS-latch. The best solution is selected among 1) and 3) depending on the cost function.

Table I specifies compatible values of the BR for different types of gates: a C-element, a D-latch, a reset-dominant Rs-latch, a set-dominant Sr-latch, a two-input AND gate and a two input OR gate. All states that are not reachable in the SG form the DC-set for the BR. E.g., for each state s from $ER(y+)$ only one compatible solution, 11, is allowed for input functions H_1, H_2 of a C-element. This is because the output of a C-element in all states, $s \in ER(y+)$ is at zero and $F(s) = 1$. Under these conditions the combination 11 is the only possible input combination that implies the value one at the output of the C-element. On the other hand, for each state $s \in QR(y+)$, the output $y = 1$ and $F(s) = 1$. Hence, it is enough to keep at least one input of the C-element in 1. This is expressed by values $\{1-,-1\}$ in the second line of the table. Similarly, all other compatible values are derived.

B. Functional Representation of Boolean Relations

Given an SG satisfying the CSC requirement, each output signal $y \in X$ is associated with a unique *incompletely* specified function $F(X)$, whose DC-set represents the set of unreachable states. $F(X)$ can be represented by three *completely* specified functions, denoted $ON(y)(X)$, $OFF(y)(X)$, and $DC(y)(X)$ representing the ON-, OFF-, and DC-set of

$F(X)$, such that they are pairwise disjoint and their union is a tautology.

Let a generic n-input gate be represented by a Boolean equation $q = G(Z, q)$, where $Z = \{z_1, \ldots, z_n\}$ are the inputs of the gate, and q is its output.[6] The gate is sequential if q belongs to the true support of $G(Z, q)$.

We now give the characteristic function of the BR for the implementation of $F(X)$ with gate G. This characteristic function represents *all* permissible implementations of $z_1 = H_1(X), \ldots, z_n = H_n(X)$ that allow F to be decomposed by G

$$BR(y)(X, Z) = ON(y)(X) \cdot G(Z, y) + OFF(y)(X) \cdot \overline{G(Z, y)} + DC(y)(X). \quad (1)$$

Intuitively, this equation specifies the relations:

- between every minterm (with support in X) in the ON-set and the functions of Z that make $G(Z, y) = 1$;
- between every minterm in the OFF-set and the functions of Z that make $G(Z, y) = 0$.

In the example of Fig. 9, the minterms $\overline{a}\,\overline{c}\,\overline{d}\,\overline{y}$, $\overline{a}\,\overline{c}\,\overline{d}y$, and $acdy$ belong to the OFF-, ON- and DC-set, respectively. Thus, the BR for the function $G(R, S, y) = \overline{R}(S + y)$ will be

$$\begin{aligned} &BR(y)(a, c, d, y, R, S) \\ &\quad = \overline{a}\,\overline{c}\,\overline{d}\,\overline{y} \cdot \overline{\overline{R}(S + y)} + \overline{a}\,\overline{c}\,\overline{d}y \cdot \overline{R}(S + y) + \cdots + acdy \\ &\quad = \overline{a}\,\overline{c}\,\overline{d}\,\overline{y} \cdot (R + \overline{S}) + \overline{a}\,\overline{c}\,\overline{d}y \cdot \overline{R} + \cdots + acdy. \end{aligned}$$

Given the characteristic function (1), the corresponding table describing the BR can be derived using cofactors. For each minterm m with support in X, the cofactor $BR(y)_m$ gives the characteristic function of all compatible values for $z_1, \ldots, z_n$.

Finding a decomposition of F with gate G is reduced to finding a set of n functions $\mathcal{H}(X) = (H_1(X), \ldots, H_n(X))$ such that

$$BR(y)(X, \mathcal{H}(X)) = 1. \quad (2)$$

Example 3: (*Example 2 continued.*) The SG shown in Fig. 9 corresponds to the STG in Fig. 8. Let us consider how the implementation of signal y with a reset-dominant Rs latch can be expressed using the characteristic function of the BR. Recall that the table shown in Fig. 9 represents the function $F(a, c, d, y) = ac\overline{d} + y(\overline{c} + \overline{d})$ and the permissible values for the inputs R and S of the Rs latch. The ON-, OFF-, and DC-sets of function $F(a, c, d, y)$ are defined by the following completely specified functions:

$$\begin{aligned} ON(y) &= y(\overline{c} + \overline{d}) + ac\overline{d} \\ OFF(y) &= \overline{y}(\overline{a} + \overline{c} + d) + \overline{a}cd \\ DC(y) &= acdy. \end{aligned}$$

The set of permissible implementations for R and S is characterized by the following characteristic function of the BR specified in the table. It can be obtained from (1) by substituting expressions for $ON(y)$, $OFF(y)$, $DC(y)$, and the function of an Rs-latch, $\overline{R}(S + y)$:

$$\begin{aligned} BR(y)(a, c, d, y, R, S) = &\overline{R}Sac\overline{d} + \overline{R}y(\overline{c} + \overline{d}) \\ &+ (R + \overline{S})\overline{y}(\overline{a} + \overline{c} + d) \\ &+ \overline{a}cd(R + \overline{S}\,\overline{y}) \\ &+ acdy. \end{aligned} \quad (3)$$

This function has value one for *all* combinations represented in the table of Fig. 9 and value zero for all combinations that are not in the table [e.g., for $(a, c, d, y, R, S) = 000\,001$]. For example, the set of compatible values for $acdy = 0110$ is given by the cofactor

$$BR(y)_{\overline{a}cd\overline{y}} = R + \overline{S}$$

which correspond to the terms $1-$ and -0 given for the BR for that minterm.

Two possible solutions for the equation $BR(y(a, c, d, y, R, S) = 1$ corresponding to Fig. 8(c)–(d) are

$$\begin{aligned} &R = cd; \quad S = ac \\ &R = cd + \overline{y}\,\overline{c}; \quad S = a. \end{aligned}$$

C. Two-Level Sequential Decomposition

Accurate estimation of the cost of each solution produced by the BR minimizer is essential in order to ensure the quality of the final result. The minimizer itself can only handle combinational logic, but often (as shown below) the best solution can be obtained by replacing a combinational gate with a sequential one. This section discusses some heuristic techniques that can be used to identify when such a replacement is possible without altering the asynchronous circuit behavior, and without undergoing the cost of a full-blown sequential optimization step. In particular, it discusses how a decomposed combinational function $z = H(X, y)$ can be replaced by a sequential function $z = H'(X, z)$ without changing the behavior of the circuit. Let us consider our example again.

Example 4: (*Example 2 continued.*) Let us assume that the considered library contains three-input AND, OR gates and Rs-, Sr-, and D-latches. Implementation (1) of signal y by an Rs-latch with inputs $R = cd$ and $S = ac\overline{d}$ matches the library and requires two AND gates, one with two and one with three inputs, and one Rs-latch. The implementation (2) of y by an Rs-latch with inputs $R = cd + \overline{y}\,\overline{c}$ and $S = a$ would be rejected, as it requires a complex AND-OR gate which is not in the library. However, when input $\overline{y}$ in the function $cd + \overline{y}\,\overline{c}$ is replaced by signal R, the output behavior of R will not change, i.e., function $R = cd + \overline{y}\,\overline{c}$ can be safely replaced by $R = cd + R\overline{c}$. The latter equation corresponds to the function of a D-latch and gives the valid implementation shown in Fig. 8(e).

[6] In the context of Boolean equations representing gates we shall liberally use the "=" sign to denote "assignment," rather than mathematical equality. Hence, literal q in the left-hand side of this equation stands for the *next* value of signal q while the same literal in the right-hand side corresponds to its previous value.

Our technique to improve the precision of the cost estimation step, by partially considering sequential gates, is as follows.

1) Produce permissible functions $z_1 = H_1(X)$ and $z_2 = H_2(X)$ via the minimization of BR's (z_1 and z_2 are always combinational as $z_1, z_2 \notin X$).
2) Estimate the complexity of H_1 and H_2:
   if H_i matches the library *then Complexity = cost of the gate else Complexity =* literal count
3) Estimate the possible simplification of H_1 and H_2 due to adding signals z_1 and z_2 to their supports, i.e., estimate the complexity of the new pair $\{H_1', H_2'\}$ of permissible functions $z_1 = H_1'(X, z_1, z_2)$, $z_2 = H_2'(X, z_1, z_2)$.
4) Choose the best complexity between H_i and H_i'.

Let us consider the task of determining H_1' and H_2' as in Step 3. Let A be an SG encoded by variables from set V and let $z = H(X, y)$, such that $X \subseteq V$, $y \in V$, be an equation for the new variable $z \notin V$ which is to be inserted in A. The resulting SG is denoted $A' = Ins(A, z = H(X, y))$. Sometimes we will simply write $A' = Ins(A, z)$ or $A' = Ins(A, z_1, \ldots, z_k)$ when more than one signal is inserted.

A solution for Step 3 of the above procedure can be obtained by minimizing functions for signals z_1 and z_2 in an SG $A' = Ins(A, z_1, z_2)$. However, this is rather inefficient because the creation of SG A' is computationally expensive. Hence instead of looking for an exact estimation of complexity for signals z_1 and z_2 we will rely on a heuristic solution, following the ideas on input resubstitution presented in Example 4. For computational efficiency, the formal conditions on input resubstitution should be formulated in terms of the original SG A rather than in terms of the SG A' obtained after the insertion of new signals.[7]

Lemma 1: Let Boolean function $H(X, y)$ implement the inserted signal z and be positive (negative) unate in y. Let $H'(X, z)$ be the function obtained from $H(X, y)$ by replacing each literal y (or $\overline{y}$) by literal z. The SG's $A' = Ins(A, z = H(X, y))$ and $A'' = Ins(A, z = H'(X, z))$ are isomorphic, i.e., have the same states and arcs, iff the following condition is satisfied:

$$\delta H(X, y)/\delta y \cdot S^* \equiv 0$$

where S^* is the characteristic function describing the set of states $ER_A(z+) \cup ER_A(z-)$ in A.

Informally Lemma 1 states that resubstitution of input y by z is permissible if in all states where the value of function $H(X, y)$ depends on y, the inserted signal z has a stable value.

The intuition behind this lies in the way of constructing SG's A' or A'' when signal z is inserted in SG A. Any state $s \in A$ which belongs to the excitation region $ER(z*)$ of z gives rise to two states s' and s'' in A' or A'' ($s' \stackrel{z*}{\rightarrow} s''$). For A' and A'' to be isomorphic, the value of the functions $H(X, y)$ and $H'(X, z)$ in these states must coincide, otherwise the enabling of signal z would be different, meaning that either s' or s'' would have different output arcs in A' and A''. However, if the condition of Lemma 1 is violated, then $H(X, y)$ keeps the same value in both s' and s'', while $H'(X, z)$ changes its value due to the change of z. Hence, A' and A'' cannot be isomorphic if the conditions of the Lemma are violated.

Example 5: (*Example 2 continued.*) Let input R of the RS-latch be implemented as $cd + \overline{y}\,\overline{c}$ [see Fig. 8(d)]. The ON-set of function $H = cd + \overline{y}\,\overline{c}$ is shown by the dashed line in Fig. 9. The input border of H is the set of states by which its ON-set is entered in the original SG A, i.e., $IB(H) = \{0111\}$. By similar consideration, we have that $IB(\overline{H}) = \{0100\}$. These input borders satisfy the SIP conditions and, hence, $IB(H)$ can be taken as $ER_A(R+)$, while $ER_A(R-)$ must be expanded beyond $IB(\overline{H})$ by state 1100 so that it does not delay the input transition $a+$ ($ER_A(R-) = \{0100, 1100\}$).

The set of states where the value of function H essentially depends on signal y is given by the function $\delta H(X, y)/\delta y = a\overline{c}$. H is negative unate in y and cube $a\overline{c}$ has no intersection with $ER_A(R+) \cup ER_A(R-)$. Therefore by the condition of Lemma 1 literal $\overline{y}$ can be replaced by literal R, thus producing a new permissible function $R = cd + R\overline{c}$.

This result can be generalized to binate functions, as follows.

Lemma 2: Let Boolean function $H(X, y)$ implement the inserted signal z and be binate in y. Function H can be represented as $H(X, y) = Fy + G\overline{y} + R$, where F, G, and R are Boolean functions not depending on y. Let $H'(X, z) = z(F + G) + R$. Then SG's $A' = Ins(A, z = H(X, y))$ and $A'' = Ins(A, z = H'(X, z))$ are isomorphic iff the following conditions are satisfied:

$$\delta H(X, y)/\delta y \cdot S^* \equiv 0 \tag{1}$$

$$F * G * \overline{R} \cap S^+ \equiv 0 \tag{2}$$

where S^* and S^+ are the characteristic Boolean functions describing sets of states $ER_A(z+) \cup ER_A(z-)$ and $ER_A(z+)$ in A, respectively.

The proof is given in the Appendix.

The conditions of Lemma 2 can be efficiently checked within our binary decision diagram (BDD)-based framework. They require to check two tautologies involving functions defined over the states of the original SG A. This heuristic solution is a tradeoff between computational efficiency and optimality. Even though the estimation is still not exact (the exact solution requires the creation of $A' = Ins(A, z)$), it allows us to discover and possibly use the implementation of Fig. 8(e).

V. Implementation Issues

The method for logic decomposition presented in the previous section has been implemented in a synthesis tool for speed-independent circuits. The main purpose of such implementation was to evaluate the potential improvements that could be obtained in the synthesis of speed-independent circuits by

[7] Note that this heuristic estimation covers only the cases when one of the input signals for a combinational permissible function H_i is replaced by the feedback z_i from the output of H_i itself. Other cases could also be investigated, but checking them would be too complex.

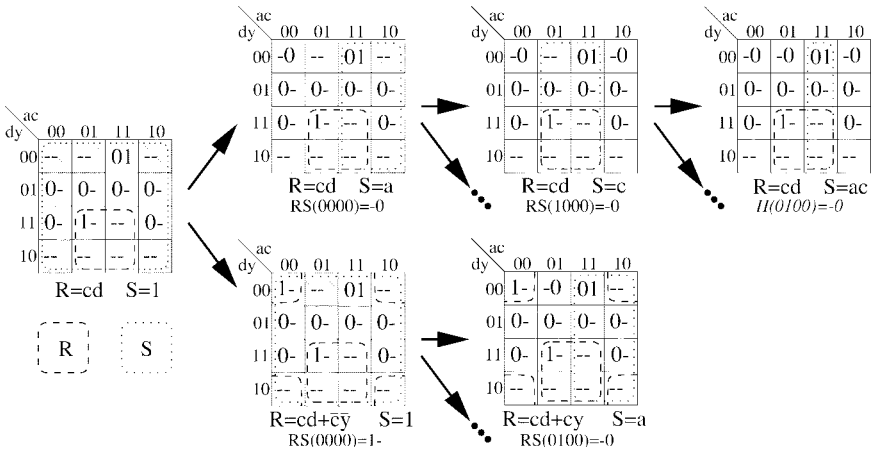


Fig. 10. Exploration tree for solving BR's.

using a BR-based decomposition approach. Efficiency of the current implementation was considered to be a secondary goal at this stage of the research.

A. Solving Boolean Relations

In the overall approach, it is required to solve BR's for each output signal and for each gate and latch used for decomposition. Furthermore, for each signal and for each gate, several solutions are desirable in order to increase the chances to find SIP functions.

Previous approaches to solve BR's [2], [21] do not satisfy the needs of our synthesis method, since (1) they minimize the number of terms of a multiple-output function and (2) they deliver (without significant modifications to the algorithms and their implementation) only one solution for each BR. In our case we need to obtain *several compatible solutions* with the primary goal of *minimizing the complexity of each function individually*. Term sharing is not significant because two-level decomposition of a function is not speed-independent in general and, hence, each minimized function must be treated as an *atomic* object. Sharing can be exploited, on the other hand, when resynthesizing the circuit after insertion of each new signal. For this reason we devised a heuristic approach to solve BR's. We next briefly sketch and illustrate it with the example of Fig. 10, that corresponds to the decomposition shown in Fig. 9.

Given a BR $BR(y)(X, z_1, \ldots, z_n)$, each function H_i for z_i is individually minimized by assuming that all other functions H_j $(i \neq j)$ will be defined in such a way that $\mathcal{H}(X)$ will be a compatible solution for BR. Formally, for each z_i we define

$$\mathcal{F}^i = \exists_{z_1 \ldots z_{i-1} z_{i+1} \ldots z_n} BR(y)(X, z_1, \ldots, z_n).$$

The existential abstraction derives a new relation only for z_i, that ignores the constraints due to other $z_j \neq z_i$. Next, the ON-, OFF-, and DC-sets for minimization are obtained as follows:

$$\begin{aligned} ON(z_i) &= \mathcal{F}^i_{z_i} \cdot \overline{\mathcal{F}^i_{\overline{z_i}}} \\ OFF(z_i) &= \mathcal{F}^i_{\overline{z_i}} \cdot \overline{\mathcal{F}^i_{z_i}} \\ DC(z_i) &= \mathcal{F}^i_{z_i} \cdot \mathcal{F}^i_{\overline{z_i}}. \end{aligned}$$

In the example of Fig. 9 that corresponds to the BR (3) we have

$$\begin{aligned} \mathcal{F}^R &= \exists_S BR(y)(a,c,d,y,R,S) \\ &= R\underbrace{\overline{a}cdy}_{ON(R)} + \overline{R}\underbrace{(acd + \overline{c}y + \overline{d}y)}_{OFF(R)} \\ &\quad + \underbrace{\overline{y}(\overline{a}+\overline{c}+d) + acdy}_{DC(R)} \\ \mathcal{F}^S &= \exists_R BR(y)(a,c,d,y,R,S) \\ &= S\underbrace{ac\overline{d}\,\overline{y}}_{ON(S)} + \underbrace{\overline{a}+\overline{c}+d+y}_{DC(S)} \end{aligned}$$

that corresponds to the leftmost Karnaugh map of Fig. 10.

A two-level cover for each z_i is obtained individually by using a standard Boolean minimizer. Let us call $C(z_i)$ the cover obtained for z_i. In general, an incompatible solution may be generated when combining all covers. In the example, an individual minimization of R and S yields the solution $C(R) = cd$ and $C(S) = 1$.

The set of minterms that are incompatible with the BR can be represented with a characteristic function, by substituting each z_i by $C(z_i)$ in $\overline{BR(y)(X,Z)}$. In the example,

$$\overline{BR(y)(a,c,d,y,cd,1)} = \overline{c}\,\overline{y} + \overline{a}\,\overline{d}\,\overline{y}.$$

These minterms correspond to the shadowed cells in the Karnaugh map of Fig. 10.

If no incompatible minterms are generated after minimization, the obtained solution is chosen as a valid solution for the BR. Otherwise, one of the incompatible minterms is heuristically selected. In our case, we select one of the minterms with the largest number of incompatible neighbors (Hamming distance 1). Intuitively and empirically we have observed that this strategy helps to "legalize" a larger number of minterms in forthcoming iterations of the solver, since the compatibility of one minterm often propagates to its neighbors.

Once a minterm has been selected, new BR's are defined by redirecting its output to a set of compatible output cubes that cover the flexibility expressed by the original BR. A new BR is defined for each such cube. In our example, we select the minterm $\overline{a}\,\overline{c}\,\overline{d}\,\overline{y}$ and derive two new BR's by assigning the

cube $(R,S)=-0$ and $(R,S)=1-$ to each one, respectively. Thus, we obtain the BR's

$$BR_1(y)(a,c,d,y,R,S)=BR(y)(a,c,d,y,R,S)\cdot\overline{\overline{a}\,\overline{c}\,\overline{d}\,\overline{y}\,S}$$

$$BR_2(y)(a,c,d,y,R,S)=BR(y)(a,c,d,y,R,S)\cdot\overline{\overline{a}\,\overline{c}\,\overline{d}\,\overline{y}\,\overline{R}}.$$

Any valid solution for BR_1 or BR_2 is also a valid solution for BR.

This approach generates a tree of BR's to be solved. This provides a way of obtaining several compatible solutions for the same BR. However, the exploration may become prohibitively expensive if the search tree is not pruned. In our implementation we use a heuristic pruning strategy that at each step keeps only those nodes that have the most promising chances of generating a valid solution. In the current implementation, the cost of each node is evaluated as a weighted sum of the number of incompatible minterms and the number of literals of the (possibly invalid) solution. The solver stops when the number of generated solutions is considered to be satisfactory.

In the example, several solutions are generated. Among them, Fig. 10 depicts the part of the exploration tree leading to two valid solutions

$$R=cd \quad S=ac$$

$$R=cd+\overline{c}\,\overline{y} \quad S=a.$$

The time required by the BR solver dominates the computational cost of the overall method in our current implementation. Ongoing research on solving BR's for our framework is being carried out. We believe that the fact that we pursue to minimize functions individually, i.e., not targeting the term sharing among different output functions, and that we only deal with two-output decompositions, may be crucial to derive algorithms which might be much more efficient than the existing approaches.

B. Selection of the Best Decomposition

Each generated decomposition for an output signal consists of a (possibly sequential) output gate with behavior $y=G(Z,y)$ and a set of decomposed functions $Z=\mathcal{H}(X)$. From the selected decomposition, only one of the $H_i(X)$ functions will be chosen for its implementation as a new signal.

Thus, once a set of compatible solutions has been generated for each output signal, the best candidate is selected according to the following criteria (in priority order).

1) At least one of the $H_i(X)$ decomposed functions must be speed-independent. This means that at least one new signal that contributes to decompose an output signal can be inserted.
2) The acknowledgment of the decomposed functions must not increase the complexity of the implementation of other signals (see Section V-C).
3) Solutions in which all $H_i(X)$ decomposed functions are implementable in the library are preferred.
4) Solutions in which the complexity of the largest non-implementable function $H_i(X)$ is minimized are preferred. This criterion helps to balance the complexity of the decomposed functions and derive balanced tree-like structures rather than linear ones.[8]
5) The estimated savings obtained by sharing a function for the implementation of several output signals is also considered as a second order priority criterion.

Among the best candidate solutions for all output signals, the function $H_i(X)$ with the largest complexity, i.e., the farthest from implementability, is selected to be implemented as a new output signal of the SG.

The complexity of a function is calculated as the number of literals in factored form. In case it is a sequential function and it matches some of the latches of the gate library, the implementation cost is directly obtained from the information provided by the library.

C. Signal Acknowledgment and Insertion

For each function delivered by the BR solver, an efficient SIP insertion must be found. This reduces to finding a partition $\{ER_A(x+),QR_A(x+),ER_A(x-),QR_A(x-)\}$ of the SG A such that $ER_A(x+)$ and $ER_A(x-)$ (that are restricted to be SIP-sets, Section II-C) become the positive and negative ER's of the new signal x. $QR_A(x+)$ and $QR_A(x-)$ stand for the corresponding state sets where x will be stable and equal to one and zero, respectively.

In general, each function may have several $ER_A(x+)$ and $ER_A(x-)$ sets acceptable as ER's. Each one corresponds to a signal insertion with different acknowledging outputs signals for its transitions. In our approach, we perform a heuristic exploration seeking for different $ER_A(x+)$ and $ER_A(x-)$ sets for each function. We finally select one according to the following criteria.

- Sets that are only acknowledged by the signal that is being decomposed (i.e., local acknowledgment) are preferred.
- If no set with local acknowledgment is found, the one with the least acknowledgment cost (i.e., with the least number of signals delayed by the new inserted signal) is selected.

The selection of the $ER_A(x+)$ and $ER_A(x-)$ sets is done independently. The cost of acknowledgment is estimated by considering the influence of the inserted signal x on the implementability of the other signals. The cost can be either increased or decreased depending on how $ER_A(x+)$ and $ER_A(x-)$ are selected, and is calculated by incrementally deriving the new SG after signal insertion.

As an example consider the SG of Fig. 5(c) and the insertion of a new signal x for the function $x=c+d$. A valid SIP set for $ER_A(x+)$ would be the set of states $\{1100,0100,1110,0110\}$, where the state $\{1100\}$ is the input border for the inserted function. A valid SIP set for $ER_A(x-)$ would be the set of states $\{1001,0001\}$. With such insertion, $ER_A(x+)$ would be acknowledged by the transition $z+$ and $ER_A(x-)$ by $z-$. However, this insertion is not unique. For the sake of simplicity, let us assume that a and d are also output signals. Then an insertion with $ER_A(x+)=\{1100\}$ would also be valid. In that case, transition $x+$ would be acknowledged by transitions $a-$ and $d+$.

[8] Different criteria, of course, may be used when we also consider the delay of the resulting implementation, since then keeping late arriving signals close to the output is generally useful and can require unbalanced trees.

TABLE II
EXPERIMENTAL RESULTS

Circuit	signals	Siegel	literals/latches		CPU (s)	Area non-SI			Area SI		
	I/O	[16]	old	new	old/new	lib 1	lib 2	best	2 inp	map	best
chu 133	3/4	yes	12/1	10/2	2/4	224	216	216	216	208	208
chu 150	3/3	no	14/2	10/1	2/18	192	160	160	160	168	208
converta	2/3	no	12/3	8/3	2/14	352	312	312	216	224	216
dff	2/2	yes	12/2	0/2	4/1	144	96	96	88	88	88
drs	2/2		n.a.	0/2	n.a./7	112	112	112	80	80	80
ebergen	2/3	no	20/3	6/2	2/4	184	160	160	160	144	144
hazard	2/2	yes	12/2	0/2	1/1	144	120	104	104	104	104
half	2/2	no	2/2	6/2	1/4	184	184	184	154	154	154
mp-forward-pkt	3/5	yes	14/3	14/2	3/31	232	232	232	256	256	256
mr1	4/7		36/9	28/2	126/456	656	624	624	480	982	480
nak-pa	4/6	no	20/4	18/2	4/441	256	248	248	250	344	250
nowick	3/3	yes	16/1	16/2	3/170	248	248	248	232	256	232
rcv-setup	3/2	yes	10/1	8/1	2/10	120	120	120	136	128	128
sbuf-ram-write	5/7	no	22/6	20/2	23/696	296	296	296	360	338	338
trimos-send	3/6		36/8	14/10	129/2071	576	480	480	786	684	684
vbe5b	3/3	no	10/2	10/2	1/10	208	216	208	202	224	202
vbe5c	3/3	no	4/3	4/3	1/12	160	160	160	178	208	178
Total			252/52	172/36	306/3960	4288	3984	**3976**	4058	4590	**3902**

D. Library Mapping

The logic decomposition of the noninput signals is completed by a technology mapping step aimed at recovering area and delay based on a technology-dependent library of gates. These reductions are achieved by collapsing small fanin gates into complex gates, provided that the gates are available in the library. The collapsing process is based on the Boolean matching techniques proposed by Mailhot *et al.* [11], adapted to the existence of asynchronous memory elements and combinational feedback in speed-independent circuits. The overall technology mapping process has been efficiently implemented based on the utilization of BDD's.

VI. EXPERIMENTAL RESULTS

A. Results in Decomposition and Technology Mapping

The method for logic decomposition presented in the previous sections has been implemented and applied to a set of benchmarks. The results are shown in Table II.

The column "Siegel" reports the results from [16] on the decomposability of the benchmarks into two-input gates. The columns "literals/latches" report the complexity of the circuits derived after logic decomposition into two-input gates. The results obtained by the method presented in this paper ("new") are significantly better than those obtained by the method presented in [7] ("old"). Note that the library used for the "new" experiments was deliberately restricted to D, Sr, and Rs latches, i.e., without C-elements that are not generally part of standard cell libraries. This improvement is mainly achieved because of two reasons.

- The superiority of Boolean methods versus algebraic methods for logic decomposition.
- The intensive use of different types of latches to implement sequential functions compared to the C-element-based implementation in [7].

Comparing with "Siegel," the new method improves the results significantly. However, Boolean decomposition does not make a tangible contribution to the decomposability of the circuit, since all the reported examples were already decomposable by using algebraic methods. Thus, we can conclude that Boolean methods mainly affect the quality of the circuits, in area and delay, whereas the method for signal insertion with multiple acknowledgment mainly contributes to their decomposability.

However, the improved results obtained by using Boolean methods are paid in terms of a significant increase in CPU time (about one order of magnitude). This is the reason why some of the examples presented in [7] have not been decomposed. We are currently exploring ways to alleviate this problem by finding new heuristics to solve BR's efficiently.

B. The Cost of Speed Independence

The second part of Table II is an attempt to evaluate the cost of preserving speed independence during the decomposition of an asynchronous circuit. The experiments have been done as follows. For each benchmark, the following script has been run in SIS, using the library `asynch.genlib`: `astg_to_f`; `source script.rugged`; `map`. The resulting netlists could be considered a lower bound on the area of the circuit regardless of its hazardous behavior, since the circuit only implements the correct function for each output signal without regard to hazards. `Script.rugged` is the best known general-purpose optimization script for combinational logic. The columns labeled "lib 1" and "lib 2" refer to two different libraries, one biased toward using latches instead of combinational feedback,[9] the other one without any such bias.

[9] This was chosen due to the common claim that a latch is more "robust," e.g., with respect to meta-stability, than an equivalent combinational standard cell plus an external feedback wire.

The columns labeled `SI` report the results obtained by the method proposed in this paper. Two decomposition strategies have been experimented before mapping the circuit onto the library.

- Decompose all gates into two-input gates (`2 inp`).
- Decompose only those gates that are not directly mappable into gates of the library (`map`).

In both cases, decomposition and mapping preserve speed independence, since we do not use gates (such as MUXes) that may have a hazardous behavior when the select input changes. There is no clear evidence that performing an aggressive decomposition into two-input gates is always the best approach for technology mapping. The insertion of multiple-fanout signals offers opportunities to share logic in the circuit, but also precludes the mapper from taking advantage of the flexibility of mapping tree-like structures. This tradeoff must be better explored in forthcoming work.

Looking at the best results for non-SI/SI implementations, we can conclude that preserving speed independence does not involve a significant overhead. In our experiments we have shown that the reported area is similar. Some benchmarks were even more efficiently implemented by using the SI-preserving decomposition. We believe that these improvements are due to the efficient mapping of functions into latches by using BR's.

VII. Conclusions and Future Work

In this paper we have shown a new solution to the problem of multilevel logic synthesis and technology mapping for asynchronous speed-independent circuits. The method consists of three major parts. Part 1 uses BR's to compute a set of candidates for logic decomposition of the initial complex gate circuit implementation. Thus each complex gate F is iteratively split into a two-input combinational or sequential gate G available in the library and two gates H_1 and H_2 that are simpler than F, while preserving the original behavior and speed-independence of the circuit. The best candidates for H_1 and H_2 are selected for the next step, providing the lowest cost in terms of implementability and new signal insertion overhead. Part 2 of the method performs the actual insertion of new signals for H_1 and/or H_2 into the state graph specification, and resynthesizes logic from the latter. Thus parts 1 and 2 are applied to each complex gate that cannot be mapped into the library. Finally, part 3 does library matching to recover area and delay. This step can collapse into a larger library gate the simple two-input combinational gates (denoted above by G) that have been conservatively used in decomposing complex gates. No violations of speed-independence can arise if the matched gates are atomic.

This method improves significantly over previously known techniques [1], [8], [9]. This is due to the significantly larger optimization space exploited by using 1) BR's for decomposition and 2) a broader class of latches.[10] Furthermore, the ability to implement sequential functions with SR and D latches significantly improves the practicality of the method.

[10] In fact, any sequential gate could be used, including, e.g., asymmetric C elements, the only limit being the size of the space to be explored.

TABLE III
Substitution of Input Signal in Binate Function

$RFGy$	$H(X,y)$	Value of z in SG A'	H'(X,z)
0——	1	1 or 0*	1
000–	0	0 or 1	0
0010	1	1 or 0*	1
0011	0	0 or 1	0
0100	0	0 or 1	0
0101	1	1 or 0*	1
0110	1	1 or 0*	1
0111	1	1 or 0*	1

Indeed one should not completely rely, as earlier methods did, on the availability of C-elements in a conventional library.

In the future we are planning to improve the BR solution algorithm, aimed at finding a set of optimal functions compatible with a BR. This is essential in order to improve the CPU times and synthesize successfully more complex specifications.

Additionally, the methods proposed in this paper and [8] have both been aimed at area minimization. The fact that both methods generate several candidates for the decomposition of each output signal suggests the possibility of defining a tunable cost function, trading-off area and delay, that could improve the quality of the circuit according to the designer's preferences.

Appendix

Proof: (**Lemma 2**)

$\Rightarrow$. In order to prove that SG's $A' = Ins(A, z = H(X,y))$ and $A'' = Ins(A, z = H'(X,z))$ are isomorphic under Conditions 1) and 2), we will show that starting from the same initial state all the corresponding states of A' and A'' have the same set of enabled signals. As the construction of SG proceeds by switching enabled signals, the latter clearly means that A' and A'' have the same sets of states and arcs, i.e., are isomorphic in the graph sense.

The only signal function that is different in A' and A'' is the function for signal z. Therefore, to prove the isomorphism it is sufficient to show that enablings of signal z in all reachable states of A' and A'' are the same.

Let us consider all possible combinations of values for R, F, G, and y and the implied values for $H(X,y)$, z and $H'(X,z)$ in SG A'. They are presented in Table III. Resume by cases.

1) $R = 1$, $F = G = y = x$ (State $s = 1xxx$), where x stands for any value. Clearly, functions $H(X,y)$ and $H'(X,z)$ will exhibit the same behavior in these states of SG A'.
2) $s = 000x$. Then both $H(X,y)$ and $H'(X,z)$ will have value zero independent of z.
3) $s = 0010$. The implied value for $H(X,y)$ is one due to the $G\overline{y}$ part. The implied value for z in SG A' is either 1 or $0*$. Let us consider the state $s1 = 0011$ adjacent to s by signal y. In $s1$ $H(X,y) = 0$ and, therefore, $\delta H(X,y)/\delta y = 1$ in both s and $s1$. Then according to Condition 1 z must be stable in s. If $z = 1$ then $H(X,y) = H'(X,z) = 1$.

4) $s = 0011$. In this state z is stable (see the consideration above) and therefore $H(X,y) = H'(X,z) = 0$.
5) For states 0100 and 0101 we can apply considerations similar to 3) and 4).
6) $s = 0110$. The implied value for $H(X,y)$ is 1. The implied value for z in SG A' is either 1 or $0*$. $s \in F*G*\overline{R}$, then according to Condition 2 z must be at stable 1. In such case $H(X,y) = H'(X,z) = 1$.
7) $s = 0111$. Similar to 6).

Hence, by exhaustive consideration of all possible cases we can conclude that when Conditions (1) and (2) are satisfied the values of $H(X,y)$ and $H'(X,z)$ coincide. This ensures the same enablings of signal z in the states of A' and A''. Therefore A' and A'' are isomorphic.

$\Leftarrow$. Let us consider the consequences of violations of Condition 1) or 2).

1) Assume that Condition 1) is violated. Then in the original SG A there exist two states $s1$ and $s2$ that are different only in the value of signal y and at least for one of them $\delta H(X,y)/\delta y \cdot S^* = 1$. I.e., for $s1$ e.g., $\delta H(X,y)/\delta y = 1$ and $s1 \in ER_A(z+)$ (the case $s1 \in ER_A(z-)$ is similar). From $\delta H(X,y)/\delta y = 1$ it follows that in $s1$ the value of function R ($H(X,y) = Fy + G\overline{y} + R$) should be 0, while one of the functions F or G is at 1.

From $s1 \in ER_A(z+)$ it follows that in SG $A' = Ins(A, z = H(X,y))$ there exist two states $s1'$ and $s1''$ which correspond to $s1$ and such that $s1' \stackrel{z+}{\rightarrow} s1''$. In both these states the value of functions F, G, and R is the same because z is the only signal which is changed and F, G, and R does not depend on z. This means that $H(X,y)$ also has the same value in both states $s1'$ and $s1''$.

Let us consider states $s1'$ and $s1''$ in SG $A'' = Ins(A, z = H'(X,z))$. From $H'(X,z) = z(F+G) + R$ it follows that $H'(X,z)$ has different values in $s1'$ and $s1''$ because of the change of signal z. Hence, in the corresponding states of A' and A'' the functions $H(X,y)$ and $H'(X,z)$ have different values that lead to different enablings of signal z in them. We can conclude that A' and A'' are not isomorphic.

2) Assume that Condition 2) is violated. Then in the original SG A there exists state $s \in ER_A(z+)$ and $F*G*\overline{R} = 1$ in s. s corresponds to states s' and s'' in SG $A' = Ins(A, z = H(X,y))$ where $s' \stackrel{z+}{\rightarrow} s''$. In both these states $H(X,y)$ has the same value, while $H'(X,z)$ has different values. Arguments similar to those used in the previous case prove that A' and A'' are not isomorphic.

■

REFERENCES

[1] P. A. Beerel, C. Myers, and T. H.-Y. Meng, "Covering conditions and algorithms for the synthesis of speed-independent circuits." *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 205–219, Mar. 1998.

[2] R. K. Brayton and F. Somenzi, "An exact minimizer for Boolean relations," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1989, pp. 316–319.

[3] S. M. Burns, "General conditions for the decomposition of state holding elements," in *Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, Aizu, Japan, Mar. 1996, pp. 48–57.

[4] T.-A. Chu, "Synthesis of self-timed VLSI circuits from graph-theoretic specifications," Ph.D. dissertation, Massachusetts Inst. Technol., Cambridge, MA, June 1987.

[5] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "A region-based theory for state assignment in speed-independent circuits," *IEEE Trans. Computer-Aided Design*, vol. 16, pp. 793–812, Aug. 1997.

[6] M. A. Kishinevsky, A. Y. Kondratyev, A. R. Taubin, and V. I. Varshavsky, *Concurrent Hardware. The Theory and Practice of Self-Timed Design*. New York: Wiley, 1993.

[7] A. Kondratyev, J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev, "Technology mapping for speed-independent circuits: Decomposition and resynthesis," in *Proc. 3rd Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, Apr. 1997, pp. 240–253.

[8] ——, "Logic decomposition of speed-independent circuits," *Proc. IEEE*, vol. 87, pp. 347–362, Feb. 1999.

[9] A. Kondratyev, M. Kishinevsky, and A. Yakovlev, "Hazard-free implementation of speed-independent circuits," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 749–771, Sept. 1998.

[10] L. Lavagno and A. Sangiovanni-Vincentelli, *Algorithms for Synthesis and Testing of Asynchronous Circuits*. Norwell, MA: Kluwer Academic, 1993.

[11] F. Mailhot and G. De Micheli, "Algorithms for technology mapping based on binary decision diagrams and on Boolean operations," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 599–620, May 1993.

[12] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. New York: McGraw Hill, 1994.

[13] D. E. Muller and W. C. Bartky, "A theory of asynchronous circuits," in *Ann. Computing Lab. Harvard Univ.*, 1959, pp. 204–243.

[14] S. M. Nowick and D. L. Dill, "Exact two-level minimization of hazard-free logic with multiple-input changes," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1992, pp. 626–630.

[15] E. Pastor, J. Cortadella, A. Kondratyev, and O. Roig, "Structural methods for the synthesis of speed-independent circuits," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 1108–1129, Nov. 1998

[16] P. Siegel and G. De Micheli, "Decomposition methods for library binding of speed-independent asynchronous designs," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1994, pp. 558–565.

[17] P. Siegel, G. De Micheli, and D. Dill, "Automatic technology mapping for generalized fundamental mode asynchronous designs," in *Proc. Design Automation Conf.*, June 1993, pp. 61–67.

[18] S. H. Unger. *Asynchronous Sequential Switching Circuits*. New York: Wiley Interscience, 1969.

[19] P. Vanbekbergen, B. Lin, G. Goossens, and H. De Man,"A generalized state assignment theory for transformations on Signal Transition Graphs," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1992, pp. 112–117.

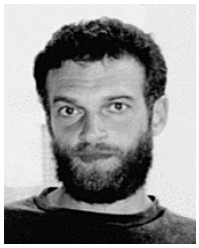
[20] V. I. Varshavsky, M. A. Kishinevsky, V. B. Marakhovsky, V. A. Peschansky, L. Y. Rosenblum, A. R. Taubin, and B. S. Tzirlin, *Self-Timed Control of Concurrent Processes* (Russian edition: 1986). Norwell, MA: Kluwer Academic, 1990.

[21] Y. Watanabe and R. K. Brayton, "Heuristic minimization of multiple-valued relations," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 1458–1472, Oct. 1993.

[22] Y. Watanabe, L. M. Guerra, and R. K. Brayton, "Permissible functions for multioutput components in combinational logic optimization," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 732–744, July 1996.

Jordi Cortadella (S'87–M'88) received the M.S. and Ph.D. degrees in computer science from the Universitat Politècnica de Catalunya, Barcelona, Spain, in 1985 and 1987, respectively.

He is an Associate Professor in the Department of Software of the same University. In 1988, he was a Visiting Scholar at the University of California, Berkeley. His research interests include computer-aided design of VLSI systems with special emphasis on synthesis and verification of asynchronous circuits, concurrent systems, co-design and parallel architectures. He has coauthored over 80 research papers in technical journals and conferences. He has served on the technical committees of several international conferences in the field of Design Automation and Concurrent Systems. He organized the 5th International Symposium on Advanced Research in Asynchronous Circuits and Systems as a Symposium Co-Chair.



Michael Kishinevsky (M'95–SM'96) received the M.Sc. and Ph.D. degrees in computer science from the Electrotechnical University of St. Petersburg, St. Petersburg, Russia.

He has been a Researcher at the St. Petersburg Mathematical Economics Institute Computer Department, Russian Academy of Science, St. Petersburg, Russia, in 1979–1982 and 1987–1989. From 1982 to 1987, he has been with a software company. From 1988 to 1992, he was a Senior Researcher at the R&D Coop TRASSA, St. Petersburg, Russia. In 1992, he joined the Department of Computer Science, Technical University of Denmark, Lyngby, Denmark, as a Visiting Associate Professor. From the end of 1994 until 1998, he was a Professor at the University of Aizu, Aizu-Wakamatsu, Japan. In 1998, he joined the Strategic CAD Labs, Intel Corporation, Hillsboro, OR. His current research interests include design of asynchronous and reactive systems and theory of concurrency. He coauthored two books in asynchronous design and has published over 50 journal and conference papers.

Alex Kondratyev (M'94–SM'97) received the M.S and Ph.D. degrees in computer science from the Electrotechnical University of St. Petersburg, St. Petersburg, Russia in 1983 and 1987, respectively.

He is an Associate professor of the Hardware Department at the University of Aizu, Aizu-Wakamatsu, Japan. From 1988 to 1993, he was with the R&D Coop TRASSA, St. Petersburg, Russia, where he was a Senior Researcher. Previously, he held a position of Assistant Professor in the Electrotechnical University of St. Petersburg. He is a coauthor of *Concurrent Hardware. The Theory and Practice of Self-Timed Design* (New York: Wiley, 1994). He was a co-chair of Async'96 Symposium, co-chair of CSD'98 Conference, and has served as a member of the program committee for several conferences. His research interests include several aspects of the computer-aided design with particular emphasis on asynchronous design and theory of concurrency.

Luciano Lavagno (S'88–M'93) graduated *magna cum laude* in electrical engineering from Politecnico di Torino, Torino, Italy in 1983. In 1992, he received the Ph.D. degree in electrical engineering and computer science from the University of California at Berkeley.

From 1984 to 1988, he was with CSELT Laboratories, Torino, Italy, where he was involved in the ESPRIT 802 CVS project that developed a complete high-level synthesis system. In 1988, he joined the Department of Electrical Engineering and Computer Science of the University of California at Berkeley, where he worked on logic synthesis and testing of synchronous and asynchronous circuits. Since 1993, he has been the architect of the POLIS project (a cooperation between the University of California at Berkeley, Cadence Design Systems, Magneti Marelli, and Politecnico di Torino), developing a complete hardare/software co-design environment for control-dominated embedded systems. Since 1997, he has participated in the ESPRIT 25443 COSY project, developing (based on the POLIS and Felix technologies) a methodology for software synthesis and performance analysis for embedded systems. He has been an Associate Professor at the University of Udine, Udine, Italy, and a research scientist at Cadence Berkeley Laboratories, since 1998. His research interests include the synthesis of asynchronous and low-power circuits, the concurrent design of mixed hardware and software systems, and the formal verification of digital systems. He is the author of a book on asynchronous circuit design, the co-author of a book on hardware/software co-design of embedded systems, and has published over 80 journal and conference papers. He has served on the technical committees of several international conferences in his field (namely, the Design Automation Conference, the International Conference on Computer Aided Design, the European Design Automation Conference).

Enric Pastor received the M.S. and Ph.D. degrees in computer science from the Universitat Politècnica de Catalunya, Barcelona, Spain, in 1991 and 1996, respectively.

He is an Associate Professor at the Department of Computer Architecture of the Universitat Politècnica de Catalunya. He was a Visiting Scholar at the University of Colorado, Boulder, and the Inter-university Microelectronics Centre (IMEC), Leuven, Belgium, in 1992 and 1994, respectively. In 1998, he was a Leverhulme Trust Fellow visiting the University of Newcastle upon Tyne, U.K. His research interests include formal methods for the computer-aided design of VLSI systems with special emphasis on synthesis and verification of asynchronous circuits and concurrent systems.

Alexandre Yakovlev (M'98) received the M.S. and Ph.D. degrees in computer science from the Electrotechnical University of St. Petersburg, St. Petersburg, Russia, in 1979 and 1982, respectively.

He has been working in the area of asynchronous and concurrent systems since 1980. Between 1982 and 1990, he held positions of Assistant and Associate Professor at the Computing Science Department at the Electrotechnical University of St. Petersburg. He visited Newcastle in 1984/1985 for research in VLSI and design automation. After returning to Great Britain in 1990, he worked for one year at the Polytechnic of Wales (now University of Glamorgan). In 1991, he was appointed as a Lecturer at the Newcastle University Department of Computing Science, where he obtained a Personal Readership in Computing Systems Design in 1997. He is leading the VLSI Design Research Group and an Asynchronous Systems Laboratory at Newcastle. His current interests and publications are in the field of modeling and design of asynchronous, concurrent, real-time and dependable systems. He has recently served as a co-organizer of the two international workshops "Hardware Design and Petri Nets" and a program committee co-chair for the 5th International Symposium on Advanced Research in Asynchronous Circuits and Systems.