Logic Decomposition of Speed-Independent Circuits

ALEX KONDRATYEV, SENIOR MEMBER, IEEE, JORDI CORTADELLA, MEMBER, IEEE, MICHAEL KISHINEVSKY, SENIOR MEMBER, IEEE, LUCIANO LAVAGNO, MEMBER, IEEE, AND ALEXANDRE YAKOVLEV

Invited Paper

Logic decomposition is a well-known problem in logic synthesis, but it poses new challenges when targeted to speed-independent circuits. The decomposition of a gate into smaller gates must preserve not only the functional correctness of a circuit but also speed independence, i.e., hazard freedom under unbounded gate delays. This paper presents a new method for logic decomposition of speed-independent circuits that solves the problem in two major steps: 1) logic decomposition of complex gates and 2) insertion of new signals that preserve hazard freedom. The method is shown to be more general than previous approaches and its effectiveness is evaluated by experiments on a set of benchmarks.

***Keywords*—** *Asynchronous circuits, hazards, logic decomposition, speed indepedence, technology mapping.*

I. INTRODUCTION

Asynchronous-circuit design has traditionally been considered as a sort of "black magic" that could be tackled only by hand and at great cost. On the other hand, asynchronous circuits offer a few advantages over their synchronous counterparts that explain a recent revival of interest in asynchronous design techniques. The ultimate goal of asynchronous computer-aided design (CAD) research is to create a design flow that is as easy to use by designers as the standard synchronous synthesis-based flow.

Unfortunately the separation between function and timing, which is the key to the success of synchronous design techniques, is much more problematic in the asynchronous case. Hazards, i.e., unexpected transitions on gate outputs, are not filtered out by letting the combinational logic stabilize before clocking registers, but they must be avoided by a careful design of the logic and of its timing.

Three main classes of asynchronous circuits avoid hazards purely by simple timing assumptions and by logical means, thus preserving as much as possible the above mentioned separation.

1) Fundamental mode circuits [1], [2], [3] assume that the environment of a circuit is so slow that the logic has time to stabilize before inputs can change again. Intuitively a fundamental mode circuit behaves similar to a synchronous one with a clock rate defined by the arrival of input patterns. Therefore the problem of avoiding hazards has a much simpler solution that allows one to apply conventional design methods known from the synchronous world. In particular the decomposition of gates using algebraic techniques does not lead to any circuit malfunctions as long as the fundamental mode assumption remains valid [4]. However, despite being simple and convenient, this assumption reduces the amount of concurrent activity that can take place in a circuit with potentially loosely coupled interfaces.
2) Delay-insensitive circuits [5] make no assumptions on the delays of logic blocks and wires. Such circuits are often synthesized by syntax-directed methods, using nonstandard libraries of relatively large control blocks [6], [7], followed by limited peephole optimizations [8].
3) Speed-independent circuits [9], which can be built by using conventional logic gates, assume that the skew in the delays of fanout branches is smaller than the delays of the logic gates. For this class of

Manuscript received November 30, 1997; revised September 1, 1998. This work was supported in part by ACiD-WG (ESPRIT21949) and CICYT TIC 95-0419.

A. Kondratyev is with University of Aizu, Aizu-Wakamatsu 965 Japan.
J. Cortadella is with the Department of Software, Universitat Politècnica de Catalunya, Barcelona 08034 Spain.
M. Kishinevsky is with Intel Corporation, Hillsboro, OR 97124-5961 USA.
L. Lavagno is with the Dipartimento di Elettronica, Politecnico di Torino, Torino 10129 Italy.
A. Yakovlev is with the Department of Computing Science, University of Newcastle upon Tyne, Newcastle upon Tyne U.K.

Publisher Item Identifier S 0018-9219(99)00888-9.

circuits recent research has developed a variety of logic synthesis techniques that allow one to trade off synthesis speed (as in the case of the fast heuristics developed by [10], [11], and [12]) and optimality (as in the case of the more powerful and expensive techniques developed by [13] and [14]).
Even though the underlying assumption may seem at the same time pessimistic (about gates) and optimistic (about wires), recent results [15] suggest that delay-aware postoptimizations may further improve the quality of the synthesized results. Moreover, delay tuning [16] and low-skew routing [17] may help satisfy the hypothesis about the low wire skew.

The main problem of logic synthesis for speed-independent circuits is that they assume nonstandard implementation libraries, such as arbitrarily complex Boolean gates [18] or arbitrary fanin AND gates [13], [14]. Standard logic decomposition followed by technology mapping is not applicable here, because arbitrary decomposition of a large gate may introduce hazards [1], [4].

This paper is aimed exactly at solving that problem by defining speed-independence-preserving decomposition of large logic gates into smaller ones, carried down to two-input NAND or NOR gates, on which standard technology mapping techniques operate [19]. The proposed decomposition method guarantees that every transition of a new signal resulting from decomposition is acknowledged by some other signal in the circuit, in order to avoid hazards. This is achieved in two major steps:

1) finding a logic decomposition of the complex gate circuit based on algebraic factorization originally proposed for combinational logic [19];
2) inserting new signals whilst preserving overall hazard freedom, based on an idea proposed in [20] and on the efficient implementation techniques described in [21].

Both these parts are provided with an appropriate progress condition check and cost function evaluation, so as to achieve global optimization.

The method has been embedded into the overall synthesis procedure implemented in the software tool "petrify" [22]. This publicly available tool[1] can perform various Petri net manipulations [23], as well as solve the state encoding problem [21] and derive a speed-independent technology-mapped implementation of asynchronous circuits. It contains about 50 000 lines of C code and runs on different UNIX platforms and MS-Windows. Most of the implemented algorithms use symbolic BDD-based representations of the state space [24]. Currently, "petrify" is being used by different universities and industries for their research in asynchronous circuits. Few projects have also been developed with the assistance of "petrify" for the design of control logic. Among them, it is worthwhile to mention the AMULET asynchronous implementation of the ARM microprocessor.

[1] [Online.] Available WWW: http://www.lsi.upc.es/~jordic/petrify.

A. Comparison With Previous Work

The approaches described in [25] and [26] work only under the fundamental mode assumption, which is often too restrictive as discussed above.

The method to perform technology mapping for speed-independent circuits described in [27] decomposes existing gates (e.g., a three-input AND into two two-input AND's), without any further search of the implementation space. It does not explore complex decompositions, which could use multicube divisors, or decompose several gates simultaneously.

The work of [28] treated the decomposition problem for speed-independent circuits in a manner that is similar to the method presented in this paper, by inserting new signals that implement subfunctions of complex gates. However, explicit insertion of new signals is, as we will discuss later, too expensive in the core of an optimization loop. Moreover, efficient filters are required to limit the originally huge search space.

The approach of [29] allows almost any Boolean logic optimization available from the synchronous world and assumes the use of the hazard-absorbing MHS-flops. These special-purpose flip-flops need to be designed very carefully by hand, with extensive failure-analysis tests, before they can be reliably used in practice. Moreover, their correct operation relies on the assumption that NOT gate delays are negligible with respect to AND gate delays (in contrast, most other work in the area assumes that they are smaller than AND gate delays).

Finally, Burns [30] analyzes the correctness conditions for a decomposition of a sequential element that is part of a speed-independent circuit into two sequential elements (or a sequential and a combinational element). Notably, these conditions are analyzed using the original (unexpanded) behavioral model, thus helping the efficiency of the method. Burns' work is, in our opinion, a significant step in the right direction, but it addresses mainly correctness issues. It does not describe how to use the efficient correctness checks in an automated optimization loop, and it does not allow the sharing of a decomposed gate by commondivisor extraction.

The method presented in this paper, on the other hand:

1) allows one to automatically search for a solution aimed at a given library (e.g., with specific maximum gate fanin restrictions);
2) exploits logic sharing based on multiway acknowledgment;
3) performs global optimization via resynthesis (rather than sequential decomposition).

The rest of the paper is organized as follows. Section II contains a theoretical background to facilitate subsequent understanding of the method. Section III presents an overview of the method and a simple example. Section IV provides a more detailed view of the logic decomposition and speed-independent signal insertion algorithms.

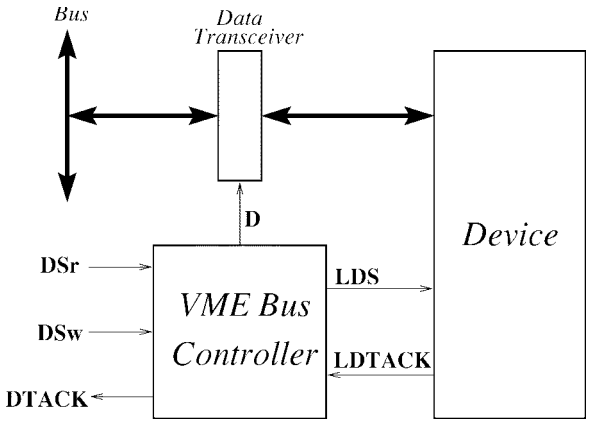


Fig. 1. VME bus controller.

Section V presents results of experiments on a set of benchmarks, obtained from various recent publications on asynchronous circuit synthesis. Section VI concludes the work.

II. Theoretical Background

In this section, an overview of the synthesis flow for speed-independent circuits is presented and illustrated with a design example. Throughout the paper we assume that the reader is familiar with multilevel logic synthesis [19], [31].

A. Circuit Specification

Signal transition graphs (STG's) [18], [32] are a class of interpreted Petri Nets [33], [34] that allow the designer to comfortably capture the behavior of an asynchronous circuit in a manner that is quite similar to timing diagrams.

As an example,consider Fig. 1, which depicts the interface of a device with a VME bus. The behavior of the controller is as follows: a request to read from or write into the device is received by one of the signals, *DSr* or *DSw*, respectively. In a read cycle, a request to read is done through signal *LDS*. When the device has the data ready (*LDTACK*), the controller must open the transceiver to transfer data to the bus (signal D). In the write cycle, data are first transferred to the device (D). Next, a request to write is done (*LDS*). Once the device acknowledges the reception of the data (*LDTACK*), the transceiver must be closed to isolate the device from the bus. Each transaction must be completed by a return-to-zero of all interface signals, seeking for a maximum parallelism between the bus and the device operations.

Fig. 2 shows a timing diagram of the read cycle and Fig. 3 the corresponding STG. All events in this STG are interpreted as signal transitions: rising and falling edges are labeled with "+" and "−" respectively.[2]

An STG has two types of vertices: transitions (denoted by boxes) and places (circles). Places can be marked with tokens (black dots). The set of all places currently marked is called a marking. A transition is enabled if all its input places contain a token. In the initial marking of the STG in Fig. 3 only one transition, *DSr*+, is enabled; LDS+ is not enabled because its input place p_2 does not have a token. Every enabled transition can fire. Firing removes one token from every input place of the transition and adds one token to each of its output places. After the firing of transition DSr+, the net moves to a new marking $\{p_1, p_2\}$ and then $LDS+$ becomes enabled, while other transitions (none in this case) sharing the same input place(s) may be disabled due to the lack of input tokens. Transitions are called concurrent if they both can fire from some marking without disabling each other.

[2]We also use the notation $a*$ if we are not specific about the direction of the signal transition.

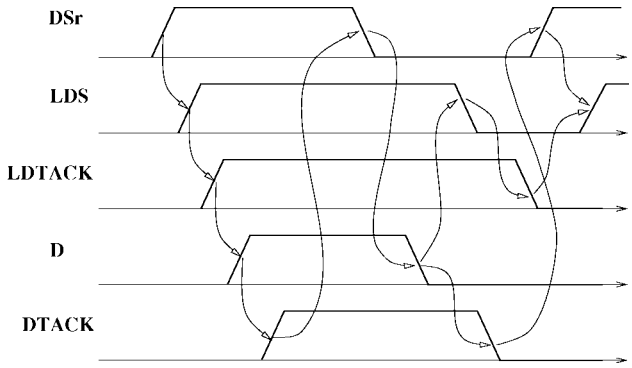


Fig. 2. Waveforms for the READ cycle.

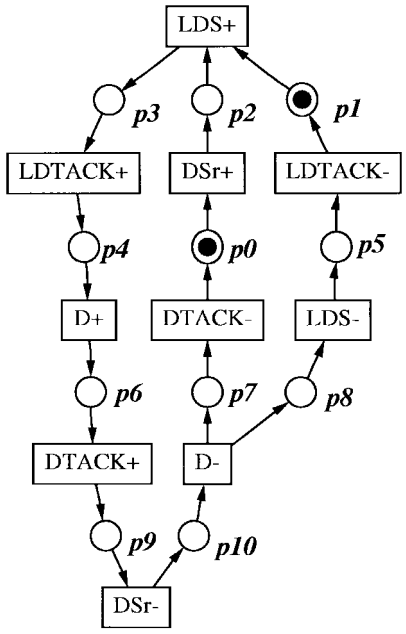


Fig. 3. STG for the READ cycle.

B. State Graphs

Playing the token game one can generate the reachability graph (RG) with vertices corresponding to markings and arcs to transitions between markings. Fig. 4 depicts the RG for the READ cycle of the VME bus controller.

Each state of the RG can also be associated with a binary code of signal values, which label the states in Fig. 4.

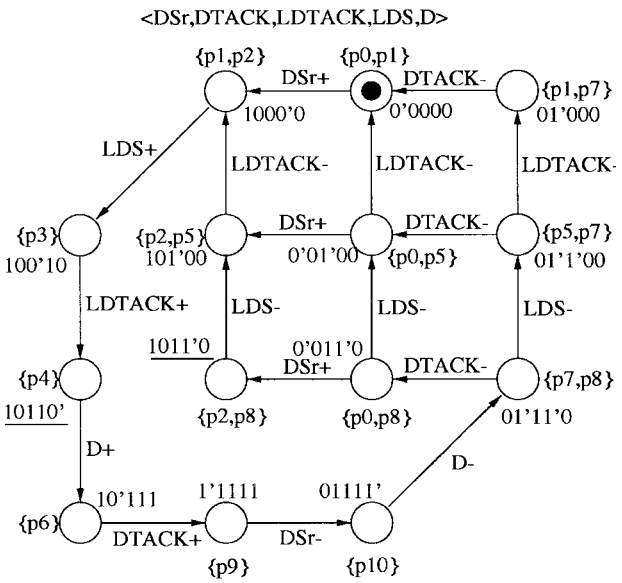


Fig. 4. RG and SG for the READ cycle.

Enabled signals in each state are marked with a prime[3]. An RG with binary encoding is called a state graph (SG) of an STG. State graphs are of primary importance since they form the basis of logic synthesis for asynchronous circuits [18].

C. Implementability Properties

The following properties must hold in an SG to be implementable as a speed-independent circuit [9].

1) *Consistency* holds when rising and falling transitions alternate for each signal.
2) *Completeness of state encoding* ensures that any two states with the same binary code have the same set of enabled noninput signals, i.e., have the same post-behavior.
3) *Speed independence*[4] holds when a) no noninput signal transition can be disabled by another signal transition and b) no input signal transition can be disabled by a noninput signal transition. The former ensures that no short glitches, known as hazards, can appear at the gate outputs, while the latter ensures that no hazards can occur at inputs of the device.

The speed-independence property is often associated with the notion of acknowledgment. Informally, we say that transition b^* acknowledges transition a^* if the fact that b^* fires after a^* has been enabled indicates that a^* has already fired. We say that a^* is acknowledged if any firing sequence starting from a^* enabled is acknowledged by some transition.

[3]Previous work, following [9] used a "*" for the same purpose, but this may cause ambiguity with the notation a^*.

[4]This property does not correspond exactly to Muller's original definition [9], and it should be more properly called output persistency. However, the term "persistency" has been used with several different meanings in the asynchronous circuit literature, and we prefer to use speed independence here.

Table 1
Examples of Next-State Function Values for Signal *LDS* in the SG of Fig. 5

state (s)	$f_{LDS}(s)$
1000′01	1
10′1111	1
0′011′00	0
0′00000	0
010100	–

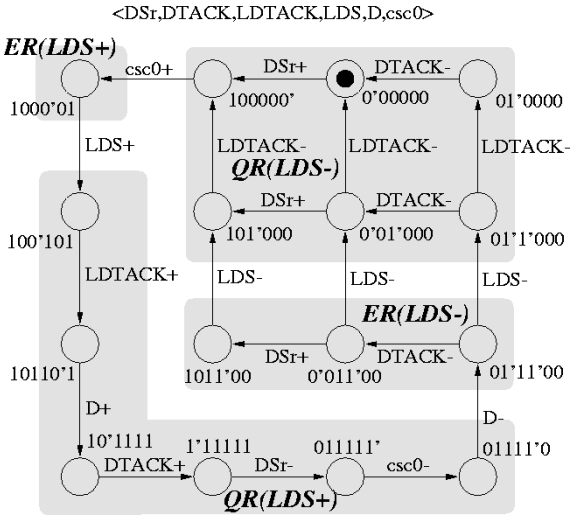


Fig. 5. SG for the READ cycle with complete state coding.

D. Logic Synthesis

The goal of logic synthesis is to derive a gate netlist that implements the behavior defined by the specification. For the sake of simplicity, this step will be illustrated by synthesizing a speed-independent circuit for the read cycle of the VME bus (see Fig. 3).

The main steps in logic synthesis assume that the SG is consistent and speed independent and are as follows:

1) encode the SG in such a way that the complete state coding property holds; this may require the addition of internal signals;
2) derive the next-state functions for each output and internal signal of the circuit;
3) map the functions onto a netlist of gates.

1) Next-State Functions: The next-state function for a signal a is defined as follows. It maps the binary code of each SG state s into:

1) 1 if the signal has value 0′ or 1 in the binary code of s (it is either excited to go to 1, or stable at 1);
2) 0 if the signal has value 1′ or 0 in the binary code of s;
3) —(don't care) for all binary codes that do not correspond to any reachable SG state.

Table 1 presents several examples of values for the next-state function of signal LDS in Fig. 5.

2) Complete State Coding (CSC): The previous definition, however, has a problem, shown by the two underlined states in the SG of Fig. 4. They correspond to different markings, $\{p_4\}$ and $\{p2, p8\}$, but their binary codes are

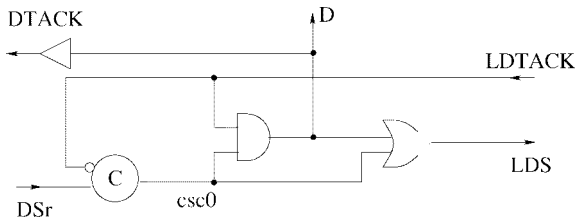


Fig. 6. Implementation with a C-latch.

equal, 10 110. Moreover, enabling conditions in these two states for output signal *LDS* are different. Therefore, the value of the next state boolean function for signal *LDS* for vector 10110 should be 1 (for the first state) and 0 (for the second state). A similar problem holds for signal D.

This is a conflict in the definition of the function. A possible method to solve this problem is to insert new state signals that disambiguate the encoding conflicts. Fig. 5 depicts a new SG in which a new signal, *csc0*, has been inserted. Now the next-state functions for signals *LDS* and D can be uniquely defined. The insertion of new signals must be done in such a way that the resulting SG satisfies consistency and speed-independence, as discussed in [20] and [21].

3) Next-State Function Implementation: Once the next-state function has been derived, Boolean minimization can be performed to obtain a logic equation that implements the behavior of the signal. In this step it is crucial to make an efficient use of the don't care conditions. For the example of Fig. 5, the following equations can be obtained:

$$\begin{aligned} D &= LDTACK \cdot csc0 \\ LDS &= D + csc0 \\ DT\,ACK &= D \\ csc0 &= DSr \cdot \overline{LDTACK} + csc0 \cdot (DSr + \overline{LDTACK}). \end{aligned}$$

The properties of the SG described in Section II-C ensure that any circuit implementing the next-state function of each signal with only one atomic complex gate is speed independent [9] (by atomic gate we mean a gate without internal hazardous behavior). A possible hazard-free gate implementation for the next-state function of the READ cycle example is shown in Fig. 6, where the gate shown as a circle with "C" is a so-called C-element [9] with next state function $c = ab + c(a + b)$.

However, this design flow has an essential problem, because logic functions for signals might be too complex to be mapped into single gates available in the library. Hence the need for decomposition arises.

E. Gate-Level Implementability Without Hazards

In this paper, we develop a decomposition method based on the use of standard architectures. In particular, we concentrate on the standard-C architecture, which is described in Fig. 7(a) (multiple AND-OR gates can exist for both setting and resetting the output). A synthesis method based on this architecture was first suggested in [13] and [35]. This method defines an implementation condition that is equivalent to the monotonic cover (MC) conditions [14] that we use, but only for the case of decomposition into simple gates. In our work we use the more general MC conditions because they allow one to: 1) consider a wider class of specifications (allowing both AND and OR causality) and 2) extend the basic theory to support more aggressive optimizations detailed in [36].

In the rest of this paper we will show how to use only implementable gates, that is gates which exist in the chosen library, instead of the unbounded fanin gates assumed by the above methods.

1) Excitation and Quiescent Regions: Given a signal a, we can classify the states of the SG into the following sets:

1) positive and negative excitation regions (ER's);
2) positive and negative quiescent regions (QR's).

A set of states is called an ER for event a^* (denoted by $ER_j(a^*)$) if it is a maximal connected set of states in which a^* is enabled. Since any event a^* can have several separated ER's, an index j is used for the distinction between different connected occurrences of a^* in the SG.

The QR (denoted by $QR_j(a^*)$) of a transition a^*, with excitation region $ER_j(a^*)$, is a maximal set of states s reachable from $ER_j(a^*)$ such that a is stable (not enabled) in s and s is not reachable from any other $ER_k(a^*)$ $(k \neq j)$ without going through $ER_j(a^*)$. Examples of ER and QR for signal *LDS* are shown in Fig. 5.

2) MC Conditions: Let $C_j(a^*)$ denote one of the first-level AND-OR gates in the standard-C architecture. $C_j(a^*)$ is a correct monotonic poly-term cover for the excitation region $ER_j(a^*)$ if the following three conditions are satisfied.

1) *Cover condition*: $C_j(a^*)$ covers all states of $ER_j(a^*)$ (i.e., $C_j(a^*)$ evaluates to 1 in all states of $ER_j(a^*)$).
2) *One-hot condition*: $C_j(a*)$ does not cover any state outside $ER_j(a*) \cup QR_j(a*)$.
3) *Monotonicity condition*: $C_j(a^*)$ can fall at most once along any state sequence within $QR_j(a^*)$.

The meaning of the MC conditions can be understood by considering the operation of a speed-independent circuit. Suppose, for example, that at some point during circuit operation we enter a state belonging to $ER_j(a^+)$. The cover condition ensures that the gate implementing function $C_j(a^+)$ should go from 0 to 1 in that state. The second condition guarantees that no other gates $C_k(a^+)$ in the signal network of S_a and no gates $C_k(a^-)$ in the signal network of R_a can be at 1 at that moment. Therefore $C_j(a^+)$ is the only gate in the network of signal a with the output value 1 and the propagation of this value to the output of a (through the OR gate and the C-element) gives a complete information on (acknowledges) the switchings in the network. When signal a changes its value from 0 to 1, the circuit moves from the ER $ER_j(a^+)$ into the QR $QR_j(a^+)$. In this region, according to MC, gate $C_j(a^+)$ will be reset and this switching (the only one possible in $QR_j(a^+)$) will be implicitly acknowledged when a will go low.

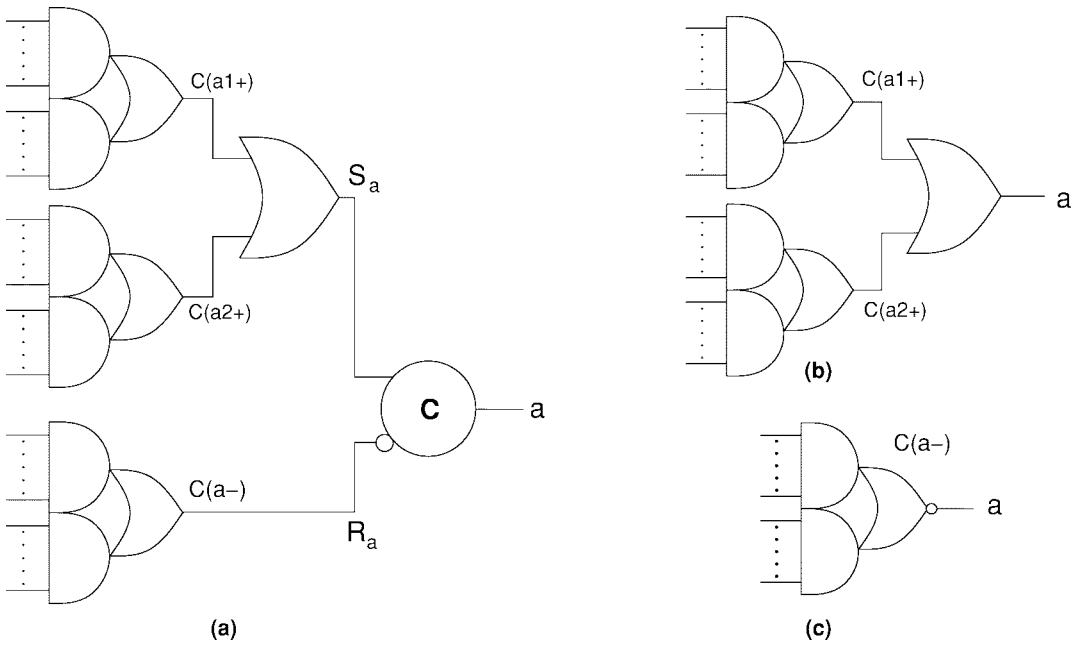


Fig. 7. The standard-C architecture extended for (a) complex gates and (b), (c) its possible optimizations.

Since under these conditions the outputs of the first-level AND gates are one-hot encoded, any valid Boolean decomposition of the second-level OR gates is speed independent.

The standard-C architecture also permits a combinational implementation of a signal. If the set and reset networks are the complements of each other, then a C-element with identical inputs can be simplified to a wire [see Fig. 7(b) and (c)][5].

III. Decomposition Method

As described in Section II, any speed-independent (i.e., output-persistent) SG satisfying the CSC condition can be implemented using the standard-C architecture. This guarantees that a correct boolean equation can be obtained for each cover $C(a*)$. However it does not guarantee that $C(a*)$ can be implemented by one of the gates in the library.

To perform technology mapping, complex gates must be decomposed until all their fragments are mappable onto library gates. The problem of decomposition of combinational circuits is well known, but the methods are not directly applicable to speed-independent circuits. The decomposition of a gate into smaller gates implicitly introduces new internal signals (with delays associated with these new gates) that may cause hazards.

The approach proposed in this work splits the problem of logic decomposition of a gate into two subproblems:

1) combinational decomposition;
2) insertion of a new hazard-free signal.

[5]More precise condition for such an optimization can be formulated as: the set network covers all states of $ER_j(a+) \cup QR_j(a+)$ for all j, or similarly the reset network covers all states of $ER_j(a-) \cup QR_j(a-)$.

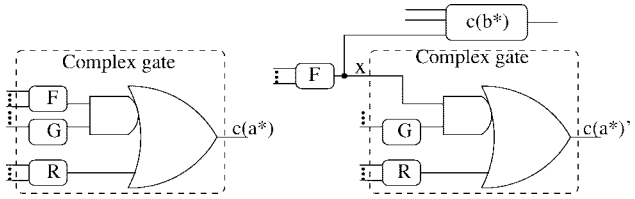


Fig. 8. Algebraic decomposition.

This process is iterated until all gates of the circuit can be mapped onto library gates or no more progress can be achieved, e.g., because no hazard-free decomposition can be found for any of the complex gates. Each subproblem is briefly described in the forthcoming sections.

A. Combinational Decomposition

As is traditionally done in multilevel combinational synthesis, algebraic division has been chosen as the main operation for logic decomposition. For each cover function $C(a*)$ we look for algebraic divisors, aiming at decompositions of the following type: $C(a*) = F \cdot G + R$ where G is the quotient $C(a*)/F$, as shown in Fig. 8. In this figure, $C(a*)$ on the left is an atomic complex gate with function $F \cdot G + R$, while on the right it is an atomic complex gate with (simpler) function $x \cdot G + R$. This decomposition scheme reduces to AND-decomposition when $R = 0$ and OR-decomposition when $G = 1$. Different examples of algebraic division are shown in Table 2.

1) Example: Fig. 9(a) and (b) depicts the STG and the SG of the specification of a circuit. A complex gate implementation of the circuit is shown in Fig. 10(a).

Let us assume that only two-input gates are available in the library. Thus, signals a and b are not directly mappable and must be decomposed. Contrary to synchronous circuits,

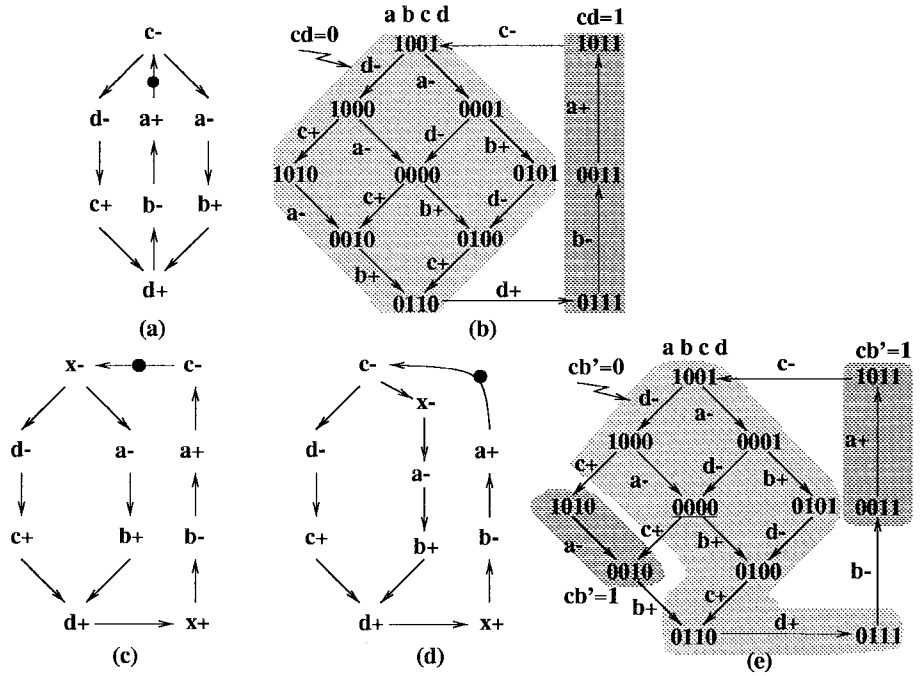


Fig. 9. Signal insertion.

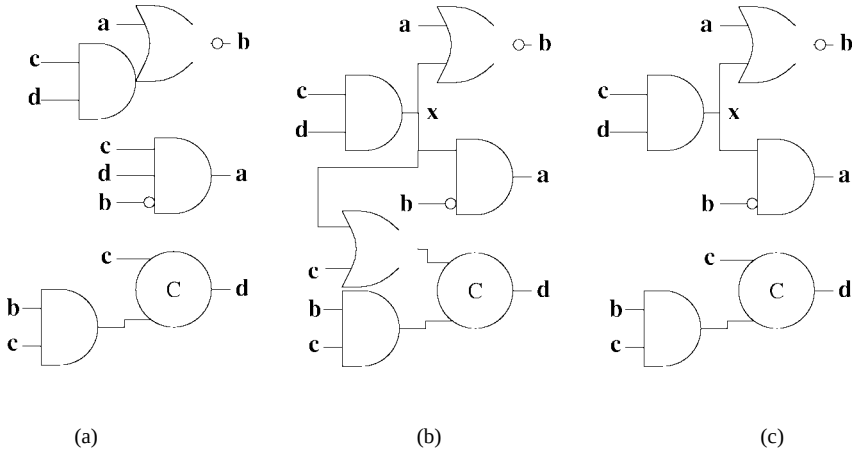


Fig. 10. Implementations of the STG's of Fig. 9.

Table 2
Examples of Algebraic Division

Complex gate	Divisor (x)	New gate
$abc + abd + ef$	ab (co-kernel)	$xc + xd + ef$
$abc + abd + ef$	$c + d$ (kernel)	$abx + ef$
abc	ab (AND-decomp.)	xc
$abd + ef$	abd (OR-decomp.)	$x + ef$

not every algebraic decomposition is valid. Some of them may introduce unavoidable hazards and hence violate the speed-independence requirements. To illustrate this, let us decompose the gate a in Fig. 10(a) by extracting the algebraic divisor $y = cb'$. The ON- and OFF-sets of the function for y are shown in Fig. 9(e) by shadowed areas. When the circuit enters state 0000 [underlined in Fig. 9(e)], two transitions may occur concurrently: $c+$ and $b+$. Firing $c+$ first will enable gate $y = cb'$ to make a transition from low to high, while $b+$ pulls the output of the gate again to low. In a speed-independent circuit, no assumptions can be made about the relative speed of concurrent transitions, and therefore the considered situation is a classical illustration of hazardous behavior on the output of gate y. Hence, the decomposition $y = cb'$ is invalid.

B. Insertion of Hazard-Free Signal

Each divisor of $C(a*)$ is a candidate function to be implemented as a new signal x of the circuit. The new signal will be hazard-free if all its transitions are acknowledged by other signals of the circuit. In the technique presented in this paper, transitions of x may be acknowledged by several signals. This is more general and powerful than

[27] and [30] where transitions of x must be acknowledged locally, only by the same signal a from whose cover x was extracted.

Multiple acknowledgment offers two advantages:

1) the same signal x can be shared by several cover functions (this corresponds to the extraction of common divisors in classical multilevel decomposition);
2) correct speed-independent decomposition can be found even if it does not exist for solutions with single acknowledgments (as shown by the experimental results).

Hazard freedom is guaranteed for the new signal x as follows. Two new events, namely $x+$ and $x-$, are inserted in the SG so that the properties for speed-independent implementability are preserved. The new events are defined in such a way that the implementation of signal x corresponds to the selected divisor for decomposition. If $x+$ and $x-$ can be inserted under such conditions, x is hazard-free. Now x can be used as a new signal in the support of any function cover and contribute to derive simpler equations. Care must be taken not to increase the complexity of other cover functions (Section IV-C).

1) Example (Continued): Let us consider again the example of Fig. 9 and look for a hazard-free decomposition. Among the different algebraic divisors for a and b, there is one that looks especially interesting for a possible sharing of logic: $x = cd$.

The insertion of the events $x+$ and $x-$ must be done according to the implementation of the signal as $x = cd$. The shadowed areas in Fig. 9(b) indicate the sets of states in which the Boolean function cd is equal to 0 and 1, respectively. $x+$ must implement the transition from the states in which cd is equal to 0 to the states in which cd is equal to 1, i.e. $x+$ must be a successor of $d+$, whereas $x-$ must implement the opposite transition and therefore is inserted after $c-$.

Fig. 9(c) and (d) depicts two possible insertions of signal x at the STG level. Both insertions result in specifications that are implementable as different speed-independent circuits (shown in Figs. 10(b) and (c) respectively). Interestingly, both can be implemented with only 2-input gates. However, the insertion of $x-$ as a predecessor of $a-$ and $d-$ [Fig. 9(c)] changes the implementation of signal d, because the fact that $x-$ triggers $d-$ forces x to be in the support of any realization of d. A simpler circuit can be obtained if $x-$ is made concurrent with $d-$ and thus only trigger $a-$ [Fig. 9(d)]. In the resulting circuit, signal x is only in the support of a and b, i.e., of those signals that acknowledge the transitions of x.

Therefore, the insertion of new signals for logic decomposition can be done by exploring different degrees of concurrency with regard to the behavior of the rest of the signals. Finding the best tradeoff between concurrency and logic optimization is one of the crucial problems in the decomposition of speed-independent circuits that can be explored by using our method and that makes it different from previous work (e.g., [28]).

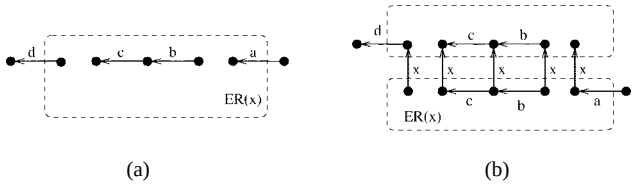


Fig. 11. Event insertion scheme: (a) before insertion and (b) after insertion.

IV. Decomposition Techniques

The generation of divisors for decomposition should be pruned to avoid an explosion of candidates for complex functions.

Two conditions help in constructing an efficient filter of solutions in the huge decomposition space. Only those, decompositions are considered valid which:

1) do not introduce hazards (i.e., preserve speed-independence);
2) heuristically guarantee progress in mapping the circuit to the given library.

The above conditions could be verified in a straightforward (and inefficient) way for every function F used for decomposition, as was proposed in [28]. We could explicitly insert a new signal x, with logic function F, into the original SG and then check whether the modified SG satisfies these conditions. However, there are several reasons that make such a naive approach hardly acceptable. It was already mentioned that for complex functions the number of divisors can be huge. The situation is even worse because another dimension of complexity arises from the fact that for the same function F a new signal $x = F$ can be inserted in many different ways [see, e.g., two different insertions for $x = cd$ in Fig. 9(c) and (d)]. Taking into account that the construction of a new SG is computationally expensive, there is no way to get an efficient implementation by the above straightforward approach.

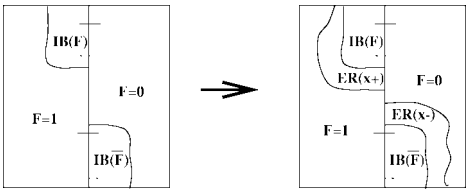
Better results can be obtained if one checks both speed-independence (as proposed in [30] and discussed in Sections IV-A and IV-B) and progress conditions (as discussed in Section IV-C) directly in the original SG.

A. Property-Preserving Event Insertion

Event insertion is the operation on an SG which assigns a subset of states to be an excitation region for a new event. A new event $x*$ can fire from any state of an excitation region $ER(x*)$. Hence in the SG that is obtained after the insertion of a new event $x*$ each state of $ER(x*)$ is split into two: before and after the firing of event $x*$ (see Fig. 11). This operation was defined and implemented in [20], [21] in the context of modifying an SG to satisfy the CSC property.

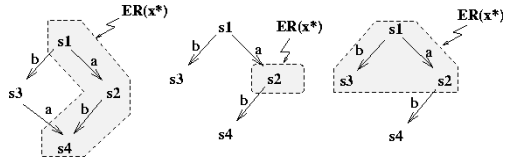
When a new signal $x = F$ is inserted into the SG the value of its logic function F defines a natural bipartition over the set of SG states: $F = 1$ and $F = 0$ [see Fig. 12(a)].

Signal $x = F$ must change when going from the states with $F = 1$ to the states with $F = 0$ and back. The sets of



(a) (b)

Fig. 12. Partition of states induced by a function F.



(a) (b) (c)

Fig. 13. Possible violations of SIP conditions.

states where x changes are called input borders of $F = 1$ or $F = 0$ (denoted by $IB(F)$ or *IB*($\bar{F}$), respectively). Informally, *IB*(F) is the subset of states by which we enter $F = 1$ from $F = 0$ (and vice-versa for *IB*($\bar{F}$)). Input borders are always included into the corresponding excitation regions of x.

When a new signal x is inserted in the SG, care should be taken about preserving:

1) speed-independence;
2) consistency (in any sequence rising and falling transitions of x must alternate), CSC (next-state functions must be well-defined for both old and new signals);
3) input–output (I/O) interface (signal x must be an internal signal of the circuit).

An insertion that satisfies all these conditions will be called valid. Let us consider each requirement of valid insertion separately.

1) Speed-Independence: If the insertion of $x*$ preserves the speed-independence of the SG the corresponding set of states *ER*($x*$) is called a speed-independence preserving (SIP) set. The formal conditions for the set of states *ER*($x*$) to be a SIP set can be given in terms of intersections between *ER*($x*$) and the so-called state diamonds of the SG [21], which are quadruples of states obtained via the interleaving of concurrent events a and b. These conditions are illustrated by Fig. 13, where all possible cases of the illegal intersections of *ER*($x*$) with state diamonds are shown. It has been shown in [21] that any illegal intersection results in the insertion of a new signal with a hazardous behavior. For example, in the case of Fig. 13(b) it results in the invalid decomposition in Fig. 9(e).

It is easy to see that any illegal intersection can be transformed into a legal one by adding states from the relevant diamond into *ER*($x*$). For example, adding state $s1$ to *ER*($x*$) in Fig. 13(b) transforms *ER*($x*$) into a SIP set and the insertion of signal x becomes hazard-free. However, such transformation may change the function for signal x and is not always possible if the function, as in the case of decomposition considered in this paper, is fixed.

2) Consistency: The only consistent changing sequence for x is: $1 \rightarrow 1' \rightarrow 0 \rightarrow 0' \rightarrow 1 \ldots$. Bearing in mind that in any state of *IB*(F) x is going to rise (i.e., these are states in which $x = 0'$), consistency is violated if *IB*(F) is entered from a state s in which $F = 1$ (similar considerations apply to *IB*($\bar{F}$) and $F = 0$). The simplest way to avoid this problem is to expand *IB*(F) by including state s into it. This will make transitions from state s internal to *IB*(F), that is no longer dangerous for consistency.

3) CSC: CSC is guaranteed for the newly inserted signal x, since its excitation and quiescent regions are defined based on a Boolean function F, that is the next-state function of x. It is also easy to show that CSC is not changed for old signals, since:

1) any signal whose transitions are not delayed by x is obviously unaffected;
2) any signal s whose transitions are delayed by x can have CSC conflicts only due to the states which have been split by the insertion of x, but these states have different binary labels in signal x.

4) I/O Interface: When signal x is inserted into the SG using *ER*($x*$), the events with which *ER*($x*$) is exited are delayed until $x*$ fires (see, e.g., event d in Fig. 11). If such an exit event is associated with an input signal, the environment is forced to wait before it can change this input until $x*$ is observed. Signal x thus becomes a primary output of the circuit. This is against the idea of keeping the I/O interface intact during the decomposition. To preserve the I/O interface we can use the same remedy as for consistency violations: whenever input signals exit *ER*($x*$), the latter must be expanded until all the exit signals are noninputs.

B. Finding a Valid Excitation Region

An efficient procedure that finds a valid set *ER*($x+$) [and similarly for *ER*($x-$)] given a function F for x can be organized as follows (see [21] for more details). *ER*($x+$) is initialized to be the input border *IB*(F) of F. If the initial *ER*($x+$) is consistent, preserves I/O interface and is an SIP set, then a valid insertion has been found. If one of the validity conditions does not hold, the *ER*($x+$) is expanded toward the states with $F = 1$.

The expansion is done in such a way that the insertion of the new signal preserves speed independence for all signals. As an example, let us consider *ER*($x*$) in Fig. 13(b). The insertion of $x*$ would not preserve speed independence for b. There are two ways of locally expanding *ER*($x*$) to overcome this problem: 1) by including s_1, thus $x*$ delaying b and being concurrent with a and 2) by including s_4, thus $x*$ being delayed by a and concurrent with b. In the proposed approach only forward expansions are considered and, therefore, the latter would be applied. The expansion is iteratively performed until a satisfactory solution is found.

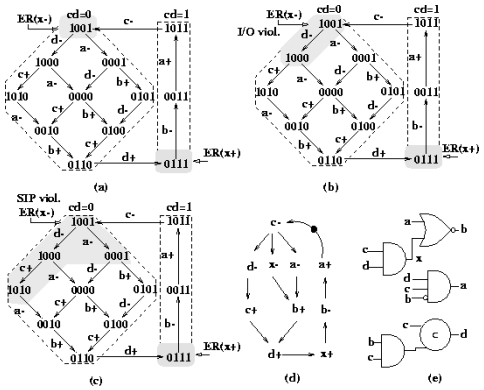


Fig. 14. Selection of valid excitation regions.

As shown in Fig. 12(b), by expanding $ER(x+)$ we can:

1) either obtain a set which is SIP, consistent, and preserving I/O interface (which gives the valid insertion we are looking for);
2) or reach the boundaries of the set of states with $F = 1$ with some remaining violations.

The latter implies that there exists no valid insertion of signal x with the partition implied by logic function F, and the decomposition based on F must be rejected.

Note that the solution found in the expansion of $ER(x+)$ (if any) produces a unique valid $ER(x+)$ which has the minimum size, as shown in [21]. This solution, however, may not be optimal, and further expansion can be applied, e.g., to increase the amount of concurrency for $x+$, while preserving the above conditions.

1) Example (Continued): Fig. 9 shows the decomposition based on the insertion of a new signal x with function $F = cd$. Let us explore different ways to select the excitation regions of x. The set of states with $cd = 0$ is entered through state 1001, while the set with $cd = 1$ is entered through 0111. Hence $IB(\overline{cd}) = 1001$ while $IB(cd) = 0111$. Both input borders are SIP sets, satisfy consistency, and their exit events correspond to output signals. Therefore $IB(\overline{cd})$ and $IB(cd)$ give valid excitation regions for the insertion of signal x, and these regions have the minimum size among all valid insertions. The corresponding STG and implementation of signal x were shown in Figs. 9(c) and 10(b), respectively. The implementation of x requires the acknowledgment of transitions of x by gates a, b, and d. This makes the function of d more complex than in the original SG.

To simplify the implementation let us consider the expansion of $ER(x-)$ within the set of states $cd = 0$.

The first case of expansion is shown in Fig. 14(b), where by including the state 1000 $ER(x-)$ becomes $\{1001, 1000\}$. $c+$ is an exit transition from $ER(x-)$ and c is an input signal. Hence $ER(x-)$ is an invalid selection because it does not preserve I/O interface for the original circuit. The violation can be fixed by adding state 1010 to $ER(x-)$. This gives a valid selection of $ER(x-)$ with the corresponding STG and implementation of signal x shown in Figs. 9(d) and 10(c). This circuit is simpler than that of Fig. 10(b).

$ER(x-)$ can be expanded further, e.g., by including state 0001 as shown in Fig. 14(c). This, however, leads to an illegal intersection with the state diamond $\{1001, 1000, 0001, 0000\}$, which violates speed independence. To solve this problem state 0000 must also be included into $ER(x-)$. After that, $ER(x-)$ illegally intersects the state diamond $\{1000, 1010, 0000, 0010\}$, which in its own turn can be fixed by adding 0010 to $ER(x-)$. The latter gives a valid selection of $ER(x-)$ with the corresponding STG and circuit shown in Fig. 14(d) and (e). This example shows how, starting from $IB(x-)$, the excitation region for transition $x-$ is expanded to satisfy the conditions of Section IV-A. It results in a successful decomposition because in the procedure of expansion no states in which $F = 1$ were required to be included in $ER(x-)$.

We have thus identified a way of finding a correct position in the state graph to insert a new signal for a given Boolean decomposition. In the Section IV-C we shall look at the conditions that heuristically guarantee progress toward the overall goal of decomposing all gates that do not belong to the target library.

C. Progress Analysis

This section investigates a heuristic procedure that explores the huge optimization space by quasi-greedy optimization of a two-level cost function. The cost function is split into two levels in order to make sure that:

1) the newly inserted signal x allows a correct speed-independent factorization of the complex gate; as we will see below, this is not always the case, even if x is speed independent, since excitation region expansion may cause the factored gate to be more complex than expected;
2) other signals do not increase in cost "too much" due to the need to acknowledge the transitions of x; allowing some increase in cost is sometimes necessary in order to escape from local minima.

Both cost estimations must be performed on the original SG, without explicitly adding the new signal, in order to keep the execution time of the decomposition algorithm within reasonable limits. We will call a reduction of the former local progress and a reduction of the latter global progress, and examine each one in turn after a motivating example.

Let the target cover function be $C(a*) = F \cdot G + R$, in which F is the candidate for extraction. At first we should find valid excitation regions for the new signal $x = F$. If such $ER(x+)$ and $ER(x-)$, can be derived, as discussed in Section IV-B, then there is a speed-independent implementation of the SG with a new signal x. The purpose of the insertion of signal x is to simplify the cover function $C(a*) = F \cdot G + R$ by substituting F with x.

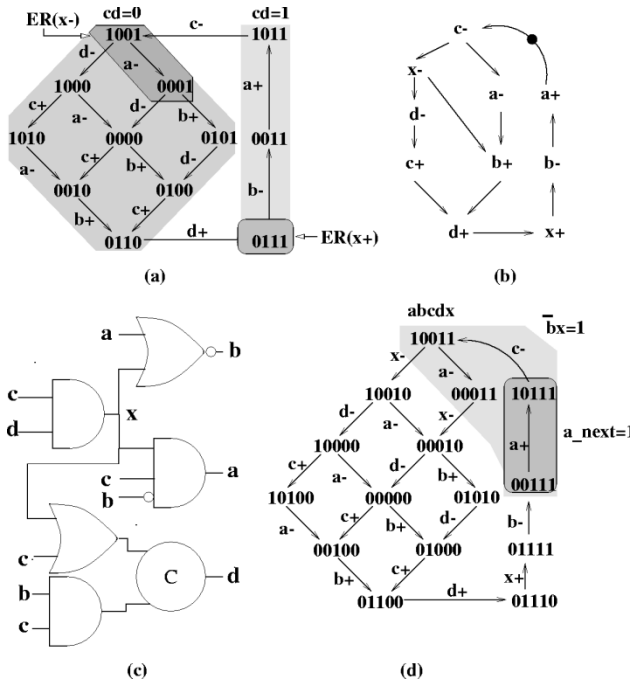


Fig. 15. Solution with no progress.

This purely algebraic simplification is, however, not always possible in the asynchronous case, since in order to preserve speed independence, $C(a*)$ may now require more fanin signals. Let us illustrate this by considering our example.

1) Example (Continued): Fig. 15 shows one of the speed-independent insertions of signal x based on extracting function cd out of the functions for signals a and b for the initial specification given in Fig. 9. This insertion corresponds to selecting excitation regions as follows [see Fig. 15(a)]: $ER(x-) = \{1001, 0001\}$ and $ER(x+) = \{0111\}$. The STG with the new signal inserted, the new state graph, and the new circuit are shown in Fig. 15(b), (d), and (c), respectively.

Although function cd has been extracted, it does not help to reduce the literal count for the target cover of signal a. It still has three literals: $a = \bar{b}cx$, because c must be an input to both gates for a and x. To avoid confusion, note that the circuit with a two input gate implementation for signal a shown in Fig. 10(b) does not correspond to the selected $ER(x-)$, as can be seen by comparing STG's from Fig. 9(c) and Fig. 15(d). In the former, $x-$ precedes $a-$, while in the latter, $a-$ is concurrent with $x-$.

The most natural implementation (based on algebraic factoring) $a = \bar{b}x$ is unfortunately incorrect, as shown in Fig. 15(d), because function $\bar{b}x$ covers four states of the new state graph, while there are only two states in which signal a has an implied value equal to one. Hence function $\bar{b}x$ does not provide a correct cover for signal a.

2) Local Progress Conditions: The local progress condition has the purpose of verifying that the algebraic decomposition is indeed a valid speed-independent decomposition. Thus we must check that $C_{A'}(a*) = x \cdot G + R$ satisfies in the new SG A' all three MC conditions defined in Section II-C. This is true, informally, if and only if the following apply.

1) Cover $x \cdot G$ [effectively, set $ER(x+)$] completely covers the intersection of $ER(a*)$ and $F \cdot G \cdot \bar{R}$. This guarantees that no transition $a*$ is inside $ER(x+)$ and thus ensures the Cover Condition for $C_{A'}(a*)$.
2) Cover $x \cdot G$ does not evaluate to 1 outside the union of $ER(a*)$ and $QR(a*)$. This ensures the one-hot condition for $C_{A'}(a*)$.
3) a) Cover $x \cdot G$ does not change its value from 0 to 1 in the intersection of $QR(a*)$ and $F \cdot G \cdot \bar{R}$. This guarantees the absence of nonmonotonic "1-0-1" transitions along any path in $ER(a*) \cup QR(a*)$ in A'.

   b) No transition of cover $x \cdot G$ from 1 to 0 can occur inside $QR(a*)$, unless R already evaluates to 1. This ensures the absence of nonmonotonic "0-1-0" transitions along any path in $ER(a*) \cup QR(a*)$ in A'.

In the above example function, $\bar{b}x$ is not a correct cover for a because the chosen $ER(x-)$ contains transition $a-$. Due to this $x-$ cannot trigger $a-$ (similarly, transitions of b cannot trigger $a-$). Hence function $\bar{b}x$ cannot implement a trigger signal for $a-$ and signal c should be added for a proper implementation of gate a. Hence no local progress is achieved for the target of the decomposition, gate a.

A more precise formulation of the local progress conditions requires the notion of state image.

A state s' from the expanded SG, A', is said to be an image of a state s from the original SG, A, if the values of all the signals except x are the same in s and in s'. Correspondingly, state s is called the inverse image of s'. The inverse image for any state in A' is unique, but the converse is not true. Each state $s \in ER(x*)$ from SG A has two images s', s'' in A', since we use a signal insertion scheme such that $s' \xrightarrow{x*} s''$. All the other states in A have one image. The image relation is naturally extended to sets of states. If S is a set of states in A', then its inverse image is denoted by S^{-1}. To avoid confusion, we will add subscript A or A' to identify the objects in SG's A and A' if necessary.

The validity of substituting a new signal x in a cover function $C(a*)$ is checked by considering the inverse images of $ER(a*)_{A'}$ and $QR(a*)_{A'}$. By construction, $ER(a*)_A$ is the inverse image of $ER(a*)_{A'}$. The inverse image for QR's can include additional states because some original signal transitions are delayed by x. For example, $QR(d+)_A = \{0111, 0011, 1011\}$ [see Fig. 15(a)] while in the expanded SG $QR(d+)_{A'} = \{01110, 01111, 00111, 10111, 10011, 00011\}$ [see Fig. 15(d)] with an inverse image $QR(d+)^{-1}_{A'} = \{0111, 0011, 1011, 1001, 0001\}$. In general, computing inverse images $QR(a*)$ is easy by starting with $QR(a*)_A$ and expanding it forward inside the next excitation region of signal a, if it has nonempty intersection with $ER(x-)$ or $ER(x+)$.

Table 3
Experimental Results

Circuit	# of gates with n lit. in initial circuit						Added signals/CPU for library with n lit.			Impl. in 2 lit. by Siegel [27]	Area after tech. mapping	
	2	3	4	5	6	7	2	3	4	2	SI	non-SI
alloc-outbound	4	2					1/7	–	–		264	280
chu133	2	2					2/2	–	–	yes	200	208
chu150	3	2					2/2	–	–	no	192	208
converta	5	2					1/2	–	–	no	296	328
dff	2	2					2/4	–	–	yes	176	128
ebergen	5	2					3/2	–	–	no	280	184
half	1	1					1/1	–	–	no	160	168
hazard	2	2					2/1	–	–	yes	208	128
master-read	4	4	1				8/274	1/39	–		624	720
mmu	4	2	1	1	1		n.i.	5/59	2/26		792	536
mp-forward-pkt	2	2					2/3	–	–	yes	232	240
mr0	5	1	2	3	1	1	n.i.	5/130	4/91		1032	904
mr1	3	2	2	1	1		10/126	4/26	3/20		768	656
nak-pa	8	1					1/4.0	–	–	no	328	256
nowick	6	2					2/3	–	–	yes	336	240
pe-rcv-ifc	10	10	3	1			n.i.	3/450	1/203	no	1192	696
pe-send-ifc	10	7	7	1	1		n.i.	n.i.	n.i.			
ram-read-sbuf	4	3					2/8	–	–	yes	320	368
rcv-setup	1		1				2/2	1/1	–	yes	128	120
rpdft		1	3				5/10	2/3	–	yes	528	192
sbuf-ram-write	2	4					4/23	–	–	no	320	336
sbuf-send-ctl	2		1				3/19	1/6	–		240	312
sbuf-send-pkt2	5	4					6/130	–	–	no	296	312
seq_mix	2	3		2			9/247	2/20	1/12		624	576
seq4	1	2		1			4/12	1/4	1/4		464	392
trimos-send	6		3				10/129	3/15	–		640	480
tsend-bm	8	6	4	2		1	n.i.	n.i.	n.i.			
vbe5b	3	1					1/1	–	–	no	192	216
vbe5c		1					1/1	–	–	no	168	176
vbe6a	4		4				8/31	4/11	–		400	608
vbe10b	2	5				1	10/76	3/20	1/6		576	608
wrdatab	5	7	1				7/93	2/11	–		696	616
Total											12672	11192

We will now formulate the local progress condition by presenting conditions for preserving monotonic cover conditions for substituting function F with one literal x in the cover function $C(a*)$. If these conditions, formulated in terms of the original state graph, are satisfied, then simplification for gate implementing signal a is guaranteed to be possible.

For a given event e and a set of states S we define the set of states following immediately after S as follows:

$$after(e,S) = \Big\{ s : s \notin S \wedge (\exists s' \in S : s' \stackrel{\rightarrow}{e} s) \Big\}.$$

We can now more formally state our local progress conditions. Let $C_A(a*) = F \cdot G + R$ be an MC of $ER(a*)$ in SG A. Let $ER(x+)$ and $ER(x-)$ be selected for inserting a signal x. The function $C_{A'}(a*) = x \cdot G + R$ satisfies the three MC conditions in the new SG A', if and only if, as informally explained above[6]:

1) *Cover condition*: $after(a*, (ER(a*) \cap F \cdot G \cdot \bar{R})) \cap ER(x+) = \emptyset$;

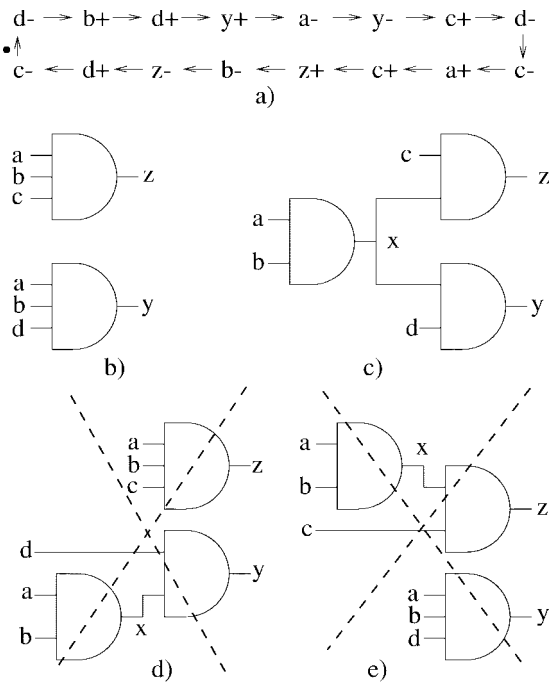


Fig. 16. Example of global acknowledgment: a) STG, b) three-AND gate implementation, c) two-AND gate implementation and d), e) invalid "local" decompositions.

[6] In the formulas we use F, G and H to denote the sets of states in which these functions evaluate to one, and hence we liberally apply Boolean operations, such as $\cdot$, $+$, and set operations, such as $\cap, \cup$, interchangeably.

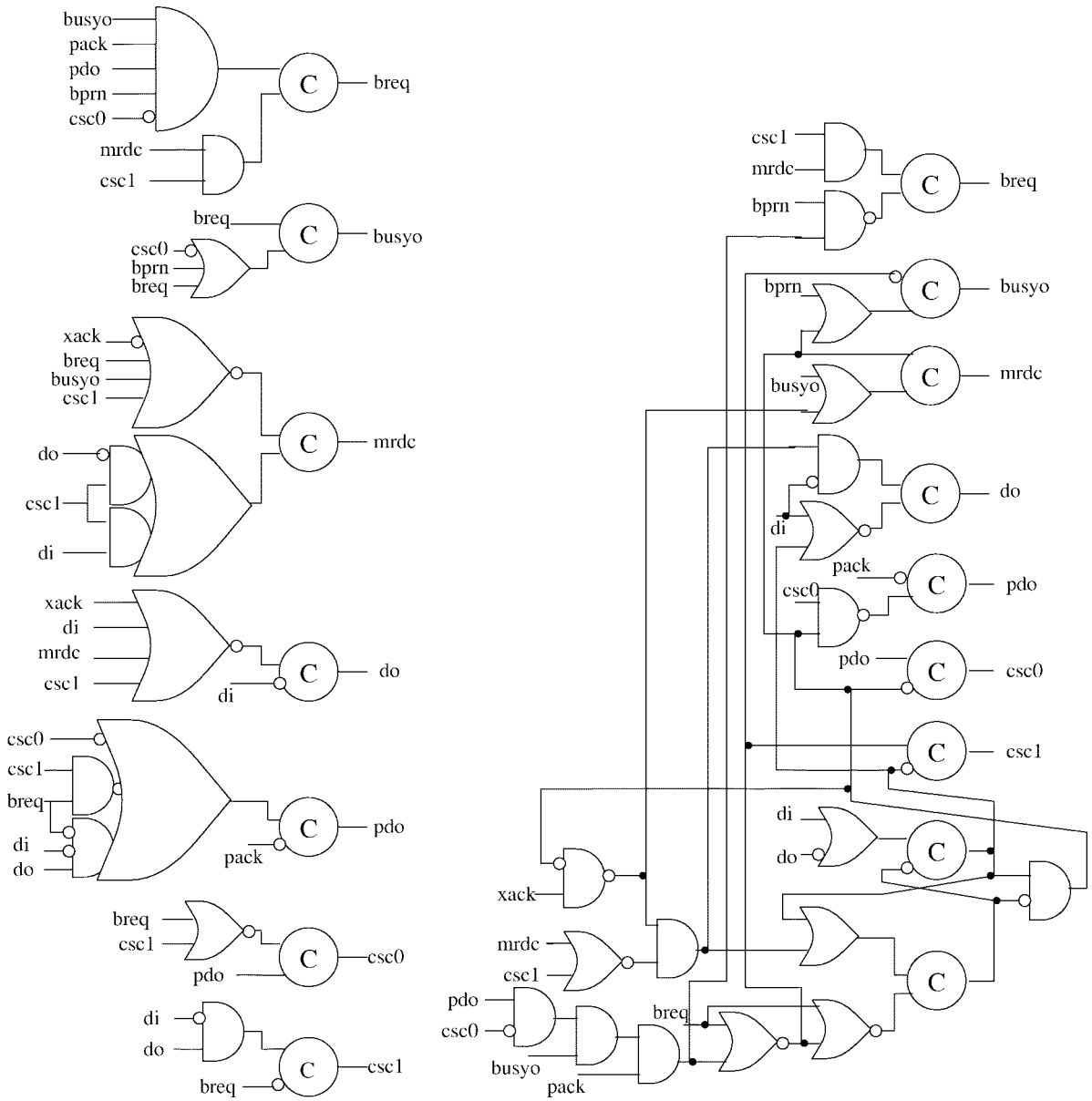


Fig. 17. mr1 before and after logic decomposition into 2-literal gates.

2) *One-hot condition*: for all s: if $s \notin ER(a*) \cup (QR(a*)_{A'})^{-1}$, then $s \notin ER(x-) \cap G$;
3) *Monotonicity conditions*:
   a) $\forall s : s \in (QR(a*) \cap F \cdot G \cdot \bar{R}) \Rightarrow s \notin ER(x+)$;
   b) $\forall s : s \in (QR(a*)_{A'})^{-1} \cap ER(x-) \cap G \Rightarrow Pred(s) \in G + R$.

3) Global Progress Conditions: The local progress conditions (if satisfied) guarantee that the implementation of cover function $C(a*)$ will be simplified as a result of decomposition. However, to accept a decomposition we need to ensure that it does not significantly increase the complexity of logic for other signals. The solutions in Figs. 10(b) and 15(c) increase the size of the cover for d. This may or may not be considered acceptable. In this work, we use a conservative estimate of the increase of logic complexity, based on the calculation of the number of trigger signals before and after the insertion of a new signal. This is based on the facts that:

1) trigger signals of a signal x (signals whose transitions are immediate predecessors of transitions of x) area always in the support of the function implementing x;
2) the function implementing a signal with no new trigger signals after the insertion will never be more complex than the original one.

The reader is referred to [37] for more details and more sophisticated methods for global progress estimation.

V. Experimental Results

The strategy for algebraic decomposition presented above has been implemented in the CAD tool "petrify." Algebraic

decomposition has been applied to a set of benchmarks, and the results are shown in Table 3.

We measured the complexity of each gate as the number of literals required to implement it as a sum-of-product gate, ignoring the cost of input and output inversions. Thus both a two-input EXOR gate $(a\bar{b}+\bar{a}b)$ and a gate implementing function $ab+ac+db+dc=\overline{\bar{a}\bar{d}+\bar{b}\bar{c}}$ are is considered to be four-literal gates.

The improvements obtained by allowing global acknowledgment of signals are illustrated in Fig. 16. For the STG of Fig. 16a), output signals z and y are implemented by three-input AND gates. Global acknowledgment makes it possible to find a decomposition into two-input AND gates, in which both outputs z and y are used to acknowledge the transitions of a new signal x. No valid decomposition preserving speed-independence exists when x is acknowledged by only one output (either y or z). Methods which consider only local acknowledgment (i.e., within a single signal network), such as [6] would fail to find such decompositions.

The first set of columns in Table 3 indicates the complexity of the circuit before decomposition. The second set of columns reports the number of signals inserted for decomposition using gates with at most i literals $(i=2,3,4)$, and the CPU time required to find the decomposition (in seconds, for a SparcStation 20). The number of inserted signals shows also the number of iterations in the decomposition algorithm (the circuit is re-synthesized every time a new signal is inserted). The next column summarizes the results presented by Siegel [27] about the implementability of the circuit with only two-input gates.[7] All decompositions have been independently verified to be speed-independent.

Only five out of the 32 examples were not implementable by our method (which, like all other known methods in the literature, is only heuristic) with two-input gates (entry "n.i." in the table). Only one five-input gate in "pe-send-ifc" and two five-input gates in "tsend-bm" could not be decomposed when attempting to implement these circuits with four-input gates. We significantly improve over the results presented in [27], and only one circuit (pe-rcv-if) could not be implemented with two-input gates from that benchmark suite.

The global-acknowledgment allows the method to effectively decompose complex gates with high fan-in (six or seven literals). This is shown by circuits like mr1 and vbe10b that are implemented with two-input gates. Fig. 17 illustrates this fact, depicting the circuit mr1 before and after logic decomposition into two-input gates.

The final columns present a rough estimation of the cost of speed-independence-preserving logic decomposition. The cost is evaluated by comparing the area (after technology mapping) in the case of logic decomposition that preserves (SI) and does not preserve (non-SI) speed independence respectively. The former is performed by decomposing the circuits into three-literal gates and then mapping onto a gate library. During mapping, small gates can be collapsed to match larger gates in the library without introducing hazardous behavior. The non-SI mapping is performed by SIS by using the following script: `astg_to_f; source script.rugged; map; phase`. In some cases, such as vbe6a, the area of the SI implementation is smaller than the non-SI one, because even non-SI decomposition is just a heuristic technique, and hence our algorithm happens to find a better solution. In most cases, on the other hand, a relatively low area cost (generally less than 15%) is required in order to preserve speed independence.

[7] A more direct comparison with more recent work [6] is difficult because the complexity of implementation is measured in [6] in terms of inputs to FPGA lookup tables, but not in terms of simple gates.

VI. Conclusion

This paper has presented a solution to the problem of logic decomposition of asynchronous speed-independent circuits. The method, implemented in the tool "petrify," is based on a two-step approach. The first step chooses a candidate for decomposition from the set of algebraic divisors of the target function. The second step performs the actual decomposition by implementing the candidate function as a new signal x that can be used to resynthesize the whole circuit. Multiple acknowledgments for x appear automatically at this function generation step and help to guarantee the hazard-freedom of the decomposed function.

References

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

[2] 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.

[3] K. Y. Yun and D. L. Dill, "Automatic synthesis of 3D asynchronous state machines," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1992, pp. 576–580.

[4] D. Kung, "Hazard-nonincreasing gate-level optimization algorithms," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1992, p.631.

[5] F. U. Rosenberger, C. E. Molnar, T. J. Chaney, and T.-P. Fang, "*Q*-modules: Internally clocked delay-insensitive modules," *IEEE Trans. Comput.*, vol. 37, pp. 1005–1018, 1988.

[6] J. C. Ebergen, *Translating Programs into Delay-Insensitive Circuits*. Amsterdam, The Netherlands: Centrum voor Wiskunde en Informatica, 1989.

[7] K. van Berkel, "Handshake circuits: An intermediary between communicating processes and VLSI," Ph.D. dissertation, Technical Univ. Eindhoven, The Netherlands, 1992.

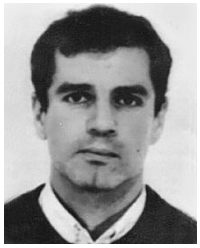
[8] K. van Berkel, R. Burgess, J. Kessels, A. Peeters, M. Roncken, F. Schalij, and R. Wiel, "A single-rail re-implementation of a DCC error detector using a generic standard-cell library," in *Proc. 2nd Working Conf. Asynchronous Design Methodologies*, London, May 1995.

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

[10] P. Vanbekbergen, F. Catthoor, G. Goossens, and H. De Man, "Optimized synthesis of asynchronous control circuits from graph-theoretic specifications," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 1426–1438, Jan. 1993.

[11] K.-J. Lin, C.-W. Kuo, and C.-S. Lin, "Synthesis of hazard-free asynchronous circuits based on characteristic graph," *IEEE Trans. Comput.*, vol. 46, pp. 1246–1263, Nov. 1997.

[12] E. Pastor and J. Cortadella, "Polynomial algorithms for the synthesis of hazard-free circuits from signal transition graphs," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1993, pp. 250–254.
[13] P. A. Beerel and T. H.-Y. Meng, "Automatic gate-level synthesis of speed-independent circuits," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1992, pp. 581–587.
[14] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, and A. Yakovlev, "Basic gate implementation of speed-independent circuits," in *Proc. Design Automation Conf.*, 1994, pp. 56–62.
[15] C. Myers, T. Rokicki, and T. H.-Y. Meng, "Automatic synthesis of gate-level timed circuits with choice," in *Adv. Res. VLSI*, pp. 42–58, Mar. 1995.
[16] L. Lavagno and A. Sangiovanni-Vincentelli, *Algorithms for Synthesis and Testing of Asynchronous Circuits*. Norwell, MA: Kluwer, 1993.
[17] R.-S. Tsay, "An exact zero-skew clock routing algorithm," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 242–249, Feb. 1993.
[18] T.-A. Chu, "Synthesis of self-timed VLSI circuits from graph-theoretic specifications," Ph.D. dissertation, Massachusetts Inst. Technol., Cambridge, June 1987.
[19] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. New York: McGraw-Hill, 1994.
[20] 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.
[21] 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.
[22] ——, "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers," *IEICE Trans. Inform. Syst.*, vol. E80-D, no. 3, pp. 315–325, Mar. 1997.
[23] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev, "Deriving Petri nets from finite transition systems," *IEEE Trans. Comput.*, vol. 47, pp. 859–882, Aug. 1998.
[24] E. Pastor, O. Roig, J. Cortadella, and R. Badia, "Petri net analysis using boolean manipulation," in *Proc. 15th Int. Conf. Application and Theory of Petri Nets*, Zaragoza, Spain, June 1994, pp. 416–435.
[25] 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.
[26] P. A. Beerel, K. Yun, and W. C. Chou, "Optimizing average-case delay in technology mapping of burst-mode circuits," in *Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, Mar. 1996, pp. 244–260.
[27] 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.
[28] P. Beerel, "CAD tools for the synthesis, verification, and testability of robust asynchronous circuits," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1994.
[29] M. Sawasaki, C. Ykman-Couvreur, and B. Lin, "Externally hazard-free implementations of asynchronous circuits," in *Proc. Design Automation Conf.*, June 1995.
[30] S. Burns, "General conditions for the decomposition of state holding elements," in *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, Aizu, Japan, Mar. 1996.
[31] R. K. Brayton, G. D. Hatchel, and A. L. Sangiovanni-Vincentelli, "Multilevel logic synthesis," *Proc. IEEE*, vol. 78, pp. 264–300, Feb. 1990.
[32] L. Y. Rosenblum and A. V. Yakovlev, "Signal graphs: From self-timed to timed ones," in *Proc. Int. Workshop Timed Petri Nets*, Torino, Italy, 1985, pp. 195–207.
[33] T. Murata, "Petri Nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, pp. 541–580, Apr. 1989.
[34] J. L. Peterson, "Petri nets," *ACM Comput. Surv.*, vol. 9, no. 3, Sept. 1977.
[35] P. A. Beerel, Ch. 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.
[36] 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.
[37] 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. Advanced Research in Asynchronous Circuits and Systems*, Eindhoven, The Netherlands, Apr. 1997, pp. 240–253.



Alex Kondratyev (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in computer science from the Electrotechnical University of St. Petersburg, Russia, in 1983 and 1987, respectively.

He is an Associate Professor in the Hardware Department at the University of Aizu, Japan. From 1988 to 1993, he was with the R&D Coop TRASSA, St. Petersburg, where he was a Senior Researcher. Previously, he was an Assistant Professor at the Electrotechnical University of St. Petesburg. He is a coauthor of *Concurrent Hardware: The Theory and Practice of Self-Timed Design* (Wiley, 1993). His research interests include several aspects of computer-aided design, with particular emphasis on asynchronous design and theory of concurrency.

Dr. Kondratyev was Co-Chair of the Async'96 Symposium, Co-Chair of the CSD'98 Conference, and he has been a member of the program committee of several conferences.

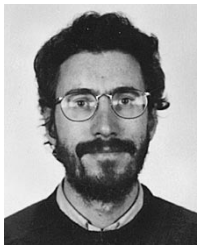
Jordi Cortadella (Member, IEEE) received the M.Sc. and Ph.D. degrees in computer science from the Universitat Politẽcnica de Catalunya, Barcelona, Spain, in 1985 and 1987, respectively.

He is an Associate Professor in the Department of Software, Universitat Politècnica de Catalunya. In 1988, he was a Visiting Scholar at the University of California, Berkeley. His research interests include theory of concurrent systems applied to computer-aided design, with special emphasis on synthesis and formal verification of asynchronous systems and hardware–software codesign. He is also doing research on computer arithmetic and parallel architectures. He has coauthored over 80 research papers in technical journals an conferences.

Dr. Cortadella served on the technical committees of several international conferences in the field of design automation and concurrent systems.

Michael Kishinevsky (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in computer science from the Electrotechnical University of St. Petersburg, Russia.

He was a Researcher at the St. Petersburg Mathematical Economics Institute Computer Department, Russian Academy of Science, in 1979–1982 and 1987–1989. From 1982 to 1987, he was with a software company. From 1988 to 1992 he was a Senior Researcher at the R&D Coop TRASSA. In 1992, he joined the Department of Computer Science, Technical University of Denmark, Lyngby, as a Visiting Associate Professor. From 1994 to 1998, he was a Professor at the University of Aizu, 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.



Luciano Lavagno (Member, IEEE) received the Ph.D. degree in electrical engineering from University of California, Berkeley, in 1992.

From 1984 to 1988 he was with CSELT Laboratories, Torino, Italy, where he was involved in an ESPRIT project that developed a complete high-level synthesis system. In 1988, he joined the Department of Electrical Engineering and Computer Science, University of California, Berkeley, where he worked on logic synthesis and testing of synchronous and asynchronous circuits. He has also been a consultant for various EDA companies, such as Synopsys and Cadence. He is currently an Assistant Professor at the Politecnico di Torino, Italy, and a Research Scientist at Cadence Berkeley Laboratories. 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 he has published over 60 journal and conference papers.

Dr. Lavagno received the Best Paper award at the 28th Design Automation Conference in San Francisco, CA, in 1991. 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, and the European Design Automation Conference.

Alexandre Yakovlev received the M.Sc. and Ph.D. degrees in computing science from the Electrotechnical University of St. Petersburg, Russia.

From 1982 to 1990, he held positions of Assistant and Associate Professor at the Computing Science Department, University of St. Petersburg, where he also worked in the area of asynchronous and concurrent systems since 1980. He first visited Newcastle in 1984–1985 for research in VLSI and Design Automation. After coming back to Britain in 1990, he worked for one year at the Polytechnic of Wales (now University of Glamorgan). Since 1991, he has been a Lecturer, and quite recently a Reader in Computing Systems Design, at the Newcastle University Department of Computing Science, Newcastle upon Tyne, U.K., where he is currently heading the VLSI Design research group. His current interests and publications are in the field of modeling and design of asynchronous, concurrent, and real-time and dependable systems.